

# Extension and Empirical Comparison of Graph-Kernels for the Analysis of Protein Active Sites

Thomas Fober\*, Marco Mernberger\*, Vitalik Melnikov, Ralph Moritz, Eyke Hüllermeier

Department of Mathematics and Computer Science

Marburg University, Germany

{thomas,mernberger,melnikov,moritz,eyke}@mathematik.uni-marburg.de

## Abstract

Graphs are often used to describe and analyze the geometry and physicochemical composition of biomolecular structures, such as chemical compounds and protein active sites. A key problem in graph-based structure analysis is to define a measure of similarity that enables a meaningful comparison of such structures. In this regard, so-called kernel functions have recently attracted a lot of attention, especially since they allow for the application of a rich repertoire of methods from the field of kernel-based machine learning. Most of the existing kernel functions on graph structures, however, have been designed for the case of unlabeled and/or unweighted graphs. Since proteins are often more naturally and more exactly represented in terms of node-labeled and edge-weighted graphs, we propose corresponding extensions of existing graph kernels. Moreover, we propose an instance of the substructure fingerprint kernel suitable for the analysis of protein binding sites. The performance of these kernels is investigated by means of an experimental study in which graph kernels are used as similarity measures in the context of classification.

## 1 Introduction

The functional analysis of proteins is a key research problem in the life sciences and a main prerequisite for resolving the proteome and interactome of living cells, tissues and organisms. Since improved technology has led to an increased number of known protein structures, structure-based prediction of protein function has now become a viable alternative to classical sequence-based prediction methods. In fact, structure-based approaches complement sequence-based methods in a reasonable way, as it is well-known that functional similarity does not necessarily come along with sequence similarity [Gibrat *et al.*, 1996].

Prediction of protein function can be seen as a classification problem. In machine learning, a large repertoire of classification methods has been developed, most of them relying, in one way or the other, on a kind of similarity measure between the objects to be classified. What is needed, therefore, is a measure of similarity between protein structures. More specifically, our focus in this paper will be on the special case of *protein binding sites* derived from crystal structures. To model such structures in a formal way,

we resort to a graph representation which is able to capture the most important geometrical and physicochemical properties of a binding site.

For a long time, graphs have been used in chemoinformatics for the modeling of chemical compounds [Bunke and Jiang, 2000]. In bioinformatics, they are becoming more and more important, too, due to their general versatility in modeling complex structures such as proteins or interaction networks [Berg and Lässig, 2004]. It is hence not surprising that a number of methods has been developed for comparing graphs representing protein structures (e.g. [Jambon *et al.*, 2003; Weskamp *et al.*, 2007; Fober *et al.*, 2009]), and for computing related similarity measures, for example based the concepts of maximum (minimum) common subgraph (supergraph) [Raymond *et al.*, 2002; Raymond and Willett, 2002] or graph edit distance [Neuhaus and Bunke, 2007].

In this context, so-called *kernel functions* (on graphs) have attracted increasing attention in recent years [Gärtner, 2003]. Here, the term ‘kernel’ refers to a class of functions that fulfill certain mathematical properties and can typically be interpreted as similarity measures. These functions are especially attractive as they can be used as a ‘plugin’ for every kernel-based machine learning method. In other words, as soon as a kernel function has been defined on a certain class of objects, the related domain becomes amenable to these methods.

The random walk kernel [Gärtner, 2003] and the shortest path kernel [Borgwardt, 2007] are among the most prominent graph kernels that have been used in the fields of bio- or chemoinformatics. However, as they have originally been defined for unweighted graphs, they are not immediately applicable to the case of graphs modeling protein binding sites. In fact, as will be explained in more detail in Section 2, binding sites are more naturally modeled in terms of graphs with node labels and edge weights, and a representation ignoring labels and weights would come along with an unacceptable loss of information. In Section 3, we therefore extend the aforementioned kernel functions to the case of node-labeled and edge-weighted graphs. Besides, we make use of the *substructure fingerprint representation* [Fechner *et al.*, 2006] to define a class of kernels for protein binding sites. An experimental comparison of these graph kernels will be presented in Section 4 and discussed in Section 5.

## 2 Modeling Protein Binding Sites

To model protein binding sites as graphs, we build upon CavBase [Schmitt *et al.*, 2001, 2002], a database developed for the purpose of identifying and extracting putative

\*These authors contributed equally to the work.

protein binding sites from structural data deposited in the protein database (PDB) [Berman *et al.*, 2000]. CavBase detects putative binding sites as cavities on the surface of proteins by using the LIGSITE algorithm [Hendlich *et al.*, 1997]. The geometry of a protein binding site is internally represented by a set of *pseudocenters*, spatial points that represent the physico-chemical properties of a surface patch within the binding site. Currently, CavBase uses seven types of pseudocenters (donor, acceptor, donor-acceptor, pi, aromatic, aliphatic and metal) that account for different types of possible interactions between residues of the binding site and the substrate of the protein. These pseudocenters are derived from the amino acid composition of the binding site.

As a natural way to model such structures, we make use of node-labeled and edge-weighted graphs. Nodes correspond to pseudocenters and are thus labeled with the pseudocenter type. On average, a graph representation of a binding pocket has around 100 nodes, though graphs with several hundred nodes and some extremes with thousands of nodes do exist.

Edges are weighted by the Euclidean distance between the pseudocenters and thus capture the geometry of the binding site. To reduce the complexity of the representation and increase algorithmic efficiency, we use an approximate representation in which edges exceeding a certain length are ignored; in this regard, a threshold of 11 Angström has proved to be a reasonable choice [Fober *et al.*, 2009]. Despite this approximation, our representation will produce graphs that are rather dense, as approximately 20 percent of all pairs of nodes are connected by an edge. Consequently, the graphs have a large number of cycles. Indeed, a cycle-free representation will normally not be able to reproduce the geometry of a binding site in an accurate way. As will be seen later on, this property leads to problems for certain types of kernel functions.

Formally, a node-labeled and edge-weighted graph will be denoted by  $G = (V, E, l_V, l_E)$ , where  $V$  is a finite set of nodes and  $E \subseteq V \times V$  a set of edges. Moreover,  $l_V : V \rightarrow \mathcal{L}_V$  is a function that maps each node to one among a finite set of labels  $\mathcal{L}_V$ . Likewise,  $l_E : E \rightarrow \mathbb{R}_+$  is a mapping that assigns weights to edges. We define the size of a graph in terms of its number of nodes  $|V|$ . The adjacency matrix of a graph  $G$  will be denoted by  $A$ .

We note that, since our edges are undirected, it would be more correct to use a subset instead of a tuple representation. For convenience, however, we stick to the simpler tuple notation, with the implicit understanding that  $(u, v) \in E$  implies  $(v, u) \in E$  and  $l_E((u, v)) = l_E((v, u))$ .

### 3 Kernels for Node-Labeled and Edge-Weighted Graphs

Let  $\mathcal{G}$  be a set of objects, in our case graphs. A  $\mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$  mapping  $k$  is called kernel if it is symmetric and positive definite, that is,  $k(x, y) = k(y, x)$  for all  $x, y \in \mathcal{G}$  and

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

for all  $m \in \mathbb{N}$ ,  $\{c_1, \dots, c_m\} \subseteq \mathbb{R}$ , and  $\{x_1, \dots, x_m\} \subseteq \mathcal{G}$ .

A generic way to define similarity measures for complex objects, such as graphs, is to use decomposition techniques, that is, to decompose a complex object into a set of simple substructures of a specific type, and to reduce the comparison to the level of these substructures. The idea is that,

for such substructures, the definition of adequate similarity measures is less difficult and, hopefully, the computation more efficient. Therefore, graph kernels often belong to the class of *R-convolution kernels*, a special type of kernel especially suitable for composite objects in a discrete space. Generally, an R-convolution kernel  $k : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$  can be expressed in the following from:

$$k(G, G') = \sum_{g \in R^{-1}(G)} \sum_{g' \in R^{-1}(G')} \kappa(g, g'), \quad (1)$$

where  $R^{-1}(G)$  denotes a decomposition of  $G$  into substructures, and  $\kappa$  is a kernel defined on such substructures. In the following, we consider specific instances of (1).

#### 3.1 Random Walk Kernels

Random walk kernels were introduced in [Gärtner, 2003] for unweighted graphs. Roughly speaking, they decompose a graph into sequences of nodes generated by random walks, and count the number of identical random walks that can be found in two graphs. Thus, the random walk kernel is an R-convolution kernels with substructures given by paths. In the following, we present an extension of these kernels to the case of edge-weighted graphs.

Interestingly, to compute a graph kernel, it is not necessary to sample random walks. Instead, one can exploit an important property of the adjacency matrix  $A$  of a graph  $G$ , namely that  $[A^n]_{i,j}$  is the number of paths of length  $n$  from node  $i$  to node  $j$ ; here,  $A^n$  denotes the  $n$ -th power of  $A$ . Let  $G_\times = G \times G'$  be the product graph of the graphs  $G$  and  $G'$ , where the node and the edge set of  $G_\times$  are defined as follows:

$$\begin{aligned} V_\times &= \{ (v_i, v'_j) \mid v_i \in V, v'_j \in V', l_V(v_i) = l_V(v'_j) \} \\ E_\times &= \{ ((v_i, v'_j), (v_k, v'_l)) \in V_\times \times V_\times \\ &\quad \mid \|l_E(v_i, v_k) - l_E(v'_j, v'_l)\| \leq \epsilon \} \end{aligned}$$

Since  $[A_\times^n]_{i,j}$  now corresponds to the number of equal paths of length  $n$  from node  $i$  to node  $j$  that occur in  $G$  as well as in  $G'$ , the product graph  $G_\times$  allows one to calculate  $k(G, G')$  by performing simple matrix-operations. The requirement that node labels and edge weights have to match along two paths is implicitly encoded in the definition of the product graph (namely by the restriction to node pairs with  $l_V(v_i) = l_V(v'_j)$  and edges with  $\|l_E(v_i, v'_j) - l_E(v_k, v'_l)\| \leq \epsilon$ ); this idea was already used by [Borgwardt *et al.*, 2005], albeit only for discrete edge labels. The similarity of the graphs  $G$  and  $G'$ , considering all equal paths of length 1 to  $\infty$ , is finally given by

$$k_{RW}(G, G') = \sum_{i,j=1}^{|V_\times|} \left[ \sum_{k=0}^{\infty} \lambda_k \cdot A_\times^k \right]_{i,j}, \quad (2)$$

where  $\lambda_k$  is a shrink factor that guarantees convergence of the series. For certain choices of  $\lambda$ , the above series can be calculated in a simple way. Choosing  $\lambda_k = \lambda^k = (1/a)^k$ , with  $a \geq \max_{v \in V_\times} \{\text{degree}(v)\}$ , leads to the geometrical series, and (2) reduces to

$$k_{RW_{geo}}(G, G') = \sum_{i,j=1}^{|V_\times|} [(I - \lambda \cdot A_\times)^{-1}]_{i,j}. \quad (3)$$

Choosing  $\lambda_k = \frac{\beta^k}{k!}$  leads to the exponential series and to

$$k_{RW_{exp}}(G, G') = \sum_{i,j=1}^{|V_\times|} [e^{\beta \cdot A_\times}]_{i,j}.$$

Since the product graph is of quadratic size and matrix inversion has cubic complexity, the complexity of the random walk kernel is  $\mathcal{O}(M^6)$ , with  $M = \max\{|V|, |V'|\}$ .

### 3.2 Shortest Path Kernels

The random walk kernel considers an extremely large number of substructures (paths). Intuitively, this may not only come with a high computational complexity but also produce a certain redundancy. To reduce the number of substructures, Borgwardt [Borgwardt and Kriegel, 2005] proposed to consider only the shortest paths between two nodes, an idea which leads to the shortest path kernel. Again, we propose an extension of this kernel to the case of edge-weighted graphs.

For two nodes  $v_i, v_j \in G$ , let  $sp(v_i, v_j)$  denote the length of the shortest path (sum of edge weights on the path) between these nodes, and let

$$SP(v_i, v_j) = (\{l_V(v_i), l_V(v_j)\}, sp(v_i, v_j)) .$$

Thus, a path is represented by its length and the labels of the start and the end node (while the node labels in-between are ignored). A simple kernel on substructures of this type is the identity (Dirac kernel):

$$\begin{aligned} \kappa_{path}(SP(v_i, v_j), SP(v_k, v_l)) & \quad (4) \\ & = \begin{cases} 1 & \text{if } SP(v_i, v_j) = SP(v_k, v_l) \\ 0 & \text{else} \end{cases} . \end{aligned}$$

Since testing equality is of course not reasonable for real-valued edge lengths, we assume these lengths to be discretized (into bins of length  $\delta = 1$ ).

Now, we can define the generalized shortest path kernel as follows:

$$\begin{aligned} k_{SP}(G, G') & = \quad (5) \\ \frac{1}{C} \sum_{v_i, v_j \in V} \sum_{v_k, v_l \in V'} \kappa_{path}(SP(v_i, v_j), SP(v_k, v_l)) , \end{aligned}$$

where  $C = \frac{1}{4}(|V|^2 - |V|) \cdot (|V'|^2 - |V'|)$  is a normalizing factor that guarantees  $0 \leq k_{SP}(G, G') \leq 1$ .

To analyze the complexity of the shortest path kernel, assume  $|V| = |V'| = M$ . The computation of all shortest paths can be done using the Floyd-Warshall [Floyd, 1962] algorithm in time  $\mathcal{O}(M^3)$ . The results are stored in a shortest path matrix, in which the entry at position  $(i, j)$  gives the cost of the shortest path from node  $i$  to node  $j$ . We consider in a pairwise way all paths in both shortest path matrices and compare them using  $\kappa_{path}$  which needs time  $\mathcal{O}(1)$ . Since there are  $M^4$  comparisons to perform, the shortest path kernel needs time  $\mathcal{O}(M^4)$ .

Representing a path only by its length and the labels of the start and end nodes and, correspondingly, using the Dirac kernel (4) for comparison does obviously come along with a considerable loss of information. To investigate whether performance can be improved by taking the labels of intermediate nodes into account, we developed another extension of the shortest path kernel. More specifically, we replaced the simple 0/1 measure (4) by a measure which compares the complete shortest paths sequence (sps). To this end, an sps  $(v_1, v_2, \dots, v_l)$  is represented in the form of a sequence

$$(l_V(v_1), l_E(v_1, v_2), l_V(v_2), \dots, l_E(v_{n-1}, v_n), l_V(v_n))$$

in which node labels and (discretized) edge lengths occur alternately. To compare such sps, standard methods from

sequence analysis can be used. A well-known approach based on the Levenshtein distance [Levenshtein, 1966] and dynamic programming has a runtime  $\mathcal{O}(l_A \cdot l_B)$ , where  $l_A$  and  $l_B$  is the length of sps  $A$  or  $B$ , respectively. When using appropriate scoring parameters like 1 for a match and 0 for a mismatch and for introducing a gap, this approach leads to a metric and therefore directly to a kernel.

To comply with the requirements of our application, the original dynamic programming approach to sequence alignment was modified as follows. First, recall that our sps involve two types of ‘‘symbols’’, namely node labels and edge weights. To ensure that the former are not aligned with the latter, which is obviously not reasonable, the cost for an assignment of this type was set to negative infinity. Second, note that long paths including many nodes represent more of the structure of a graph than paths of short length. Therefore, we normalize each score (similarity between two shortest paths) by dividing it by the length of the overall longest sps. This leads to an over-weighting of longer sequences since longer sequences are more likely to have higher scores than shorter sequences. In fact, we again obtain a measure with values between 0 and 1, which is used in (5) instead of (4).

In terms of runtime, the above extension of the shortest path kernel is of course very expensive. Again, we have to determine all shortest paths, which can be done in time  $\mathcal{O}(M^3)$ . As explained above, the similarity between these paths is then measured using dynamic programming. Since the length of the sequences is  $\mathcal{O}(M)$ , and since there are  $\mathcal{O}(M^2)$  sequences in both graphs, the total complexity for the pairwise comparison of all sps is  $\mathcal{O}(M^3) + \mathcal{O}(M^2 \cdot M^2 \cdot M^2) = \mathcal{O}(M^6)$ .

### 3.3 Fingerprint Kernels

A very simple type of kernel, which has nevertheless been applied successfully for learning on structured data such as molecular structures [Fechner *et al.*, 2006], is based on the idea of mapping a structured object to a fingerprint vector of fixed length and comparing these vectors afterward. Typically, each entry in this vector informs about the presence or absence of a specific substructure (pattern).

In our case, we consider as substructures all non-isomorphic graphs of size 3. Assuming  $n$  distinct node and  $k$  distinct edge labels, there exist

$$N(n, k) = \binom{n}{3} \cdot k^3 + n(n-1) \cdot k \cdot \binom{k+1}{2} + n \cdot \binom{k+2}{3}$$

substructures of this type, which can be verified by means of a case distinction: (i) All three node labels are distinct: There are  $\binom{n}{3}$  possibilities to choose 3 distinct labels from a set of  $n$  labels. Moreover, since edges are ordered uniquely in this case, there exist  $k^3$  possibilities for the edge labels. (ii) Two node labels are equal and different from the third: There are  $n(n-1)$  possibilities to choose the two labels, one for the identically labeled nodes and one for the other. Assuming an arbitrary ordering on the nodes and edges, an isomorphism can switch the equally labeled nodes so that the ordering of two edges will change, too. To map isomorphic graphs uniquely, we sort the edges, which leads to only  $k \cdot \binom{k+1}{2}$  possible edge combinations. (iii) All nodes have identical label: An isomorphism can reorder all nodes in this case. Therefore, to obtain a unique representation of the possible graphs, all edges must be sorted according to their label. Thus, there are  $n$  possible node labels and  $\binom{k+2}{3}$  edge combinations.

For a graph  $G$ , let

$$f_G = (G \sqsupseteq t_1, G \sqsupseteq t_2, \dots, G \sqsupseteq t_{N(n,k)}) \in \{0, 1\}^{N(n,k)}$$

where  $\{t_1, \dots, t_{N(n,k)}\}$  is the set of all non-isomorphic subgraphs of size 3, numbered in an arbitrary but fixed order. The predicate  $G \sqsupseteq t_i$  tests whether  $t_i$  is contained in  $G$  and, by convention, returns 1 if it evaluates to true and 0 otherwise. To compare two graphs  $G$  and  $G'$  in terms of their respective fingerprint vectors  $f_G$  and  $f_{G'}$ , different kernels can be used. The simplest approach is to look for the Hamming distance of the two vectors, which leads to

$$k_{FPH}(G, G') = \frac{1}{N(n,k)} \sum_{i=1}^{N(n,k)} \kappa_\delta([f_G]_i, [f_{G'}]_i) , \quad (6)$$

where  $[f_G]_i$  denotes the  $i$ -th entry in the vector  $f_G$ , and  $\kappa_\delta$  is the Dirac kernel (i.e.,  $\kappa_\delta(x, y) = 1$  if  $x = y$  and  $= 0$  if  $x \neq y$ ). As a potential disadvantage of this approach, note that it does not only reward the co-occurrence of a substructure in both graphs, but also the simultaneous absence: If the  $i$ -th pattern neither occurs in  $G$  nor in  $G'$ , then  $\kappa_\delta([f_G]_i, [f_{G'}]_i) = \kappa_\delta(0, 0) = 1$ , which may not be desirable. An alternative measure avoiding this problem is the well-known Jaccard coefficient:

$$k_{FPJ}(G, G') = \frac{\sum_{i=1}^{N(n,k)} \min([f_G]_i, [f_{G'}]_i)}{\sum_{i=1}^{N(n,k)} \max([f_G]_i, [f_{G'}]_i)} . \quad (7)$$

Our current implementation of the fingerprint approach is a naive one, in which testing the presence of a substructure in a graph  $G$  has complexity  $O(M^3)$ , with  $M = |V|$  the number of nodes in  $G$ . Thus, the overall complexity of computing  $k(G, G')$  is  $O(N(n, k) \cdot M^3)$ , with  $M = \max(|V|, |V'|)$ . However, we can utilize the fact that we can abort the search for a substructure as soon as we have found the first occurrence.

### 3.4 Kernels based on Fuzzy Fingerprints

The discretization of edge weights needed for the previous approach can be criticized for several reasons. Some of the disadvantages, notably the abrupt transition between the presence and absence of a subgraph due to a very small change of an edge length, can be avoided by means of a fuzzy discretization. A fuzzy partition of a domain  $X$  (in our case  $\mathbb{R}_+$ ) is defined by a finite family of fuzzy subsets  $F_1, F_2, \dots, F_k$  of  $X$  such that  $\sum_{i=1}^k F_i(x) > 0$  for all  $x \in X$ ; typically, one even requires that  $\sum_{i=1}^k F_i(x) = 1$  for all  $x \in X$ . Concretely, we shall use a fuzzy partition in which  $F_i$  is defined by

$$F_i(x) = \max\{0, 1 - |x - i|\} .$$

Thus,  $F_i$  can be interpreted as the fuzzy subset of numbers ‘‘approximately equal to  $i$ ’’.

A pattern  $t$  is now a graph of size 3 whose nodes are labeled as before, but whose edges are labeled with fuzzy numbers of the form  $F_i$ . A subgraph  $S$  of a graph  $G$  with real-valued edge lengths can be isomorphic to a pattern  $t$  to a certain degree. Let  $a_i$  be the label of the  $i$ -th node in  $S$ , and  $x_{ij}$  the length of the edge between node  $i$  and node  $j$ . Likewise, let  $b_i$  be the label of the  $i$ -th node in  $t$ , and  $F_{ij}$  the length of the edge between node  $i$  and node  $j$ . The degree of isomorphism of  $t$  and  $S$ , denoted  $[t \sim S]$ , is then given by

$$\max_{\pi \in S_3} \begin{cases} \min\{F_{12}(y_{12}), F_{13}(y_{13}), F_{23}(y_{23})\} & \text{if } M(\pi) \\ 0 & \text{otherwise} \end{cases}$$

where  $y_{ij} = x_{\pi(i), \pi(j)}$ ,  $S_3$  is the set of all permutation  $\{1, 2, 3\} \rightarrow \{1, 2, 3\}$ , and  $M(\pi)$  is true if

$$(a_1 = b_{\pi(1)}) \wedge (a_2 = b_{\pi(2)}) \wedge (a_3 = b_{\pi(3)})$$

and false otherwise. Likewise, the degree to which  $t$  is present in the graph  $G$  is then given by

$$[G \sqsupseteq t] = \max_S [t \sim S] , \quad (8)$$

where the maximum is taken over all subgraphs of size 3 in  $G$ . This value defines the entry for the pattern  $t$  in the (fuzzy) fingerprint vector  $[f_G]$  of  $G$ .

In the non-fuzzy approach, the search for a pattern  $t$  in a graph  $G$  can be stopped as soon as the pattern has been found. Here, this is no longer possible, since the maximum (8) has to be determined. To accelerate the calculation of this maximum, we make use of a canonical form describing graphs of size 3. To this end, we distinguish three cases:

- All node labels are equal. In this case, the canonical form is given by the node label followed by the edge lengths in increasing order.
- Two nodes have an identical label. The canonical form starts with the node label that appears once in the graph followed by the label that appears twice, the edge weight between the nodes with the same label, and finally the remaining two edge weights in increasing order.
- All nodes have different labels. The canonical form is then defined by the three occurring labels, sorted in a lexicographic order, the edge length between the first and the second, the second and the third, and finally the first and the third node.

All three cases are illustrated by an example in Fig. 3.4.

We denote the set of all canonical forms by  $\Sigma$ . The above representation enables the definition of a bijective function  $i : \Sigma \rightarrow \{1, \dots, N(n, k)\} \subset \mathbb{N}$  assigning a unique number to each each form and, therefore, subgraph of size 3. Using this mapping, the calculation of the fingerprint vector for a graph  $G = (V, E)$  can be done in a more efficient way. Instead of counting, for each entry  $i$  in the fingerprint vector, how often subgraph  $g_i$  appears in  $G$ , we can enumerate all subgraphs of size 3 in  $G$ . For each subgraph  $g_i$  of  $G$ , we perform the transformation to its canonical form  $\sigma_i$  (in time  $\mathcal{O}(1)$ ), evaluate the function  $i(\sigma_i)$  to determine the position of  $g_i$  in the fingerprint vector (in time  $\mathcal{O}(1)$ ), and finally update the entry at this position in the vector. Doing this for all  $\binom{M}{3} = \mathcal{O}(M^3)$  subgraphs of size 3 leads to a runtime of  $\mathcal{O}(M^3)$  in comparison to  $\mathcal{O}(M^3) \cdot N(n, k)$ , where  $N(n, k)$  is usually a large number (in our case 36,729).

The fingerprint vectors eventually obtained are ‘‘fuzzy’’ in the sense of having entries in the unit interval  $[0, 1]$  instead of being either 0 or 1. From a machine learning point of view, this is interesting as it may increase discriminatory power. To compare the vectors, we again use the approach based on the Jaccard coefficient in (7), where we substitute the logical operators by a t-norm and t-conorm, respectively:

$$k_{FFP}(G, G') = \frac{\sum_{i=1}^{N(n,k)} \top([f_G]_i, [f_{G'}]_i)}{\sum_{i=1}^{N(n,k)} \perp([f_G]_i, [f_{G'}]_i)} .$$

For the experiments we used  $\top(a, b) = \min(a, b)$  and  $\perp(a, b) = \max(a, b)$ .

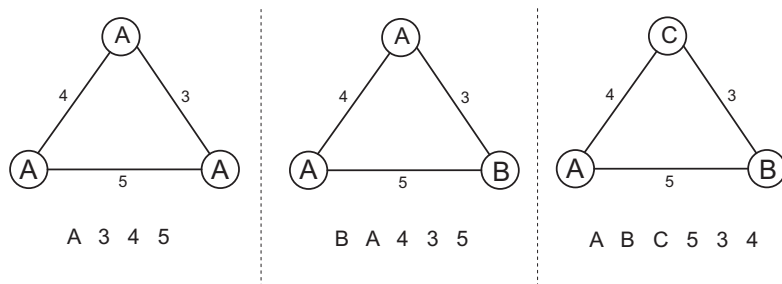


Figure 1: The three possible cases that can occur: all labels identical, two labels identical and all labels unique.

## 4 Experimental Evaluation

In our experiments, we compared the graph kernels discussed in the previous section, namely the random walk kernel (RW) using (3) with  $a$  given by the maximum size of the graphs in the data set (plus 1), the shortest path kernel (SP) and its extended version based on sequence alignment (SPSA), the fingerprint kernel based on (6) and (7), respectively (FPH and FPJ), and the fuzzy fingerprint-kernel (FFP). Moreover, to get an idea of their absolute performance, we additionally included two state-of-the-art methods for comparing protein binding sites in terms of their similarity. Both approaches are based on the concept of a *graph alignment* that has been introduced in [Weskamp *et al.*, 2007]. The first method (GA) is the original algorithm proposed in the same paper, which is based on a heuristic (greedy) optimization strategy. The second method (GAVEO) makes use of evolutionary optimization techniques to compute a graph alignment [Fober *et al.*, 2009]. Both methods need a number of parameters, which we defined as recommended in [Weskamp *et al.*, 2007]. For the kernel methods, we set the parameter  $\epsilon$  (tolerance for edge length comparison) to 0.2.

The assessment of a similarity measure for molecular structures, such as protein binding sites, is clearly a non-trivial problem. In particular, since the concept of similarity by itself is rather vague and subjective, it is difficult to evaluate corresponding measures in an objective way. To circumvent this problem, we propose to evaluate similarity measures in an indirect way, namely by means of their performance in the context of nearest neighbor (NN) classification. The underlying idea is that, the better a similarity measure is, the better the predictive performance we expect from an NN classifier using this measure for determining similar cases.

### 4.1 Data

We selected two classes of binding sites that bind to NADH or ATP, respectively. This gives rise to a binary classification problem: Given a protein binding site, predict whether it binds NADH or ATP. More concretely, we compiled a set of 355 protein binding pockets representing two classes of proteins that share, respectively, ATP and NADH as a cofactor. To this end, we used CavBase to retrieve all known ATP and NADH binding pockets that were co-crystallized with the respective ligand. Subsequently, we reduced the set to one cavity per protein, thus representing the enzymes by a single binding pocket. As protein ligands adopt different conformations due to their structural flexibility, it is likely that the ligands in our data set are bound in completely different conformations, hence the corresponding binding pockets do not necessarily share much structural similarity. To ensure a minimum level of similar-

ity, we therefore utilized the ligand information available for these binding pockets, as these structures were all co-crystallized with the corresponding ligand. Using the Kabach algorithm [Kabach, 1976], we calculated the root mean squared deviation (RMSD) between pairs of ligand structures and combined all proteins whose ligands yielded a RMSD value below a threshold of 0.4, thus ensuring that the ligands are roughly oriented in the same way. This value was chosen as a trade-off between data set size and similarity. Eventually, we thus obtained a two-class data set comprising 214 NADH-binding proteins and 141 ATP-binding proteins.

### 4.2 Results

The performance of the different methods, using an SVM (LIBSVM implementation) and a simple k-nearest neighbor classifier ( $k = 1, 3, 5, 7, 9$ ) for prediction, is summarized in Table 1. Since all methods obviously fulfill the kernel properties with the exception of GA and GAVEO, these methods can be used as precomputed kernel functions for the LIBSVM package.

Table 2 shows the average time complexity of the methods, namely the time needed for a single pairwise comparison of two structures. These numbers have been determined by averaging over 1000 comparisons with randomly chosen structures.  $FFP_{index}$  gives the runtime of the fuzzy-fingerprint kernel that uses the index-function described above. Note the significant improvement regarding runtime in comparison to the fuzzy-fingerprint kernel without the index function.

## 5 Discussion and Conclusion

The results convey a relatively clear picture: The fingerprint kernels perform best, the random walk and shortest path kernel worst, and the graph alignment methods are in-between. The overall best results are achieved by the Jaccard-variant of the fuzzy fingerprint kernel. In terms of efficiency, the fingerprint kernels are superior, too. Thus, this type of kernel is clearly of high interest in the context of comparing protein binding sites.

The poor performance of the random walk and shortest path kernels is a bit astonishing at first sight. Our extension of the shortest path kernel based on sequence comparison yield some improvement when using SVM as a classifier, though it is still not competitive with the fingerprint kernels. The failing of these types of kernel can possibly be attributed to their characteristics as R-convolution kernels. In general, the ‘all-against-all’ comparison of substructures performed by kernels of this type appears to be problematic for large graphs or, more generally, for diverse objects consisting of many substructures. It leads to a kind of averaging effect, and indeed, we observed that the entries in

Table 1: Classification rates of an SVM and a k-nearest-neighbor classifier in a leave-one-out cross validation using different values of  $k$  and different similarity measures: random walk kernel (RW), shortest path kernel (SP), shortest path kernel based on sequence alignment (SPSA), fingerprint kernel (FPH, FPJ), fuzzy fingerprint kernel (FFP), and graph alignment (GA, GAVEO).

| method | RW    | SP    | SPSA  | FPH   | FPJ   | FFP   | GA    | GAVEO |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|
| SVM    | 0.606 | 0.625 | 0.707 | 0.916 | 0.907 | 0.916 | —     | —     |
| k = 1  | 0.597 | 0.606 | 0.620 | 0.828 | 0.842 | 0.879 | 0.766 | 0.789 |
| k = 3  | 0.597 | 0.628 | 0.546 | 0.839 | 0.882 | 0.887 | 0.718 | 0.766 |
| k = 5  | 0.597 | 0.634 | 0.552 | 0.839 | 0.873 | 0.887 | 0.724 | 0.780 |
| k = 7  | 0.608 | 0.625 | 0.566 | 0.819 | 0.859 | 0.854 | 0.718 | 0.786 |
| k = 9  | 0.608 | 0.634 | 0.597 | 0.814 | 0.836 | 0.839 | 0.713 | 0.766 |

Table 2: Average runtime and standard deviation (in seconds) of the different methods for a single pairwise comparison.

| method  | RW            | SP           | SPSA             | FP          |
|---------|---------------|--------------|------------------|-------------|
| runtime | 65.51 ± 89.07 | 9.75 ± 97.77 | 574.43 ± 4089.70 | 2.05 ± 3.66 |

| method  | FFP            | FFP <sub>index</sub> | GA              | GAVEO   |
|---------|----------------|----------------------|-----------------|---------|
| runtime | 53.99 ± 121.01 | 17.90 ± 66.34        | 121.74 ± 418.02 | > 5 min |

our kernel matrix are all very similar. Moreover, in the random walk kernel, nodes and edges can appear more than once in a random walk, a problem known as *tottering*. This problem becomes especially severe in the presence of many cycles within a graph, a property which, as mentioned earlier, our graph descriptors of protein binding sites will inevitably exhibit. The shortest path kernel avoids tottering but has another problem known as *halting*: As it only looks at shortest paths, it tends to be dominated by a large number of paths with very few nodes. As we consider graphs representing geometric constraints within a binding pocket, this is likely to result in a loss of information.

The strong performance of the fingerprint kernel suggests to elaborate on this approach in more detail. In fact, the approach presented in this paper is rather simple and can be extended in different ways. First, substructures other than subgraphs of size 3 might be considered, even though our experience so far has shown that this class of patterns is able to capture considerable information while still being manageable in terms of complexity. Second, the fingerprint vectors could be constructed (and compared) in a more sophisticated way. For example, instead of just indicating the presence or absence of a pattern, one may count its number of occurrences and then apply similarity measures for frequency vectors.

## References

- Johannes Berg and Michael Lässig. Local graph alignment and motif search in biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(41):14689–14694, 2004.
- H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, , and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.
- K. M. Borgwardt and H. P. Kriegel. Shortest-path kernels on graphs. In *International Conference on Data Mining*, pages 74–81, Houston, Texas, 2005.
- Karsten Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(21):i47 – i56, 2005.
- K. M. Borgwardt. *Graph Kernels*. PhD thesis, Ludwig-Maximilians-Universität München, Germany, 2007.
- Horst Bunke and Xiaoyi Jiang. Graph matching and similarity. *Intelligent systems and interfaces*, 15:281 – 304, 2000.
- N. Fechner, G. Hinselmann, and A. Zell. Implicitly defined substructure fingerprints for support vector machines. In *German Conference on Chemoinformatics*, 2006.
- R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.
- Thomas Fober, Marco Mernberger, Gerhard Klebe, and Eyke Hüllermeier. Evolutionary construction of multiple graph alignments for the structural analysis of biomolecules. *Bioinformatics*, 2009.
- Thomas Gärtner. A survey of kernels for structured data. *SIGKDD Explorations*, 5(1):49 – 58, 2003.
- J. F. Gibrat, T. Madej, and S. H. Bryant. Surprising similarities in structure comparison. *Current Opinion in Structural Biology*, 6(3):377–385, 1996.
- M. Hendlich, F. Rippmann, and G. Barnickel. LIGSITE: Automatic and efficient detection of potential small molecule-binding sites in proteins. *Journal of Molecular Graphics and Modelling*, 15:359–363, 1997.
- M. Jambon, A. Imbert, G. Deleage, and C. Geourjon. A New Bioinformatic Approach to Detect Common 3 D Sites in Protein Structures. *Proteins Structure Function and Genetics*, 52(2):137–145, 2003.
- Wolfgang Kabsch. A solution of the best rotation to relate two sets of vectors. *Acta Crystallographica*, 32:922–923, 1976.
- V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
- Michael Neuhaus and Horst Bunke. *Briding the Gap between Graph Edit Distance and Kernel Machines*. World Scientific, New Jersey, 2007.
- J. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533, 2002.

- J.W. Raymond, E.J. Gardiner, and P. Willett. Heuristics for Similarity Searching of Chemical Graphs Using a Maximum Common Edge Subgraph Algorithm. *Journal of Chemical Information and Computer Sciences*, 42(2):305–316, 2002.
- S. Schmitt, M. Hendlich, and G. Klebe. From structure to function: A new approach to detect functional similarity among proteins independent from sequence and fold homology. *Angewandte Chemie International Edition*, 40(17):3141 – 3144, 2001.
- S. Schmitt, D. Kuhn, and G. Klebe. A new method to detect related function among proteins independent of sequence and fold homology. *Journal of Molecular Biology*, 323(2):387–406, 2002.
- N. Weskamp, E. Hüllermeier, D. Kuhn, and G. Klebe. Multiple graph alignment for the structural analysis of protein active sites. *IEEE Transactions on Computational Biology and Bioinformatics*, 4(2):310–320, 2007.