Technical Report No. TR-150

Towards the Inference of Graphs on Ordered Vertices

Alexander Zien[1,2], Gunnar Rätsch[2], Cheng Soon Ong[1,2]

Aug 7, 2006

[1] Max Planck Inst. for Biol. Cybernetics, Spemannstr. 38, Tübingen, Germany
[2] Friedrich Miescher Lab, Max Planck Soc., Spemannstr. 39, Tübingen, Germany

Towards the Inference of Graphs on Ordered Vertices

Alexander Zien, Gunnar Rätsch, Cheng Soon Ong

Abstract. We propose novel methods for machine learning of structured output spaces. Specifically, we consider outputs which are graphs with vertices that have a natural order. We consider the usual adjacency matrix representation of graphs, as well as two other representations for such a graph: (a) decomposing the graph into a set of paths, (b) converting the graph into a single sequence of nodes with labeled edges. For each of the three representations, we propose an encoding and decoding scheme. We also propose an evaluation measure for comparing two graphs.

1 Introduction

Structured output learning [1, 2, 3] is a learning paradigm of increasing interest and popularity, because it extends classification from using atomic class labels to structured objects like sequences ("label sequence learning" [2, 4]). As in binary classification a mapping of inputs x to outputs y is sought for, but in structured output learning y is an element of a structured space, eg of sequences or graphs. The common idea is to map the pairs of inputs x and outputs y to a joint feature representation $\Phi(x, y)$ and then to find a linear function $\mathbf{w}^\top \Phi(\mathbf{x}, y)$ such that

$$x \quad \mapsto \quad \arg\max_y \mathbf{w}^\top \Phi(\mathbf{x}, y) \tag{1}$$

represents the desired map. Solving (1) for a given $\mathbf{w}$ is also called decoding. Note that with appropriate choice of $\Phi(\cdot)$ the model may be kernelized; however, for the sake of easier accessability we will not explixitly carry this through.

Learning with a training set of known pairs (x_i, y_i) is usually performed by solving a mathematical program like

$$\min_{\mathbf{w},\xi} \qquad ||\mathbf{w}||^2 + \sum_i \xi_i \tag{2}$$

$$s.t. \qquad \forall i: \ \forall \bar{y}: \neq y_i \xi_i \geq \ell\left(f(x_i, y_i) - f(x_i, \bar{y})\right) \quad , \tag{3}$$

where a frequent choice for the loss function $\ell(\cdot)$ is the hinge loss given by $\ell(t) = \max(1 - t, 0)$. Since the number of constraints is usually too large (eg, exponential in the size of the x_i), column generation methods are used to (approximately) find the optimimum $\mathbf{w}$: the prediction problem (1) is repeatedly solved to find structures $\bar{y}$ yielding most active constraints.

We here present an effort to push the limits of structured output learning within this framework beyond sequences: we move on to a certain class of graphs characterized by a pre-known ordering of the nodes. Precisely, x is a variable-sized set of nodes together with their ordering, and y is a graph on these nodes such that all edges point forward. We enable to learn maps $x \mapsto y$ for these data types by developing three decoding algorithms.

Beyond being an exiting advance in itself, this extension draws relevance from potential applications. For instance, it applies to the prediction of tertiary (3D) protein structure on the basis of contact maps. A contact map is an undirected graph that connects spatially adjacent amino acids. The amino acids have a natural ordering given by the backbone of the protein.

A second possible application, and our true motiviation for this work, is the prediction of splice graphs of genes in genomic DNA sequences. A splice graph summarizes all possibilities of dissecting the gene into exons (which encode proteins) exons and introns (which are disposed of). Existing state of the art gene finders, eg GenScan and mSplicer, predict a single splice form, ie they solve a label sequence learning problem. However, many genes can be expressed in several variants, for example by means of alternative splicing. It is of high interest to biologists to discover these variants. Each alternative is described by a single path from a given start node to a given end node;

a graph may be used to summarize the paths. We will take the importance of paths into account in the specification of the formal problem below.

As mentioned above, we will need to define features on graphs (that may take the information in the inputs into account, ie node sets, node labels, node ordering, etc). The recent years have seen some research on such features in the context of kernel methods. While subgraphs are traditionally used in chemoinformatics, only recently mining for frequent substructures has become practical [5, and references therein]. Further, graph kernels have been defined that use paths resulting from random walks as features [6, 7]. Finally, it has recently been proposed to use spectral features deduced from graph Laplacians [8].

The structure prediction problem (1) is closely related to the problem of pre-image construction, which consists of solving $\arg\min_y \|\Psi(y) - \mathbf{w}\|^2$ for a given $\mathbf{w}$. Note that this is equal to $\arg\max_y \mathbf{w}^\top \Psi(y) - 1/2\|\Psi(y)\|^2$, which can be seen as a regularized decoding with the feature map $\Psi(y) := \Phi(x, y)$. While there exists work on finding graph pre-images [9], it consideres general graphs and thus has to cope with a very hard combinatorial optimization problem.

This paper is organized as follows. In the next section, we specify the problem setting in more detail. The sections 3 and 4 present corresponding graph prediction/decoding methods: two that are based on sequential features, and one based on subgraph features, respectively. We then briefly touch on possibilities to evaluate predictions and finally discuss advantages and disadvantages of the three proposed methods.

2 Problem Setting

We consider the problem of learning a graph on a set V of nodes with known ordering. This problem has a value in itself, e.g. for the applications mentioned above. From a wider perspective it can also be seen as an intermediate step – to our knowledge, the first one – from label sequence learning [4] towards learning general graphs. In the now popular framework for structured output learning, the idea is to define a joint kernel over inputs and outputs [1]. Given such a kernel, the remaining required ingredient is a decoding algorithm that produces high scoring structures based on a given parameter setting. In this work, we present several strategies for decoding algorithms for graphs on node sets with a pre-known ordering.

Once a decoding method is implemented, parameters can be learned by solving a mathematical program (usually a LP or QP) [3]. Due to the combinatorial nature of the structured output space, the optimization problem usually has a huge number of constraints, and column generation techniques are commonly used. Thereby alternating steps of (i) parameter updates by solving a smaller optimization problem and (ii) incrementing the constraint set according to decoded structures are repeated until convergence.

We now formalize the problem. Let, w.l.o.g., the n nodes be $V = \{1, \ldots, n\}$. The problem becomes both more practically relevant and more challenging by allowing for different numbers n of nodes for both training and testing cases. This includes the possibility to predict a graph on a node set of a size that does not occur in the training set. We restrict ourselves to graphs the edges of which point forward with respect to the ordering of the nodes: $(i, j) \in E \Rightarrow i < j$, where $E \subset V \times V$ is the set of edges. Note that there is a one-to-one correspondence between directed acyclic graphs (DAGs) respecting these ordering requirements and undirected graphs without self-loops. In our setting it is often more convenient to specify the edges by means of an adjacency matrix $A \in \{0, 1\}^{V \times V}$ such that $A_{ij} = 1 \Leftrightarrow (i, j) \in E$. Corresponding to our assumptions about directions, A has to be upper triagonal. When it is convenient, we might equivalently treat A as being symmetric. In either case, the diagonal is zero.

3 Graph Learning with Path Features

We start with an informal description of the idea, which is detailed below. Let Γ be a directed graph on ordered vertexes with known start and end node. We define a path through Γ as a subgraph with the following properties: (a) it connects the start and end node, (b) every node has at most one edge, and (c) every edge is part of the connection between the start and end nodes. We define the width d of the graph as the maximum number of edges between all ordered bi-partitions of nodes (all nodes of one set are smaller than the nodes of the other set). It can easily be seen that graphs of width d can always be *covered* by n paths (see illustration in Figure 1). We call this the *multiple path representation* of the graph. Additionally, we consider a *composite path representation*. Here, a single path on the union of all nodes used in the multiple path representation represents the graph. The edges are labeled by which path was started and ended with the edge (cf. Figure 1).

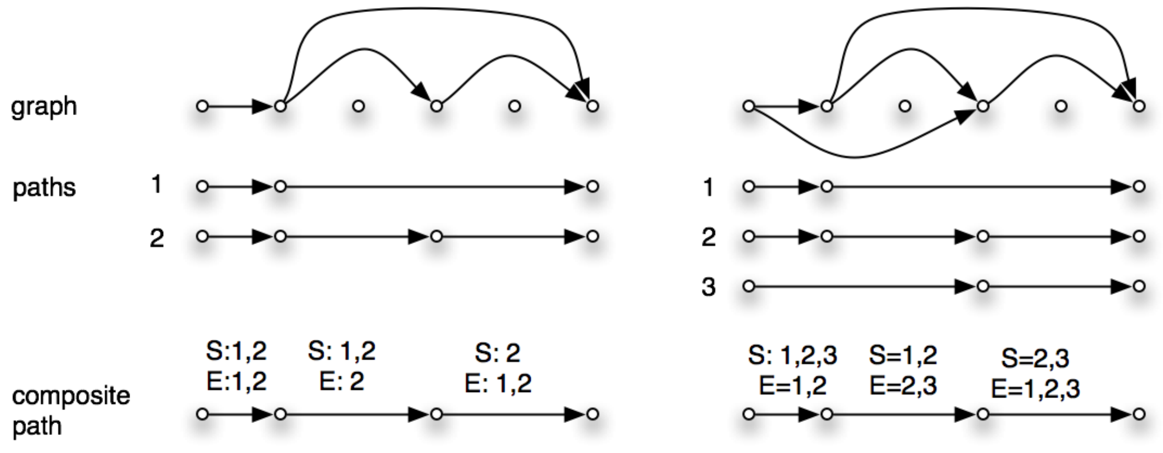

Figure 1: Shown are two graphs on ordered vertexes (left: width $d = 2$; right: width $d = 3$). The graph can be represented by a small number of paths covering the full graph. By introducing edge labels one can represent these paths by one path on the union of the vertexes of the paths.

3.1 Converting between Paths and Graphs

From each graph $\Gamma = (V, E)$, all possible paths in the multiple path representation can be obtained by first arbitrarily ordering the branches at each vertex, and then performing depth first search. Since there is only one terminal node, n, and there are no cycles, this results in a finite number of paths from each branch of the depth first search tree. Observe however, the subtle independence assumption that has been made by this approach. At each vertex, the choice of the subsequent branch is independent of how we arrived at the vertex in the first place. The number of independent paths generated is exponential in the branching factor of the graph (i.e. the maximum number of outgoing edges). However, we can "cover" the graph by using a much smaller number of paths.

Since the vertexes of Γ are ordered, finding the maximum cut that respects the ordering amounts to finding the maximum number of edges when cutting between all pairs of consecutive vertexes,

$$d := \max_{i=1,\ldots,n} \left|\{(j,k) \in E \text{ such that } j \leq i \text{ and } k \geq i+1\}\right|,$$

where $|S|$ denotes the number of elements in the set S. Note that we also consider the edges which start before i and end after $(i + 1)$. The width can be computed by increasing a counter for every outgoing edge of each vertex and decreasing it for every incoming edge. The maximum value attained by this counter is the width of the graph. For a graph Γ, we say that a set of paths $\mathcal{P}$ *covers* Γ if each edge in Γ has a corresponding edge in $\mathcal{P}$. Since the maximum number of edges at any point on a graph is given by the width, the number of paths we need to cover the graph is also given by the width. Observe that there can be more than one possible set of paths which cover a given graph.

Given a set of paths, we can deduce the corresponding graph by taking the union of all the vertexes in the paths. If an edge between vertex v_i and v_j occurs in more than one path, it is merged. In other words, an edge is identified only by the vertexes that it connects. This merging of edges means that the events before the edge are independent of the events after the edge. While a graph can be covered by more than one set of paths, as described in the previous paragraph, each set of paths $\mathcal{P}$ has a unique graph Γ.

Given the multiple path representation it is straightforward to generate the composite path representation by considering a single path through all nodes used in the multiple path representation and adding labels to the edges indicating which path uses the connected nodes. One way is to add to each edge the information which path is started and which is ended (cf. Figure 1).

3.2 Predictions using the Multiple Path Representation

If we would like to predict a set of paths in order to predict the graph, then we can consider learning a function f on paths p. The function should classify paths in the graph as positive and those which are not in the graph as negative. Let $\mathcal{P}^*(\Gamma)$ be the set of *all* paths through the graph Γ, $\mathcal{P}^+(\Gamma) \subseteq \mathcal{P}^*(\Gamma)$ a set of path covering the graph, and $\mathcal{P}^-(\Gamma)$ the set of invalid paths on V (anything not in $\mathcal{P}^*(\Gamma)$). Given a set of N graphs $\Gamma_1, \dots, \Gamma_N$, we first require that all invalid paths score negative (with margin):

$$f(\pi) \leq -1 + \xi_i, \qquad \text{for} \quad \pi \in \mathcal{P}^-(\Gamma_i) \text{ and } i = 1, \dots, N, \tag{4}$$

where the ξ_i's are slack variables implementing a soft-margin. Note that the number of constraints is exponential in the number of nodes of the graph. Moreover, we would like to score true paths positive. Here we have two possibilities: (1) we only force the score of the chosen covering paths to be positive, i.e.

$$f(\pi) \geq 1 - \xi_i, \qquad \text{for} \quad \pi \in \mathcal{P}^+(\Gamma_i) \text{ and } i = 1, \dots, N, \tag{5}$$

or we require that all valid paths should score positive:

$$f(\pi) \geq 1 - \xi_i, \qquad \text{for} \quad \pi \in \mathcal{P}^*(\Gamma_i) \text{ and } i = 1, \dots, N. \tag{6}$$

For the first case we only have to consider a small number of constraints, while in the latter case we have an a potentially exponential number of constraints in the number of active nodes.

Let us assume the function f has parameters $\boldsymbol{\theta}$. Then we may solve the following optimization problem:

$$\min_{\boldsymbol{\theta}, \boldsymbol{\xi} \in \mathbb{R}^N_+} \mathbf{P}(\theta) + C \frac{1}{N} \sum_{i=1}^{N} \xi_i, \tag{7}$$

subject to constraints (4) and (5), or alternatively (4) and (6), where C is a regularization parameter and $\mathbf{P}$ a regularization term.

As usual we need to use column generation techniques in order to solve the above problem, i.e. we need to find unsatisfied constraints for a given function f. Except for the small number of constraints in (5), we need to take a Markov assumption: that the function f can be written as a sum of terms which only depend on the edges of the path (or runs of consecutive edges of limited length, for a higher order Markov property). Then we can use dynamic programming (DP) techniques in order to efficiently compute the maximal or minimal scoring paths to be added as constraint to the optimization problem.

Finding the worst scoring valid paths for the constraints (6) can easily be implemented by DP on the given correct graph Γ. Finding the best scoring invalid paths (cf. (4)) can be done by computing the k best paths[1] and removing all valid paths. However, if the number of valid paths is large, this technique becomes infeasible. To solve this problem one can extend the dynamic program to determine only such paths that use at least one edge that is not in the original graph and therefore is invalid. For example, one approach is to use dynamic programming with one bit of memory. This can be realized by duplicating each node, where one copy represents state 0 and the other copy represents state 1. State 0 edges are only connected by edges that are present in the original graph. Any other edge leads from a state 0 node to a state 1 node. The state 1 nodes are completely connected. Thus, any path ending in a state 1 node has at least one invalid edge.

A similar strategy can be applied in order to construct a cover of the graph using positively scoring paths, for instance during testing. Note that the number of paths that score positive can be exponential in the number of nodes. Here we propose an idea that generates at most a quadratic number of paths covering the graph: We start by generating the best scoring path. We mark all edges of the path as *used* and generate in the second iteration a path that uses at least one edge that was not used before (again, by adding one bit of memory). We continue to generate paths until the generated paths have a negative score.

3.3 Predictions using the Composite Path Representation

Using the multiple path representation one scores paths independently from each other. The scoring function can therefore not reward structures involving several parallel edges. Using the composite path representation we try to solve this problem. We assume that the width of the graph is at most d. Here we determine a function f that scores composite paths. Let $\mathcal{P}_c^*(\Gamma)$ be the set of composite paths of graph Γ and $\mathcal{P}_c^-(\Gamma)$ the set of invalid composite paths.

[1]This may lead to the problem that all paths that we choose are valid if k is not large enough. Hence we may need an iterative procedure that increases k for this case.

We pick one representative path $\pi^+ \in \mathcal{P}_c^*(\Gamma)$ that should score positive. All invalid composite paths should score negative. Hence, we would like to determine a function f such that

$$f(\pi_i^+) \geq 1 - \xi_i, \qquad \text{for} \quad i = 1, \ldots, N \tag{8}$$

$$f(\pi) \leq -1 + \xi_i, \qquad \text{for} \quad \pi \in \mathcal{P}_c^-(\Gamma_i) \text{ and } i = 1, \ldots, N. \tag{9}$$

As before, there are exponentially many negative constraints and we need a method to generate the highest scoring invalid composite path. Again, if the function f can be expressed as a sum of contributions of the edges of the composite path (also considering the edge labels), then one can employ dynamic programming in order to find the best scoring composite path. By considering the k best composite paths or by extending the dynamic program one may filter out all valid composite paths in order to obtain the best scoring invalid path.

4 Graph Learning with Subgraph Features

The second method that we propose first performs predictions on subgraphs of a fixed number of nodes, and then forms a consensus from the resulting overlapping and possibly inconsistent predictions. It will thereby ensure both sparsity (in the number of used edges) and the existence of paths through the graph.

Instead of decoding a graph with "hard" edges, i.e. $A_{ij} \in \{0, 1\}$, we relax the edges to be probabilistic, i.e. $A_{ij} \in [0, 1]$. This way we can make use of continuous optimization techniques to optimize over the adjacency matrix. To keep the notation simple we will treat the entire matrix A as variable, although not all its n^2 entries represent variables but only the $n(n-1)/2$ upper triangular ones.

4.1 Graph Decomposition by Subgraphs

The decoding problem considered here is to find a maximally scoring graph on V, where the scores are computed by a learned function. We assume that the scores of graphs are given by scores on subgraphs as detailed in the following. Let $\mathcal{S}$ be a set of node sets, i.e. $S \subset V, \quad \forall S \in \mathcal{S}$. It might eg contain all pairs of nodes, i.e. all potential edges. We assume that for each $S \in \mathcal{S}$ we are given a prediction of the subgraph defined by restricting the graph to S. Such fixed-size subgraphs can be predicted by standard methods, since all features relevant to the subgraph form a fixed-length vector.

We assume that for each $S \in \mathcal{S}$ we are given scores for all possible subgraphs on S. Formally, for each set S we are given a function $f_S : \{0, 1\}^{S \times S} \to \mathbb{R}$ that assigns a real-valued score to each possible configuration of edges on the nodes in S. The functions f_S need to be defined only on upper diagonal (or symmetric) adjacency matrices. Based on the subgraph scores, a single score can be assigned to every graph on V given by its adjacency matrix A:

$$f(A) \quad := \quad \sum_{S \in \mathcal{S}} f_S(A_{S \times S}) \quad , \tag{10}$$

where $A_{S \times S}$ is the adjacency matrix of the subgraph given by the restriction of A on S. To be able to work with relaxed adjacency matrices we extend the definition of the functions f_S (and thus also of f) to $[0, 1]$-valued inputs. We define the score of a relaxed subgraph to be the weighted sum of the scores of the hard subgraphs where the weights can be seen as probabilities of generating the corresponding hard graphs:

$$g_S(A_{S \times S}) \quad := \quad \sum_{B \in \{0,1\}^{S \times S}} f_S(B) \prod_{i,j \in S, \; i<j} A_{ij}^{B_{ij}} (1 - A_{ij})^{(1 - B_{ij})} \quad , \tag{11}$$

defining $0^0 := 1$. Consequently, a high scoring graph will maximize the function $g(A) := \sum_{S \in \mathcal{S}} g_S(A_{S \times S})$. Note that g is linear in the path scores.

By employing subgraph predictions, learning can be based on features attached to nodes, (potential) edges, and pairs or higher-order tuples of nodes and/or edges. Note that we need to limit $\mathcal{S}$ since it may otherwise grow exponentially in n. Limiting the size of subgraphs is a meaningful way to impose structure which can be exploited by a decoding algorithm. In the applications which we consider subgraphs of sizes beyond three appear rather impractical. Note that subgraphs of size two correspond to individual edges (or their absence).

4.2 Graph Decoding with Subgraph Features

Now it might seem natural to specify the decoding problem simply as the search for a maximally scoring graph subject to range constraints $0 \leq A_{ij} \leq 1$. However, there are potential problems with this simple approach. In the applications we consider, a meaningful graph should be connected in the sense that there exist paths from the start node, 1, to the end node, n. Also, edges that do not contribute to any such path are considered spurious. We now address these issues.

4.2.1 Enforcing Sparsity

In many applications we might want to enforce sparsity; both in the number of used edges and in the total weight distributed on them. Both can be achieved at the same time by simply adding an l_1-constraint, e.g.

$$\sum_{i<j} A_{ij} \quad \leq \quad maxedges \ . \tag{12}$$

Remember that we do not need to take absolute values, $|A_{ij}|$, thanks to the positivity constraints. Further, as a consequence of this sparsity enforcement we may drop the upper bounds on the edges, ie the constraints that $A_{ij} \leq 1$.

4.2.2 Enforcing Path Existence

So far, it is not guaranteed that the solution of the optimization problem will form a connected graph. However, in real applications it may be required that at least one path exists which connects the first node, 1, to the last node, n. We now introduce a technique to enforce such solutions.

Let the weight of a path be the product of its edge weights. We first note that the sum of the weights of all paths from 1 to i can be computed in a recursive way by the dynamic program

$$s_1 \quad = \quad 1 \ , \tag{13}$$

$$s_i \quad = \quad \sum_{j=1}^{i-1} w_{ij} s_j \ \ \forall i > 1 \ . \tag{14}$$

Thus s_n is the sum of weights of paths from 1 to n, and by bounding it away from zero we can ensure the existence of at least one path from 1 to n. In matrix notation, we have $s = As + e_1$, where e_i is the vector in $\mathbb{R}^n$ that has a 1 at the ith component and zeros everywhere else. It immediately follows that $s_n = e_n^\top (I - A)^{-1} e_1$. Thus we can add a constraint of the form

$$e_n^\top (I - A)^{-1} e_1 \quad \geq \quad minpaths \tag{15}$$

to our optimization problem.

4.2.3 Optimization

The optimization problem resulting from the above considerations can be written as follows:

$$\begin{array}{ll} \max_{A,T} & \sum_{S \in \mathcal{S}} g_S(A_{S \times S}) \\ s.t. & A_{ij} = \exp(T_{ij}) \ \ \forall 1 \leq i < j \leq n \\ & \sum_{i<j} A_{ij} \leq maxedges \\ & e_n^\top (I - A)^{-1} e_1 \geq minpaths \end{array} \tag{16}$$

Here we have replaced the explicit positivity constraints for A_{ij} by introducing a new matrix T and equating $A_{ij} = \exp(T_{ij})$. This is the first step to transform the constrained problem into an unconstrained one. Secondly, we remove both sparsity and path existence constraint and instead add a corresponding regularizer to the objective, thus obtaining

$$\begin{array}{ll} \max_{A,T} & \left(\sum_{S \in \mathcal{S}} g_S(A_{S \times S})\right) - \lambda \left(\sum_{i<j} A_{ij}\right) + \gamma \left(e_n^\top (I - A)^{-1} e_1\right) \\ s.t. & A_{ij} = \exp(T_{ij}) \ \ \forall 1 \leq i < j \leq n \end{array} \tag{17}$$

According to standard optimization theory, any solution for Equation (16) can be recovered by Equation (17) with a corresponding choice of λ and γ. Finally, an unconstrained differentiable minimization problem is obtained by substituting the equality constraints into the objective. The resulting optimization problem can easily be optimized by gradient descent methods. The path existence regularizer is in general not positive definite; however, the other parts of the objective are. This suggests the use of annealing methods for optimization.

4.3 Graph Learning with Subgraph Features

To embed the described decoder into the standard structured output learning framework we have to address two further issues. First, we need a decoder which produces the two best scoring graphs; in case the best scoring graph is correct, the second best graph is required for generating a new constraint. Here we can take advantage of the fact that the decoding optimization is non-convex: we run it several times from different starting points to retrieve several local optima and simply use the two best of them.

Second, we need to clarify how the subgraph scoring functions can be parameterized such that the parameters can be learned from the generated constraints. By construction, for any given A the objective is linear in the subgraph scores. Now the subgraph scores just have to be linear functions of any suitable features that can be defined on subgraphs; g will then be linear in these features and thus amenable to optimization in the structured output learning framework. Of course, in principle also the kernel trick can be applied here.

5 Evaluation

In order to empirically evaluate graph learning, a measure of success is needed. More precisely, we have to assess the similarity of a predicted graph with the corresponding true graph. Of course, we can easily assess predictions on the basis of individual edges. However, for our envisioned application to alternative splicing only paths from start node to end node are relevant. To take this into account we propose to compare two graphs based on the number of paths that they share or do not share. Formally, we count the paths incident to both graphs (true positives, TP), the paths incident to the predicted graph but not to the true graph (false positives, FP), and vice versa (false negatives, FN).

Since the number of paths incident to a graph is potentially exponential in the number n of nodes, we need an efficient way to count paths. We will make use of the fact that both the true graph and the predicted graph are defined on the same set of nodes (since this is assumed to be known in advance). Given the adjacency matrices A of the true graph and B of the prediction, we employ the following dynamic program (with $\mathbb{I}\{\cdot\}$ denoting the indicator function):

$$TP_i = \sum_{j=1}^{i-1} \mathbb{I}\{A_{ij} = 1 \wedge B_{ij} = 1\}\, TP_j \tag{18}$$

$$FP_i = \sum_{j=1}^{i-1} \mathbb{I}\{A_{ij} = 0 \wedge B_{ij} = 1\}\, (TP_j + FP_j) \tag{19}$$

$$FN_i = \sum_{j=1}^{i-1} \mathbb{I}\{A_{ij} = 1 \wedge B_{ij} = 0\}\, (TP_j + FN_j) \tag{20}$$

with the initialization $TP_1 = 1$, $FP_1 = FN_1 = 0$. It makes use of the directed structure of the graphs to recursively compute the counts of partial paths up to a given node i based on shorter paths.

Finally, we use the path counts to calculate two well-known figures of merit: $precision := TP/(TP+FP)$ and $recall := TP/(TP+FN)$. We prefer not to use the number of paths not incident to either graph (true negatives, TN) although it can also be efficiently calculated. The reason is that this figure depends on the number of nodes, which might themselves be subject to prediction. In particular, in gene prediction, the potential splice sites have to be identified before putative splice forms can be predicted. It might not be meaningful to consider candidate splice sites which turn out not to be used in the true splice forms as nodes; however, they would dramatically increase the number of possible paths.

6 Discussion

We presented three algorithms for decoding a graph with strictly forward pointing edges given the ordered set of nodes. This problem setting naturally arises for instance in the task of predicting alternative splice forms of genes. Of the three algorithms, the prediction of multiple paths seems to match the application best, since each path corresponds to one splice form. However, for blind predictions we might run into severe problems: if there exists an exponential number of positive paths it becomes intractable to extract them all. We now discuss the advantages and disadvantages of the three proposed approaches in terms of computational complexity and modelling power.

While the multiple path model can in principle describe sets of paths that cannot be generated by any single graph, at also has modelling limitations: it consideres each path individually, and cannot explicity model dependencies between different paths, as the other two approaches do. The composite path approach trades the possibility for higher-order Markov dependencies in terms of preceding edges against modeling dependencies between concurrent paths. The subgraph-based approach can take into account dependencies of both types to an arbitrarily bounded degree.

All proposed methods inevitably show a scaling behavior that is exponential in the size of the considered features. For the multiple path representation, the DP complexity grows with 2^d where d is the order of the Markov assumption. For the composite path, there is the same growth of complexity in the maximal width d of the considered graphs. Finally, the number of subgraph features also grows exponentially in their size.

There are two problems that are specific to the subgraph-based approach. First, in general there is no guarantee that all edges with positive weight are part of a path from atart node to end node. However, with appropriate settings of the parameters λ and γ it should always be possible to find such a valid solution. The second problem is that the decoding is only approximate: due to the non-convexity of 17 we may only obtain suboptimal solutions. The loss of the possibility to find optima by DP is clearly caused by moving from a set of "sequential" graphs (eg, with limited width) to the full set of graphs. However, as is argued in [10], possibly suboptimal decoding may be interlaced with learning to potentially speed up the learning dramatically with only minor loss in accuracy. Exploring this approach for graph prediction remains for future research.

References

[1] Hofmann, T., Tsochantaridis, I., Altun, Y.: Learning over structured output spaces via joint kernel functions. In: Proceedings of the Sixth Kernel Workshop. (2002)

[2] Taskar, B., Guestrin, C., Koller, D.: Max-Margin Markov Networks. In: Neural Information Processing Systems Conference (NIPS03). (2003)

[3] Tsochantaridis, I., Hofmann, T., Joachims, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: Proceedings of the International Conference on Machine Learning. (2004)

[4] Altun, Y., Tsochantaridis, I., Hofmann, T.: Hidden Markov support vector machines. In: Proceedings of the International Conference on Machine Learning. (2003)

[5] Kudo, T., Maeda, E., Matsumoto, Y.: An application of boosting to graph classification. In Saul, L.K., Weiss, Y., Bottou, L., eds.: Advances in Neural Information Processing Systems 17. MIT Press, Cambridge, MA (2005) 729–736

[6] Gärtner, T.: Exponential and geometric kernels for graphs. In: NIPS*02 Workshop on Unreal Data: Principles of Modeling Nonvectorial Data. (2002) Available from http://mlg.anu.edu.au/unrealdata/.

[7] Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: Proceedings of the 20th International Conference on Machine Learning, Menlo Park, CA, AAAI Press (2003) 321–328

[8] Hancock, E.R., Luo, B.: Pattern vectors from algebraic graph theory. IEEE Trans. Pattern Anal. Mach. Intell. **27**(7) (2005) 1112–1124 Member-Richard C. Wilson.

[9] Bakir, G.H., Zien, A., Tsuda, K.: Learning to Find Graph Pre-Images. In Rasmussen, C.E., Bülthoff, H.H., Giese, M.A., Schölkopf, B., eds.: Pattern Recognition, Proceedings of the 26th DAGM Symposium, Springer, Berlin (2004) 253–261

[10] Daumé III, H., Langford, J., Marcu, D.: Search-based structured prediction. Available from http://pub.hal3.name/#daume06searn (2006)