Learning from Interpretations: A Rooted Kernel for Ordered Hypergraphs

Gabriel Wachman GWACHM01@CS.TUFTS.EDU
Roni Khardon RONI@CS.TUFTS.EDU
Department of Computer Science, Tufts University, Medford, MA 02155, USA

Abstract

The paper presents a kernel for learning from ordered hypergraphs, a formalization that captures relational data as used in Inductive Logic Programming (ILP). The kernel generalizes previous approaches to graph kernels in calculating similarity based on walks in the hypergraph. Experiments on challenging chemical datasets demonstrate that the kernel outperforms existing ILP methods, and is competitive with state-of-the-art graph kernels. The experiments also demonstrate that the encoding of graph data can affect performance dramatically, a fact that can be useful beyond kernel methods.

1. Introduction

Recently there is increased interest in learning and mining from graphs, where each example is naturally described using a graph structure (Kramer & De Raedt, 2001; Deshpande et al., 2003; Gärtner et al., 2003; Fröhlich et al., 2005). A prime application of this setting is learning to classify molecules. Here each molecule is a separate example labeled according to some property (e.g. carcinogenicity) and one would like to predict the labels of new examples. The atom-bond structure of the molecule is typically used as the underlying graph structure of the example, and the nodes and edges of the graph are annotated with atom and bond types.

Learning from graphs is a special case of a problem commonly studied in Inductive Logic Programming (ILP) under the name of Learning from Interpretations (De Raedt & Dzeroski, 1994). Here each example is an interpretation from logic programming, which can be seen as a *labeled ordered hypergraph*. For example, the hypergraph H_1 with nodes $\{n_1, n_2, n_3, n_4, n_5\}$, hyperedge (n_1, n_2, n_3) labeled p, and hyperedge (n_1, n_3, n_4) labeled q can be compactly described as $H_1 = \{p(n_1, n_2, n_3), q(n_1, n_4, n_5)\}$. This generalizes the usual notion of a directed graph, in that edges have more than two endpoints and the order of nodes is important. Similarly hypergraphs $H_2 = \{p(n_1, n_2, n_3), p(n_1, n_5, n_6), q(n_4, n_3, n_5)\}$, and $H_3 = \{p(n_1, n_2, n_3), q(n_1, n_3, n_4)\}$ could be different examples in our problem domain. Typically ILP algorithms (Muggleton, 1995; Quinlan, 1990) learn hypotheses represented as sets of first order logic rules and these are used to classify the interpretations. For example, the rule $R = [\exists w, x, y, z, \quad p(w, x, y) q(w, y, z) \rightarrow \mathit{Positive}]$ classifies H_3 as positive and H_1 or H_2 as negative. The search involved in ILP rule learning is complex and the matching problem, that is, checking whether a rule covers an example, is computationally hard. As a result such systems are typically slow and not easy to apply for large datasets.

The use of kernel methods over discrete structures, and in our case ordered hypergraphs, offers an attractive alternative. Recall that a kernel function calculates an inner product over some implicit feature space, and typically one uses a linear threshold function over this space to classify examples (Cristianini & Shawe-Taylor, 2000). A natural goal would be to capture each first-order logic rule as one feature so that the linear threshold function can combine the predictions of different rules. Notice that each rule corresponds to a potential sub-structure of the hypergraph. Therefore features in the implicit space correspond directly to substructures. Indeed, variants of this idea have already been studied for the special case of graphs, and are known as *graph kernels* (Gärtner et al., 2003; Kashima et al., 2003).

In this paper we introduce a new kernel for ordered hypergraphs. To our knowledge this is the first kernel that applies to the general case of learning directly from interpretations, i.e., hypergraphs. The kernel generalizes graph kernels in that its features are based on walks in the hypergraph. Our kernel differs in important ways from general graph kernel construc-

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007. Copyright 2007 by the author(s)/owner(s).

tions and induces a new kernel for data represented as graphs. Comparing to ILP, one can see that the walk-based feature space captures a large set of potential rules. However, as we explain later, not all rules are expressible by walks so there is some trade-off relative to ILP methods.

We evaluate the hypergraph kernel using the Perceptron Algorithm with Margins (Krauth & Mézard, 1987) that has been shown to be competitive with Support Vector Machines. We perform experiments using several challenging chemical datasets. The results demonstrate that the kernel outperforms existing ILP methods, and that it is competitive with state-of-the-art graph kernels. In addition we give insight into two important issues in applying kernel methods to chemical data. The first is the notion of discounting as used in previous work. We present evidence that discounting walks as a function of their length does not lead to a significant difference in performance. The second is that the choice of data encoding is critical in this domain and can lead to a dramatic difference in performance. In particular, the best encoding for these datasets leads to features (corresponding to rules) that are very specific, thus limiting the amount of generalization for any single rule.

To summarize, we contribute a new kernel suitable for the general case of learning from interpretations. Experimental results show that the kernel is effective both in terms of run time and in terms of classification performance. The experiments also highlight the crucial role of data encoding and identify an encoding that seems particularly suited to chemical applications.

2. Definitions and Notation

A *labeled directed graph*, $G = (V, E)$, is a set of nodes V, and a set of edges $E \subseteq V \times V$. Every edge and every vertex are annotated with a label from a fixed set of labels L. *Hypergraphs* are normally defined as a generalization of undirected graphs but here we define them as a generalization of directed graphs as follows. A *labeled ordered hypergraph*, $G = (V, E)$ has a set of vertices V and a set of edges E. Each edge $e \in E$ is a tuple of vertices, $(v_1, \ldots, v_n)$ where $n \geq 1$ is the *arity* of the edge. Every edge is labeled with a label from L. We do not label vertices; instead we can use edges of arity 1. Furthermore, we allow parallel edges, that is, the same tuple can exist in E multiple times, but with different labels. Example of ordered hypergraphs are given in the previous section.

A *walk* in a directed graph is a sequence of vertices and edges $v_1, e_1, v_2, \ldots, e_{n-1} v_n$ such that $e_i = (v_i, v_{i+1}) \in E$. We define a walk in an ordered hypergraph as a sequence of hyperedges where every two consecutive edges have at least one node in common, and no consecutive edges are identical (i.e. we forbid self loops in the graph case). We represent a walk by explicitly specifying indices of the nodes shared by two edges. In particular, we use a string $P = p_1 i_1 j_1 p_2 i_2 j_2 p_3 \ldots i_{n-1} j_{n-1} p_n$ where $p_l \in E$, every i_k represents the exit position of p_k and every j_k represents the entry position of p_{k+1}. For example, $P = p(n_1, n_2, n_3), 1, 1, p(n_1, n_5, n_6), 2, 3, q(n_4, n_3, n_5)$ represents a walk in H_2. Notice that the ordering of edge arguments is important since we track entry and exit positions for the nodes.

A *walk type* is specified by a string $w = r_1 i_1 j_1 r_2 i_2 j_2 r_3 \ldots i_{n-1} j_{n-1} r_n$ where r_l is an edge label. For example, the walk type of P given above is $w = p, 1, 1, p, 2, 3, q$. Thus a walk identifies individual edges, whereas a walk type generalizes the walk and only includes edge labels. Although every walk in a hypergraph is unique, walk types are not; two walks are of the same type iff the strings representing them are identical.

In the following we define a kernel whose features correspond to walk types. Notice that walk types are less expressive than rules in that they make fewer distinctions. For example, walk type $w = p, 1, 1, q$ captures hypergraphs H_1 and H_3 from the introduction, $w = p, 3, 2, q$ captures H_2 and H_3, but there is no walk type equivalent to the rule R from the introduction which captures H_3 but neither of H_1, H_2.

We need the following additional notation. For edge p_i in hypergraph G, $\text{rel}(p_i)$ denotes its label, and p_i^j denotes the vertex at position j in the edge. The string $x.y$ denotes the string resulting from concatenating string y to x. Finally, edge p_i in hypergraph G and walk type w we define $\#(G, p_i, w)$ to be the number of walks of type w starting at edge p_i in G. Note that if $\#(G, p_i, w) > 0$ then w begins with $\text{rel}(p_i)$.

3. A Hypergraph Kernel

We first define a kernel operating on graphs which are "rooted" at particular edges. We then define a kernel operating on graphs in general, and finally discuss variants and extensions of these kernels.

3.1. A Kernel Rooted at Specific Edges

The following kernel $K_n()$ operates on pairs of hypergraphs and edges so it should be written as $K_n((G_1, p_1), (G_2, p_2))$ but to simplify the presentation we omit G_1 and G_2 from the equations. We also omit the fact that p_1 and p'_1 are always in G_1 and p_2 and p'_2 are always in G_2.

Definition 1 *The kernel $K_n()$ is defined recursively as follows:*

$$\begin{aligned}
K_1(p_1,p_2) &= 1 \quad \textit{iff } rel(p_1) = rel(p_2) \\
K_1(p_1,p_2) &= 0 \quad \textit{otherwise} \\
K_n(p_1,p_2) &= K_1(p_1,p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\text{max}\\ \text{arity}}} \sum_{p'_1 : p_1^i = p_1'^j} \\
&\qquad \sum_{p'_2 : p_2^i = p_2'^j} K_{n-1}(p'_1, p'_2).
\end{aligned}$$

where in the sum above $p'_1 \neq p_1$ and $p'_2 \neq p_2$, k is the arity of p_1 and max arity *refers to the maximum arity of any edge in G_1 or G_2. The expression $p'_1 : p_1^i = p_1'^j$ means "an edge p'_1 such that the ith vertex of p_1 is the same as the jth vertex of p'_1."*

The definition immediately gives a dynamic programming algorithm to calculate the kernel by incrementally calculating $K_\ell()$ for $\ell = 2$ to any desired n. It may seem that we need $(\text{max arity})^2|E|^2$ steps to calculate each individual kernel value. One can do much better, however, for sparse hypergraphs (where the number of neighbors is small) by appropriately encoding the neighbors of each node. We next prove that $K_n()$ is indeed a kernel by showing explicitly that it is an inner product for a feature space indexed by all walk types, and where the feature indexed by w takes value $\#(G,p,w)$.

Theorem 2

$$K_n(p_1,p_2) = \sum_{\substack{\text{walk type w}\\ \text{of length n}}} \#(G_1,p_1,w) \cdot \#(G_2,p_2,w).$$

Proof: By induction on n. Base case for $n = 1$. Note that walk types of length 1 are simply edge labels. Hence, we need to show that

$$K_1(p_1,p_2) = \sum_{\substack{\text{edge}\\ \text{label } w}} \#(G_1,p_1,w) \cdot \#(G_2,p_2,w).$$

The sum is zero unless p_1 and p_2 have the same edge label and w is that label, in which case the sum is 1. Assume the claim is true for $n = \ell - 1$. Then

$$K_\ell(p_1,p_2) = K_1(p_1,p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\text{max}\\ \text{arity}}} \sum_{p'_1 : p_1^i = p_1'^j} \sum_{p'_2 : p_2^i = p_2'^j} K_{\ell-1}(p'_1,p'_2)$$

and by the induction hypothesis this is equal to

$$K_1(p_1,p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\text{max}\\ \text{arity}}} \sum_{p'_1 : p_1^i = p_1'^j} \sum_{p'_2 : p_2^i = p_2'^j} \sum_{\substack{\text{w of}\\ \text{length } \ell-1}} \#(G_1,p'_1,w) \cdot \#(G_2,p'_2,w).$$

By re-ordering the summations we get

$$K_1(p_1,p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\text{max}\\ \text{arity}}} \sum_{\substack{\text{w of}\\ \text{length } \ell-1}} \left(\sum_{p'_1 : p_1^i = p_1'^j} \#(G_1,p'_1,w) \right) \left(\sum_{p'_2 : p_2^i = p_2'^j} \#(G_2,p'_2,w) \right)$$

and by the definition of $\#(.,.,.)$ we get

$$K_1(p_1,p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\text{max}\\ \text{arity}}} \sum_{\substack{w \text{ of}\\ \text{length } \ell-1}} \#(G_1,p_1,\text{rel}(p_1).i.j.w) \cdot \#(G_2,p_2,\text{rel}(p_2).i.j.w).$$

Consider a string w' representing a walk type of length $\ell - 1$. By adding $\text{rel}(p_1).i_1.j_1$ to the string we create a new walk type w of length ℓ. Now if we consider an arbitrary walk type w of length ℓ, if w does not start with $\text{rel}(p_1)$ then $\#(G_1,p_1,w)$ is 0. We can therefore replace the equation above with

$$K_1(p_1,p_2) \sum_{\substack{\text{w of}\\ \text{length } \ell}} \#(G_1,p_1,w) \cdot \#(G_2,p_2,w).$$

Finally, note that if p_1, p_2 do not have the same edge label then for every w at least one of $\#(G_1,p_1,w)$, $\#(G_2,p_2,w)$ is 0 and therefore the sum is 0 so we can omit K_1 from the expression. Similarly, if p_1, p_2 do have the same relation symbol $K_1 = 1$ and we can omit K_1. Hence, as required, we have

$$\sum_{\substack{w \text{ of}\\ \text{length } \ell}} \#(G_1,p_1,w) \cdot \#(G_2,p_2,w). \qquad \blacksquare$$

3.2. A General Kernel for Hypergraphs

We next define another kernel $K'_n()$ that operates globally on the graphs:

Definition 3

$$K'_n(G_1,G_2) = \sum_{p_1 \in E_1} \sum_{p_2 \in E_2} K_n(p_1,p_2) \qquad (1)$$

One can show that (1) is a kernel by re-writing it as

$$\sum_{\substack{w \text{ of} \\ \text{length } \ell}} \left(\sum_{p_1 \in G_1} \#(G_1, p_1, w) \right) \left(\sum_{p_2 \in G_2} \#(G_2, p_2, w) \right)$$

The first inner sum is the total number of walks of type w in G_1 and likewise the second inner sum for G_2. In this representation it is easy to see that every element of the outer sum is the total number of walks of type w in G_1 times the total number of walks of type w in G_2.

3.3. Other Kernel Variants

A general idea in string and graph kernels, where we consider an infinite number of features, is to discount the contribution of longer walks. Indeed, this discount factor is necessary in order to achieve convergence when summing contributions of all possible walks on length 1 to ∞ (Gärtner et al., 2003). This is easily implemented using our kernel as follows.

Definition 4 (Discounted Kernel)

$$K_n^D(G_1, G_2) = \sum_{i=1}^{n} \gamma^i K_i'(G_1, G_2) \qquad (2)$$

$$K_n'^D(G_1, G_2) = \frac{K_n^D(G_1, G_2)}{\sqrt{K_n^D(G_1, G_1) K_n^D(G_2, G_2)}} \qquad (3)$$

It follows from standard properties that (2) and (3) are kernels (Cristianini & Shawe-Taylor, 2000). Notice that with $\gamma < 1$ we get discounting. However, for our kernel γ is not restricted in this way. In fact, we can emphasize the contribution of longer walks by using $\gamma > 1$. This is intuitively appealing since longer walks give more informative matches between the graphs.

Finally, another idea common in string kernels is to count a match even for strings that have a few mismatches. The same idea, allowing a constant number of mismatches, can be adapted to our setting by adding another index to the kernel that counts the number of edge labels on the walk that do not match and incorporating this into the dynamic programming calculation.

4. Discussion and Related Work

An important basic result for graph kernels shows that it is computationally hard to calculate a kernel whose feature space corresponds to all possible subgraphs unique up to isomorphism, where each feature is binary-valued according to the existence of that particular subgraph (Gärtner et al., 2003). Therefore, one must compromise and use a less expressive family of subgraphs as features. On the positive side, recent work on graph kernels uses various properties to create a similarity measure between two graphs: the number of labeled walks shared between graphs (Gärtner et al., 2003); the probability of a random walk in both graphs (Kashima et al., 2003); the number of a certain type of pre-defined sub-structures present in both graphs (Kramer & De Raedt, 2001; Deshpande et al., 2003; Horváth et al., 2004; Ralaivola et al., 2005; Tsuda & Kudo, 2006).

Our work is most closely related to the walk-based kernels (Gärtner et al., 2003; Kashima et al., 2003). The kernel of Gärtner et al. (2003) computes the number of walks of any length (with identical label sequences) that the two input graphs share. Kashima et al. (2003) present a *marginalized graph kernel* that computes the similarity of two graphs based on the probability that a random walk occurs in both graphs. Both kernels use walks of arbitrary length and sum their contributions so both have some form of discounting to guarantee that the kernel value is not infinite. Both kernels are also expensive to compute; the kernel by Gärtner et al. (2003) must invert or diagonalize a matrix that is quadratic in the number of vertices of the direct product graph, and that in Kashima et al. (2003) must solve a system of linear equations described by a matrix quadratic in the number of vertices in the direct product graph. See (Vishwanathan et al., 2006) for recent speedup of these kernels.

Although our kernel is similarly based on walks there are several important differences. First, we focus on a fixed finite length of walks. This helps avoid unintuitive discounting of the weights of long walks. We are not aware of a method for capturing kernels based on arbitrary length walks with hypergraphs. Second, we provide a dynamic programming algorithm to calculate the kernel, that can be more efficient than the algorithms mentioned above. Third, there are differences in the feature space that make our kernel more compact than other walk-based kernels. This is illustrated by the following example. Consider a pattern capturing a star graph with one center v_0 and 4 outer nodes where each edges has a different label, i.e. $\{l_1(v_0, v_1), l_2(v_0, v_2), l_3(v_0, v_3), l_4(v_0, v_4)\}$. To capture this pattern with a walk one must consider an edge in each direction and go back and forth on each edge (except the first and last) so we need a walk of length 6. In our case, since we match positions but do not consider the directionality of the edge this can be captured by a walk of length 4 of type $l_1, 1, 1, l_2, 1, 1, l_3, 1, 1, l_4$ (so we enter and exit the edge in the same node). Thus our kernel can be more expressive, capturing complex

Dataset	Examples	# Atoms	Majority Class
NCTRER	232	7-44	0.60
MUTAG	188	15-41	0.67
PTC(MM)	336	2-106	0.62
PTC(MR)	344	2-106	0.56
PTC(FM)	349	2-106	0.59
PTC(FR)	351	2-106	0.66
NCI-HIV	41606	2-438	0.99

Figure 1. Datasets Used in Experiments

sub-graphs using shorter walks.

It is also important to compare our features to rules used in ILP. The example in Section 2 illustrates that we do not account for multiple shared nodes between adjacent edges on a walk, therefore a walk with multiple shared nodes will be represented in the features of more than one walk type. In addition our walks are only linking adjacent edges so they cannot make connections between edges several hops away from each other and this is again weaker than the rules in ILP. Designing kernels that do capture such complexity is an important open problem.

Finally, due to the above restriction (linking one node at a time), one can translate the hypergraph into a (quadratic size) directed graph by adding a new node for each hyperedge and connecting the new node to the nodes belonging to the hyperedge. Now a walk in the new graph corresponds to a walk type in the hypergraph. Hence we may be able to simulate the hypergraph kernel through the graph though perhaps at increased complexity since the graph is larger. This may be interesting in terms of applying other graph kernels to hypergraphs.

5. Experiments and Results

5.1. Datasets

We demonstrate the performance of the hypergraph kernel on chemical datasets each of which contains chemical descriptions along with a label based on the chemical's activity. The number of examples, number of hypergraph nodes in examples, and label distribution in these datasets are given in Figure 1. The National Center for Toxicological Research Estrogen Receptor Binding (NCTRER) dataset (Fang et al., 2001; Blair et al., 2000; Branham et al., 2002) contains chemicals that are labeled based on how well they bind to estrogen receptors. In our experiments we consider molecules labeled "active strong," "active medium," and "active weak" to be positive, and molecules labeled "slight binder," and "inactive" to be negative.[1] The Mutagenesis (MUTAG) dataset (Srinivasan et al., 1996) contains chemicals that are labeled based on their mutagenicity; we used the 188 example "regression friendly" portion of the data. The Predictive Toxicology Challenge (PTC) dataset[2] contains 417 chemical descriptions labeled according to their carcinogenicity to rodents. Each chemical is evaluated based on whether it was carcinogenic to female rats, female mice, male rats and male mice. Following Kashima et al. (2003) and others, we treat any molecule labeled "CE," "SE," or "P" as positive, "NE" and "N" as negative, and ignore examples labeled "EE," "IS," and "E" as these labels indicate an unsure classification. The National Cancer Institute's AIDS Anti-viral Screen Program (NCI-HIV) dataset[3] contains chemical descriptions labeled based on the ability of the chemical to inhibit HIV in a specific experimental context. Each chemical is labeled "confirmed active" (CA), "confirmed inactive" (CI), and "moderately active" (CM). In our experiments we ignore the moderately active chemicals since their label is less reliable (these are compounds that yielded different results in multiple measurements). Notice that this is a large dataset and its class distribution is very skewed.

5.2. Dataset Encoding

Each molecule in the datasets is represented as a set of predicates. As in previous studies (e.g. (Deshpande et al., 2003; Horváth et al., 2004)), we eliminate hydrogen atoms since this reduces the size of examples and hydrogens are implicit in the reduced representation. We explore three methods of encoding the datasets. In the first encoding, bonds and their types are represented by edges, e.g., $bondtype1(x, y)$. Atom type is encoded using unary edges with the type as the relation name. Since bonds are not directed, we store bond relations twice in this encoding, once with each vertex ordering, as there is not enough information in the edge labels to imply the order; for example, using edges $(v1, v2)$ and $(v2, v3)$, both labeled bondtype1, we find the walk type "bondtype1,2,1,bondtype1" to be present. Two edges from another graph, $(w2, w1)$ and $(w2, w3)$ should generate the same walk type, but they will not unless we duplicate edges. In the second encoding we eliminate the original "bondtype" labels and create $bondXtoY$ predicates, where X is the type of the first atom in the bond, and Y is the type of the second. To get a compact representation, we

[1]We used the version of the dataset entitled `NCTRER_v3b_232_10Apr2006.sdf`.

[2]`http://www.predictive-toxicology.org/ptc/`

[3]`http://dtp.nci.nih.gov/docs/aids/aids_data.html`

impose an ordering on the $\mathit{bondXtoY}$ relation such that X is lexicographically smaller than Y. This ensures that $\mathit{bondXtoY}$ and $\mathit{bondYtoX}$ will not appear in the dataset together, avoiding the need to duplicate links (except in the case of $\mathit{bondXtoX}$ where we do duplicate). The final encoding follows the work of Gärtner (2005) and Mahé et al. (2004) and encodes even more information about endpoints of bonds. In particular each argument of the bond predicates encodes the types of its atom and the types of all its neighbors. This technique is also similar to the *neighborhood kernel* discussed by Fröhlich et al. (2005), however we do not use as detailed information, and we use the immediate neighborhood only. As in the second encoding we lexicographically order the arguments and duplicate the bond only if the arguments are identical.

The following example illustrates the three encodings. Consider a star graph similar to the one given above with labels as follows $\{\mathit{bond}(v_0, v_1)$, $\mathit{bond}(v_0, v_2)$, $\mathit{bond}(v_0, v_3)$, $\mathit{bond}(v_0, v_4)$, $A(v_0)$, $B(v_1)$, $B(v_2)$, $C(v_3)$, $D(v_4)\}$ (in the chemical domain A, B, C, D would be element names). Under encoding 1, the bond structure of the graph would be encoded as

$$\mathit{bond}(v_1, v_0), \mathit{bond}(v_0, v_1), \mathit{bond}(v_2, v_0), \mathit{bond}(v_0, v_2),$$
$$\mathit{bond}(v_3, v_0), \mathit{bond}(v_0, v_3), \mathit{bond}(v_4, v_0), \mathit{bond}(v_0, v_4)$$

and we add the node types to this structure. Under encoding 2 we get:

$$\mathit{bondAtoB}(v_0, v_1), \mathit{bondAtoB}(v_0, v_2),$$
$$\mathit{bondAtoC}(v_0, v_3), \mathit{bondAtoD}(v_0, v_4).$$

In encoding 3 for each argument we represent the atom type first, followed by the node types of its neighbors. Below we separate the node from its neighbors with a vertical bar. This yields

$$\mathit{bond}\underline{A|BBCD}\mathit{to}\underline{B|A}(v_0, v_1),$$
$$\mathit{bond}\underline{A|BBCD}\mathit{to}\underline{B|A}(v_0, v_2),$$
$$\mathit{bond}\underline{A|BBCD}\mathit{to}\underline{C|A}(v_0, v_3),$$
$$\mathit{bond}\underline{A|BBCD}\mathit{to}\underline{D|A}(v_0, v_4).$$

Encoding 3 will result in fewer walk matches between graphs, since a match requires both that the vertices are of the same type, and that all of their neighbors are of the same type. As a result the transfer between graphs is smaller and less generalization is possible from one feature. Another important property of encoding 3 is that it makes for faster computation since fewer matches mean fewer items added in the dynamic programming formula.

Length	Encoding 1	Encoding 2	Encoding 3
1	0.64 ± 0.08	0.67 ± 0.08	0.79 ± 0.08
2	0.64 ± 0.10	0.68 ± 0.10	0.83 ± 0.05
3	0.65 ± 0.10	0.67 ± 0.10	0.84 ± 0.07
4	0.66 ± 0.06	0.62 ± 0.06	0.85 ± 0.09
5	0.66 ± 0.10	0.64 ± 0.07	0.85 ± 0.06
6	0.64 ± 0.10	0.62 ± 0.10	0.83 ± 0.07
7	0.66 ± 0.08	0.62 ± 0.08	0.80 ± 0.08
8	0.66 ± 0.10	0.59 ± 0.10	0.75 ± 0.07
16	0.67 ± 0.09	0.66 ± 0.12	0.68 ± 0.11
nFOIL	0.78 ± 0.09		

Figure 2. Accuracy on the NCTRER dataset varying walk length and encoding.

5.3. Experiments and Results

In all of the experiments, we used the Perceptron with Margins (Krauth & Mézard, 1987) as the learning algorithm. Unlike the standard Perceptron (Rosenblatt, 1958), the Perceptron with Margins updates when an example falls within a certain distance (margin) of the current hyperplane. This algorithm has been shown to perform similar to SVM often reducing run time (Li et al., 2002). We did not optimize the margin parameter; instead as suggested in (Khardon & Wachman, 2007) we chose a small relative margin setting of 0.1. We ran all experiments with 10-fold cross validation. On Mutagenesis, NCTRER, and PTC we trained for 20 iterations, and on NCI-HIV we trained for 2 iterations. In all our experiments we used the kernel from Equation (3) with $\gamma = 1$ (i.e. no discounting) except when explicitly stated otherwise.

In the first set of experiments we examined the role of the data encoding by using our kernel on the NCTRER dataset. Results are given in Figure 2. Notice that for encoding 1 the kernel does not perform well but with encoding 3 the results are significantly improved. Encoding 3 illustrates the importance of a more varied substructure than walks that may be needed. Furthermore, the results suggest that very long walks do not perform well. For comparison, we give the best result reported by Landwehr et al. (2006), who compare state-of-the-art ILP solvers. The kernel method does better when combined with encoding 3. It would be interesting to test whether ILP methods can benefit from similar encodings.

In the second set of experiments, we explored the effect of discounted and incremented walks on the NCTRER dataset using γ values of $\{0.1, 0.5, 0.9, 1, 2, 10\}$ and walk lengths of $\{2, 3, 4, 5, 6, 16\}$. The results are given in Figure 3. While some performance variation is noticeable it is statistically insignificant. Thus although incrementing long paths is intuitively attractive, our experiments show that the effect of discount-

Length	$\gamma = 0.1$	0.5	0.8	1	2	10
2	0.82	0.81	0.82	0.83	0.81	0.82
3	0.82	0.84	0.84	0.84	0.84	0.83
4	0.82	0.85	0.84	0.85	0.84	0.85
5	0.83	0.84	0.84	0.85	0.85	0.85
6	0.82	0.84	0.82	0.83	0.82	0.81
16	0.84	0.69	0.68	0.68	0.67	0.68

Figure 3. Accuracy on NCTRER varying walk length and discount factor γ.

L	AB	AB+H	AB+LC	AB+H+LC
1	$0.72 \pm .11$	$0.83 \pm .08$	$0.87 \pm .06$	$0.88 \pm .06$
2	$0.69 \pm .13$	$0.85 \pm .09$	$0.88 \pm .08$	$0.85 \pm .13$
3	$0.77 \pm .09$	$0.86 \pm .10$	$0.89 \pm .07$	$0.87 \pm .08$
4	$0.77 \pm .09$	$0.85 \pm .10$	$0.88 \pm .08$	$0.88 \pm .11$
5	$0.79 \pm .08$	$0.84 \pm .09$	$0.87 \pm .09$	$0.86 \pm .12$
10	$0.85 \pm .11$	$0.81 \pm .10$	$0.84 \pm .10$	$0.83 \pm .11$
1	$0.85 \pm .09$	$0.88 \pm .10$	$0.89 \pm .10$	$0.91 \pm .08$
2	$0.84 \pm .10$	$0.86 \pm .10$	$0.91 \pm .08$	$0.89 \pm .08$
3	$0.83 \pm .08$	$0.85 \pm .12$	$0.89 \pm .09$	$0.90 \pm .13$
4	$0.85 \pm .10$	$0.85 \pm .12$	$0.91 \pm .06$	$0.87 \pm .12$
5	$0.85 \pm .09$	$0.85 \pm .12$	$0.90 \pm .06$	$0.85 \pm .13$
10	$0.86 \pm .08$	$0.78 \pm .12$	$0.84 \pm .09$	$0.77 \pm .12$

Figure 4. Accuracy on Mutagenesis Dataset. L denotes walk length. Top: encoding 1. Bottom: encoding 3.

Dataset	L	HG	OA	MG
PTC(FM)	5	$0.64 \pm .10$	$0.64 \pm .03$	$0.62 \pm .03$
PTC(FR)	16	$0.67 \pm .07$	$0.67 \pm .02$	$0.67 \pm .02$
PTC(MM)	5	$0.64 \pm .07$	$0.68 \pm .02$	$0.67 \pm .02$
PTC(MR)	16	$0.60 \pm .07$	$0.63 \pm .02$	$0.58 \pm .01$
Dataset		HG	CPK	GK
NCI-HIV	5	0.94 ± 0.02	0.91 ± 0.01	$0.94 \pm .01$

Figure 5. Accuracy on PTC and area under ROC curve for NCI-HIV. "HG" is the hypergraph kernel.

ing is minimal, and it can be avoided.

Through the third set of experiments we illustrate the benefit of being able to process high-arity edges. We ran over the Mutagenesis dataset using encoding 1 and 3, and with various combinations of binary edges (atom-bond information), hyperedges (ring structures, etc.), and discretized charge, lumo, and logp features encoded as unary edges. Only binary edges are affected by the encoding; we did not change hyperedges or unary edges. Note that lumo and logp are global properties of the molecule so they translate to isolated nodes in the graph. Note also that our kernel gives flexibility to use the hyperedges and multiple unary predicates for the same node, that is, multiple labels. We report the results in Figure 4; "AB" is atom-bond information, "H" is hyperedge information, and "LC" is lumo, logp and charge information. Our results are competitive with the best reported graph kernel for this dataset (Kashima et al., 2003). When using encoding 1, it is clear that adding the hyperedges to the dataset gives a substantial gain in performance. Note that the best length walk is shorter when using the hyperedges. This may be due to the fact that larger substructure may be captured in fewer edges. It may also explain the fact that one can do without the ring structures, since longer walks may be able to capture them. When using encoding 3, the benefit of using hyperedges is less pronounced; this is likely due to the fact that because binary edges under encoding 3 contain information about their neighbors, they perform a similar function to that captured by rings as in our hyperedges. Thus hyperedges (and multiple labels) can be useful and they are easily incorporated by our kernel. If hyperedges capture information that is not derived from the graph structure one might expect to attain significant improvements in performance.

In the final set of experiments we compared our kernel to other work with graph kernels on the challenging datasets PTC and NCI-HIV. Given the conclusions from previous experiments we used encoding 3 and no discount ($\gamma = 1$). On the PTC dataset, due to its small size we were able to explore a short (2), medium (5), and long (16) walk length and we report the best result. The NCI-HIV dataset is relatively large and it includes large molecules. To reduce learning time without altering dataset statistics, in each fold we removed molecules with more than 200 atoms from the training set but kept the test set unmodified. Overall this means we removed 4 positive and 79 negative molecules from the training data. The results for walk lengths 2-5 are very similar and we report only the result for length 5. Results are given in Figure 5. On the PTC dataset, the results we give are competitive with the best performance recorded in Fröhlich et al. (2005), which is attained using their optimal assignment kernel (OA) and the marginalized graph kernel (Kashima et al., 2003) (MG). On the HIV dataset, in order to compare with previous work we report the area under the ROC curve (although precision and recall may be more appropriate due to the skew in labels). Our results outperform the frequent substructure propositionalization approach (Deshpande et al., 2003) and are competitive with the Cyclic Pattern Kernel (CPK) (Horváth et al., 2004) and the approximation of the infinite walk graph kernel (GK) reported by Gärtner (2005).

On the NCTRER, PTC, and Mutagenesis datasets, a typical run time for 10-fold cross validation was under a minute, and often less than 20 seconds on a dual 2.8 GHz Intel Xeon machine with at most one other job scheduled on it. On the NCI-HIV dataset, the runtime varied significantly by the walk length: for a length 3 walk, the average run time *per fold* was about 11 hours, while for a length 5 walk the average run time per fold was about 18 hours.

To summarize, the experiments demonstrate that our kernel can outperform ILP methods, that high arity predicates are easily incorporated as hyperedges and that this can be useful, and that the kernel is competitive with graph kernels when used on graph data.

6. Conclusion

The paper introduced a kernel for learning from ordered hypergraphs that is suitable for learning from ILP data. The new kernel generalizes previous work on graph kernels but adds properties that make it interesting even in the graph setting. The experimental results demonstrate that the new kernel leads to good performance on chemical datasets when used with a data encoding leading to specific features. This raises the question whether similar encodings can help boost performance of rule based systems as well on such datasets. Another major open question is whether one can develop kernels whose feature spaces capture more expressive sets of rules where variables are shared among multiple predicates. Finally, our experimental results show competitive performance with Fröhlich et al. (2005) although we do not use any of the expert knowledge used there or the improved optimal assignment kernels. It is interesting to investigate whether these features are equally useful with our kernel.

References

Blair, R., Fang, H., Branham, W., Hass, B., Dial, S., Moland, C., Tong, W., Shi, L., Perkins, R., & Sheehan, D. (2000). The estrogen receptor relative binding affinities of 188 natural and xenochemicals: Structural diversity of ligands. *Toxicol. Sci.*, *54*, 138–153.

Branham, W., Dial, S., Moland, C., Hass, B., Blair, R., Fang, H., Shi, L., Tong, W., Perkins, R., & Sheehan, D. (2002). Binding of phytoestrogens and mycoestrogens to the rat uterine estrogen receptor. *J. Nutr.*, *132*, 658–664.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines*. Cambridge University Press.

De Raedt, L., & Dzeroski, S. (1994). First order *jk*-clausal theories are PAC-learnable. *Artificial Intelligence*, *70*, 375–392.

Deshpande, M., Kuramochi, M., & Karypis, G. (2003). Frequent sub-structure-based approaches for classifying chemical compounds. *ICDM* (pp. 35–42).

Fang, H., Tong, W., Shi, L., Blair, R., Perkins, R., Branham, W., Hass, B., Xie, Q., Dial, S., Moland, C., & Sheehan, D. (2001). Structure-activity relationships for a large diverse set of natural, synthetic, and environmental estrogens. *Chem. Res. Tox.*, *14*, 280–294.

Fröhlich, H., Wegner, J. K., Sieker, F., & Zell, A. (2005). Optimal assignment kernels for attributed molecular graphs. *ICML* (pp. 225–232).

Gärtner, T. (2005). Predictive graph mining with kernel methods. In *Advanced methods for knowledge discovery from complex data*. Springer.

Gärtner, T., Flach, P. A., & Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. *COLT* (pp. 129–143).

Horváth, T., Gärtner, T., & Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. *KDD* (pp. 158–167).

Kashima, H., Tsuda, K., & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. *ICML* (pp. 321–328).

Khardon, R., & Wachman, G. (2007). Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, *8*, 227–248.

Kramer, S., & De Raedt, L. (2001). Feature construction with version spaces for biochemical applications. *ICML* (pp. 258–265).

Krauth, W., & Mézard, M. (1987). Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, *20*, 745–752.

Landwehr, N., Passerini, A., De Raedt, L., & Frasconi, P. (2006). kfoil: Learning simple relational kernels. *AAAI*.

Li, Y., Zaragoza, H., Herbrich, R., Shawe-Taylor, J., & Kandola, J. (2002). The perceptron algorithm with uneven margins. *International Conference on Machine Learning* (pp. 379–386).

Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., & Vert, J.-P. (2004). Extensions of marginalized graph kernels. *ICML*.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*, 245–286.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.

Ralaivola, L., Swamidass, S. J., Saigo, H., & Baldi, P. (2005). Graph kernels for chemical informatics. *Neural Networks*, *18*, 1093–1110.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386–407.

Srinivasan, A., Muggleton, S., Sternberg, M., & King, R. (1996). Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, *85*, 277–299.

Tsuda, K., & Kudo, T. (2006). Clustering graphs by weighted substructure mining. *ICML* (pp. 953–960).

Vishwanathan, S. V. N., Borgwardt, K. M., & Schraudolph, N. (2006). Fast computation of graph kernels. *NIPS 19*.