On Graph Kernels: Hardness Results and Efficient Alternatives

Conference Paper *in* Lecture Notes in Computer Science · January 2003

DOI: 10.1007/978-3-540-45167-9_11 · Source: DBLP

3 authors, including:

Peter A. Flach
University of Bristol
321 PUBLICATIONS **10,549** CITATIONS

SEE PROFILE

On Graph Kernels: Hardness Results and Efficient Alternatives

Thomas Gärtner[1,2,3], Peter Flach[3], and Stefan Wrobel[1,2]

[1] Fraunhofer Institut Autonome Intelligente Systeme, Germany
{Thomas.Gaertner,Stefan.Wrobel}@ais.fraunhofer.de
[2] Department of Computer Science III, University of Bonn, Germany
[3] Department of Computer Science, University of Bristol, UK
Peter.Flach@bristol.ac.uk

Abstract. As most 'real-world' data is structured, research in kernel methods has begun investigating kernels for various kinds of structured data. One of the most widely used tools for modeling structured data are graphs. An interesting and important challenge is thus to investigate kernels on instances that are represented by graphs. So far, only very specific graphs such as trees and strings have been considered.
This paper investigates kernels on labeled directed graphs with general structure. It is shown that computing a strictly positive definite graph kernel is at least as hard as solving the graph isomorphism problem. It is also shown that computing an inner product in a feature space indexed by all possible graphs, where each feature counts the number of subgraphs isomorphic to that graph, is *NP*-hard. On the other hand, inner products in an alternative feature space, based on walks in the graph, can be computed in polynomial time. Such kernels are defined in this paper.

1 Introduction

Support vector machines [1] are among the most successful recent developments within the machine learning community. Along with some other learning algorithms they form the class of kernel methods [15]. The computational attractiveness of kernel methods is due to the fact that they can be applied in high dimensional feature spaces without suffering from the high cost of explicitly computing the feature map. This is possible by using a positive definite kernel k on any set $\mathcal{X}$. For such $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ it is known that a map $\phi : \mathcal{X} \to \mathcal{H}$ into a Hilbert space $\mathcal{H}$ exists, such that $k(x, x') = \langle \phi(x), \phi(x') \rangle$ for all $x, x' \in \mathcal{X}$.

Kernel methods have so far successfully been applied to various tasks in attribute-value learning. Much 'real-world' data, however, is structured – there is no natural representation of the instances of the learning problem as a tuple of constants. In computer science graphs are a widely used tool for modeling structured data. They can be used, for example, as a representation for molecules.

Unfortunately, due to the powerful expressiveness of graphs, defining appropriate kernel functions for graphs has proven difficult. In order to control the

B. Schölkopf and M.K. Warmuth (Eds.): COLT/Kernel 2003, LNAI 2777, pp. 129–143, 2003.

complexity of such kernels, one line of existing research has concentrated on special kinds of graphs, in particular, trees [2] or strings [17, 14] which results in efficient kernels, but loses most of the power of general graphs. More recently, [4, 9] have investigated efficient kernels for general graphs based on particular kinds of walks, which captures more, but still far from all of the structure of the graph. An interesting open problem is thus whether approaches such as the latter can be further generalized, and if so, if it is possible to define kernels that take the entire structure of graphs into account.

In this paper, we give answers to both of these questions. Firstly, we prove that computing any kernel function that is capable of fully recognizing the structure of graphs (using subgraph-isomorphism) is *NP*-hard, making it extremely unlikely that efficient kernels can be found. Secondly, we show that nonetheless there is room for improvement by presenting a generalized family of kernels based on walks which includes the kernels proposed in [4, 9] as special cases while still being polynomially computable.

The outline of the paper is as follows: Section 1.1 first gives an intuitive overview over the important results of this paper. Section 2 gives then a brief overview of graph theoretic concepts. Section 3 defines what might be the ideal graph kernel but also shows that such kernels cannot be computed in polynomial time. Section 4 and 5 introduce alternative graph kernels. Section 6 shows how these kernels can be extended and computed efficiently. Finally, section 7 discusses related work and section 8 concludes with future work.

1.1 Motivation and Approach

Consider a graph kernel that has one feature Φ_H for each possible graph H, each feature $\Phi_H(G)$ measuring how many subgraphs of G have the same structure as graph H. Using the inner product in this feature space, graphs satisfying certain properties can be identified. In particular, one could decide whether a graph has a Hamiltonian path, i.e., a sequence of adjacent vertices and edges that contains every vertex and edge exactly once. Now this problem is known to be *NP*-hard, i.e., it is strongly believed that this problem can not be solved in polynomial time. Thus we need to consider alternative graph kernels.

We investigate mainly two alternative approaches. One is to define a similarity of two graphs based on the length of all walks between each pair of vertices in the graph. The other is to measure the number of times given label sequences occur in this graph. The inner product in this feature space can be computed directly, by first building the product graph and then computing the limit of a matrix power series of the adjacency matrix. In both cases the computation is possible in polynomial time.

To illustrate these kernels, consider a simple graph with four vertices labeled 'c', 'a', 'r', and 't', respectively. We also have four edges in this graph: one from the vertex labeled 'c' to the vertex labeled 'a', one from 'a' to 'r', one from 'r' to 't', and one from 'a' to 't'. The non-zero features in the label pair feature space are $\phi_{c,c} = \phi_{a,a} = \phi_{r,r} = \phi_{t,t} = \lambda_0$, $\phi_{c,a} = \phi_{a,r} = \phi_{r,t} = \lambda_1$, $\phi_{a,t} = \lambda_1 + \lambda_2$, $\phi_{c,r} = \lambda_2$, and $\phi_{c,t} = \lambda_2$. The non-zero features in the label sequence feature space are

$\phi_c = \phi_a = \phi_r = \phi_t = \sqrt{\lambda_0}$, $\phi_{ca} = \phi_{ar} = \phi_{at} = \phi_{rt} = \sqrt{\lambda_1}$, $\phi_{car} = \phi_{cat} = \lambda_2$, and $\phi_{cart} = \sqrt{\lambda_3}$. The λ_i are user defined weights and the square-roots appear only to make the computation of the kernel more elegant. In particular, we show how closed forms of the inner products in this feature space can be computed for exponential and geometric choises of λ_i.

2 Graphs

This section gives a brief overview of graphs. For a more in-depth discussion of these and related concepts the reader is referred to [3, 13].

2.1 Basic Terminology and Notation

Generally, a *graph* G is described by a finite set of *vertices* $\mathcal{V}$, a finite set of *edges* $\mathcal{E}$, and a function Ψ. For *hypergraphs* this function maps each edge to a set of vertices $\Psi : \mathcal{E} \to \{X \subseteq \mathcal{V}\}$. For *undirected* graphs the codomain of the function is restricted to sets of vertices with two elements only $\Psi : \mathcal{E} \to \{X \subseteq \mathcal{V} : |X| = 2\}$. For *directed* graphs the function maps each edge to the tuple consisting of its initial and terminal node $\Psi : \mathcal{E} \to \{(u, v) \in \mathcal{V} \times \mathcal{V}\}$. Edges e in a directed graph for which $\Psi(e) = (v, v)$ are called s. Two edges e, e' are *parallel* if $\Psi(e) = \Psi(e')$. We will sometimes assume some enumeration of the vertices in a graph, i.e., $\mathcal{V} = \{\nu_i\}_{i=1}^{n}$ where $n = |\mathcal{V}|$.

For *labeled* graphs there is additionally a set of labels $\mathcal{L}$ along with a function *label* assigning a label to each edge and/or vertex. In *edge-labeled* graphs, labels are assigned to edges only; in *vertex-labeled* graphs, labels are assigned to vertices only; and in *fully-labeled* graphs, labels are assigned to edges and vertices. It is useful to have some enumeration of all possible labels at hand, i.e., $\mathcal{L} = \{\ell_r\}_{r \in \mathbb{N}}$ [1].

In this paper we are mainly concerned with labeled directed graphs without parallel edges. In this case we can – for simplicity of notation – identify an edge with its image under the map Ψ. In what follows 'graph' will always refer to labeled directed graphs without parallel edges. Each graph will be described by a two-tuple $G = (\mathcal{V}, \mathcal{E})$ such that $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. To refer to the vertex and edge set of a specific graph we will use the notation $\mathcal{V}(G), \mathcal{E}(G)$. Wherever we distinguish two graphs by their subscript (G_i) or some other symbol (G', G^*) the same notation will be used to distinguish their vertex and edge sets.

A generalization of the concepts and functions described in this paper to graphs with parallel edges is straightforward, however, directly considering them in this paper would obscure notation. Graphs with undirected edges can – for all purposes of this paper – be identified with a directed graph that has two edges for each edge in the undirected graph. Graphs without labels can be seen as a special case of labeled graphs where the same label is assigned to each edge and/or vertex. Hypergraphs are not considered in this paper.

[1] While ℓ_1 will be used to always denote the same label, l_1 is a variable that can take different values, e.g., $\ell_1, \ell_2, \ldots$. The same holds for vertex ν_1 and variable v_1.

Last but not least we need to define some special graphs. A *walk*[2] w is a sequence of vertices $w = v_1, v_2, \ldots v_{n+1}$; $v_i \in \mathcal{V}$ such that $(v_i, v_{i+1}) \in \mathcal{E}$. The *length* of the walk is equal to the number of edges in this sequence, i.e., n in the above case. A *path* is a walk in which $v_i \neq v_j \Leftrightarrow i \neq j$. A *cycle* is a path with $(v_{n+1}, v_1) \in \mathcal{E}$.

A graph $G = (\mathcal{V}, \mathcal{E})$ is called *connected* if there is a walk between any two vertices in the following graph: $(\mathcal{V},\ \mathcal{E} \cup \{(u,v) : (v,u) \in \mathcal{E}\})$. For a graph $G = (\mathcal{V}(G), \mathcal{E}(G))$, we denote by $G[\mathcal{V}^*]$ the subgraph *induced* by the set of vertices $\mathcal{V}^* \subseteq \mathcal{V}(G)$, that is $G[\mathcal{V}^*] = (\mathcal{V}^*, \{(u,v) \in \mathcal{E}(G) : u, v \in \mathcal{V}^*\})$. A *subgraph* of G is a graph $H = (\mathcal{V}(H), \mathcal{E}(H))$ with $\mathcal{V}(H) \subseteq \mathcal{V}(G)$ and $\mathcal{E}(H) \subseteq \mathcal{E}(G[\mathcal{V}(H)])$. A subgraph H of G is *proper* if $\mathcal{V}(H) \subset \mathcal{V}(G)$; it is *spanning* if $\mathcal{V}(H) = \mathcal{V}(G)$. If a path or a cycle is a subgraph of a graph G, it is often called a walk or cycle in G. A spanning path in G is called a *Hamiltonian path*; a spanning cycle in G is called a *Hamiltonian cycle*.

2.2 Matrix Notation and Some Functions

For the description of our graph kernels it turns out to be useful to have a matrix representation for (labeled directed) graphs. Let $[A]_{ij}$ denote the element in the i-th row and j-th column of matrix A. For two $m \times n$ matrices A, B (given by $[A]_{ij}, [B]_{ij} \in \mathbb{R}$) the inner product is defined as $\langle A, B \rangle = \sum_{i,j} [A]_{ij}[B]_{ij}$. Furthermore, $\mathbf{I}$ denotes the identity matrix; $\mathbf{0}, \mathbf{1}$ denote matrices with all elements equal to $0, 1$, respectively.

A graph G can uniquely be described by its label and adjacency matrices. The label matrix L is defined by $[L]_{ri} = 1 \Leftrightarrow \ell_r = \mathit{label}(\nu_i)$, $[L]_{ri} = 0 \Leftrightarrow \ell_r \neq \mathit{label}(\nu_i)$. The adjacency matrix E is defined by $[E]_{ij} = 1 \Leftrightarrow (\nu_i, \nu_j) \in \mathcal{E}$, $[E]_{ij} = 0 \Leftrightarrow (\nu_i, \nu_j) \notin \mathcal{E}$. We also need to define some functions describing the neighborhood of a vertex v in a graph G: $\delta^+(v) = \{(v,u) \in \mathcal{E}\}$ and $\delta^-(v) = \{(u,v) \in \mathcal{E}\}$. Here, $|\delta^+(v)|$ is called the *outdegree* of a vertex and $|\delta^-(v)|$ the *indegree*. Furthermore, the maximal indegree and outdegree are denoted by $\Delta^-(G) = max\{|\delta^-(v)|, v \in \mathcal{V}\}$ and $\Delta^+(G) = max\{|\delta^+(v)|, v \in \mathcal{V}\}$, respectively. It is clear that the maximal indegree equals the maximal column sum of the adjacency matrix and that the maximal outdegree equals the maximal row sum of the adjacency matrix. For $a \geq \min\{\Delta^+(G), \Delta^-(G)\}$, a^n is an upper bound on each component of the matrix E^n. This will be useful to determine the convergence properties of some graph kernels.

2.3 Interpretation of Matrix Powers

First consider the diagonal matrix $LL^\top$. The i-th element of the diagonal of this matrix, i.e. $[LL^\top]_{ii}$, corresponds to the number of times label ℓ_i is assigned to a vertex in the graph. Now consider the matrix E. The component $[E]_{ij}$ describes whether there is an edge between vertex ν_i and ν_j. Now we combine the label

[2] What we call 'walk' is sometimes called an 'edge progression'.

and adjacency matrix as $LEL^\top$. Each component $[LEL^\top]_{ij}$ corresponds to the number of edges between vertices labeled ℓ_i and vertices labeled ℓ_j.

Replacing the adjacency matrix E by its n-th power ($n \in \mathbb{N}, n \geq 0$), the interpretation is quite similar. Each component $[E^n]_{ij}$ of this matrix gives the number of walks of length n from vertex ν_i to ν_j. Multiplying this with the label matrix, we obtain the matrix $LE^nL^\top$. Each component $[LE^nL^\top]_{ij}$ now corresponds to the number of walks of length n between vertices labeled ℓ_i and vertices labeled ℓ_j.

2.4 Product Graphs

Product graphs [8] are a very interesting tool in discrete mathematics. The four most important graph products are the Cartesian, the strong, the direct, and the lexicographic product. While the most fundamental one is the Cartesian graph product, in our context the direct graph product is the most important ones.

Usually, graph products are defined on unlabeled graphs. However, in many real-world machine learning problems it could be important to be able to deal with labeled graphs. We extend the definition of graph products to labeled graphs and give the relevant definition in the appropriate places in this paper.

3 Complete Graph Kernels

In this section, all vertices and edges are assumed to have the same label. If there is no polynomial time algorithm for this special case then there is obviously no polynomial time algorithm for the general case.

When considering the set of all graphs $\mathcal{G}$, many graphs in this set differ only in the enumeration of vertices, and thus edges, and not in their structure: these graphs are *isomorphic*. Since usually in learning, the names given to vertices in different graphs have no meaning, we want kernels not to distinguish between isomorphic graphs. Formally, two graphs G, H are isomorphic if there is a bijection $\psi : \mathcal{V}(G) \to \mathcal{V}(H)$ such that for all $(u, v) \in \mathcal{E}(G) \Leftrightarrow (\psi(u), \psi(v)) \in \mathcal{E}(H)$. We denote that G, H are isomorphic by $G \simeq H$. In the remainder of this paper we define all kernels and maps on the quotient set of the set of all graphs with respect to isomorphism, i.e., the set of equivalence classes. To keep the notation simple, we will continue to refer to this set as $\mathcal{G}$, and also refer to each equivalence class simply by one of its representative graphs.

While it is easy to see that graph isomorphism is in *NP* it is – in spite of a lot of research devoted to this question – still not known whether graph isomorphism is in *P* or if it is *NP*-complete. It is believed that graph isomorphism lies between *P* and *NP*-complete [11].

The first class of graph kernels we are going to consider is those kernels that allow to distinguish between all (non-isomorphic) graphs in feature space. If a kernel does not allow us to distinguish between two graphs then there is no way any learning machine based on this kernel function can separate these two graphs. Investigating the complexity of graph kernels that distinguish between all graphs is thus an interesting problem.

Definition 1. *Let $\mathcal{G}$ denote the set of all graphs and let $\Phi : \mathcal{G} \to \mathcal{H}$ be a map from this set into a Hilbert space $\mathcal{H}$. Furthermore, let $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ be such that $\langle \Phi(G), \Phi(G') \rangle = k(G, G')$. If Φ is injective, k is called a* complete *graph kernel.*

Proposition 1. *Computing any complete graph kernel is at least as hard as deciding whether two graphs are isomorphic.*

Proof. Let all functions be as in definition 1. As Φ is injective, $k(G, G) - 2k(G, G') + k(G', G') = \langle \Phi(G) - \Phi(G'), \Phi(G) - \Phi(G') \rangle = 0$ if and only if $G \simeq G'$. □

It is well known that there are polynomial time algorithms to decide isomorphism for several restricted graph classes [11], for example, planar graphs. However, considering kernels on restricted graph classes is beyond the scope of this paper. The remaining question for us is whether the above complexity result is tight, i.e., if there is a complete graph kernel that is (exactly) as hard as deciding graph isomorphism. This is obvious considering the kernel $k(G, G') = 1 \Leftrightarrow G \simeq G'$ and $k(G, G') = 0 \Leftrightarrow G \not\simeq G'$. Now the following corollary is interesting.

Corollary 1. *Computing any strictly positive definite graph kernel is at least as hard as deciding whether two graphs are isomorphic.*

Proof. This follows directly from proposition 1 and strictly positive definite graph kernels being complete. □

We will now look at another interesting class of graph kernels. Intuitively, it is useful to base the similarity of two graphs on their common subgraphs.

Definition 2. *Let $\mathcal{G}$ denote the set of all graphs and let λ be a sequence $\lambda_1, \lambda_2, \ldots$ of weights ($\lambda_n \in \mathbb{R}; \lambda_n > 0$ for all $n \in \mathbb{N}$). The subgraph feature space is defined by the map $\Phi : \mathcal{G} \to \mathcal{H}$ into the Hilbert space $\mathcal{H}$ with one feature Φ_H for each connected graph $H \in \mathcal{G}$, such that for every graph $G \in \mathcal{G}$*

$$\Phi_H(G) = \lambda_{|\mathcal{E}(H)|} \left| \{ G' \text{ is subgraph of } G : G' \simeq H \} \right|$$

Clearly, the inner product in the above feature space is a complete graph kernel and thus computing the inner product is at least as hard as solving the graph isomorphism problem. However, we are able to show an even stronger result.

Proposition 2. *Computing the inner product in the subgraph feature space is NP-hard.*

Proof. Let $P_n \in \mathcal{G}$ be the path graph with n edges and let $\mathbf{e}_H$ be a vector in the subgraph feature space such that the feature corresponding to graph H equals 1 and all other features equal 0. Let G be any graph with m vertices. As $\{\Phi(P_n)\}_{n \in \mathbb{N}}$ is linearly independent, there are $\alpha_1, \ldots, \alpha_m$ such that $\alpha_1 \Phi(P_1) + \ldots + \alpha_m \Phi(P_m) = \mathbf{e}_{P_m}$. These $\alpha_1, \ldots, \alpha_m$ can be found in polynomial time, as in each image of a path P_n under the map Φ only n features are different from 0. Then, $\alpha_1 \langle \Phi(P_1), \Phi(G) \rangle + \ldots + \alpha_m \langle \Phi(P_m), \Phi(G) \rangle > 0$ if and only if G has a Hamiltonian path. However, it is well known that the decision problem whether a graph has a Hamiltonian path is *NP*-complete. □

A first approach to defining graph kernels for which there is a polynomial time algorithm might be to restrict the feature space of Φ to features Φ_H where H is a member of a restricted class of graphs. However, even if H is restricted to paths the above proof still applies. Closely related to the Hamiltonian path problem is the problem of finding the longest path in a graph. This problem is known to be *NP*-complete even on (most) restricted graph classes [16]. Thus even restricting the domain of Φ is not likely to improve the computational complexity.

The results shown in this section indicate that it is intractable to compute complete graph kernels and inner products in feature spaces made up by graphs isomorphic to subgraphs. Our approach to define polynomial time computable graph kernels is to have the feature space be made up by graphs homomorphic to subgraphs. In the remainder of this paper we will thus concentrate on walks instead of paths in graphs.

4 Kernels Based on Label Pairs

In this section we consider vertex-labeled graphs only. In some applications there is reason to suppose that only the distance between (all) pairs of vertices of some label has an impact on the classification of the graph. In such applications we suggest the following feature space.

Definition 3. *Let $\mathcal{W}_n(G)$ denote the set of all possible walks with n edges in G and let λ be a sequence $\lambda_0, \lambda_1, \ldots$ of weights ($\lambda_n \in \mathbb{R}; \lambda_n \geq 0$ for all $n \in \mathbb{N}$). For a given walk $w \in \mathcal{W}_n(G)$ let $l_1(w)$ denote the label of the first vertex of the walk and $l_{n+1}(w)$ denote the label of the last vertex of the walk.*

The label pair feature space is defined by one feature ϕ_{ℓ_i,ℓ_j} for each pair of labels ℓ_i, ℓ_j:

$$\phi_{\ell_i,\ell_j}(G) = \sum_{n=0}^{\infty} \lambda_n \left|\{w \in \mathcal{W}_n(G) : l_1(w) = \ell_i \wedge l_{n+1}(w) = \ell_j\}\right|$$

Let all functions and variables be defined as in definition 3. The key to efficient computation of the kernel corresponding to the above feature map is the following equation:

$$\langle \phi(G), \phi(G') \rangle = \left\langle L \left(\sum_{i=0}^{\infty} \lambda_i E^i \right) L^\top, L' \left(\sum_{j=0}^{\infty} \lambda_j E'^j \right) L'^\top \right\rangle$$

As we are only interested in cases in which $||\phi(G)||$ is finite, we generally assume that $\sum_i \lambda_i a^i$ for $a = \min\{\Delta^+(G), \Delta^-(G)\}$ converges. To compute this graph kernel, it is then necessary to compute the above matrix power series. See section 6.1 for more details.

Although the map ϕ_{ℓ_i,ℓ_j} is injective if the function *label* is injective, the dimensionality of the label pair feature space is low, if the number of different

labels is low. In particular the dimensionality of the label pair feature space equals the number of different labels squared (that is $|\mathcal{L}|^2$). In domains in which only few labels occur, this might be a feature space of too low dimension.

One obvious way to achieve a higher dimensional – and thus more expressive – feature space is to use a more expressive label set including, for example, some information about the neighborhood of the vertices. Still, this manual enrichment of the feature space is not in all cases desired. For that reason, in the next section we describe a kernel function that operates in a more expressive feature space. The key idea of the kernel is to have each dimension of the feature space correspond to one particular label sequence. Thus even with very few – and even with a single – labels, the feature space will already be of infinite dimension.

5 Kernels Based on Contiguous Label Sequences

In this section we consider graphs with labels on vertices and/or edges. In the presence of few labels, the kernel described in the previous section suffers from too little expressivity. The kernel described in this section overcomes this by defining one feature for every possible label sequence and then counting how many walks in a graph match this label sequence. In order not to have to distinguish all three cases of edge-labeled, vertex-labeled, and fully-labeled graphs explicitly, we extend the domain of the function *label* to include all vertices and edges. In edge-labeled graphs we define $label(v) = \#$ for all vertices v and in vertex-labeled graphs we define $label(u, v) = \#$ for all edges (u, v).

We begin by defining the feature space of contiguous (or unbroken) label sequences and the *direct graph product*, central to the further developments in this section.

Definition 4. *Let $\mathcal{S}_n$ denote the set of all possible label sequences of walks with n edges and let λ be a sequence $\lambda_0, \lambda_1, \ldots$ of weights ($\lambda_i \in \mathbb{R}; \lambda_i \geq 0$ for all $i \in \mathbb{N}$). Furthermore, let $\mathcal{W}_n(G)$ denote the set of all possible walks with n edges in graph G. For a given walk $w \in \mathcal{W}_n(G)$ let $l_i(w)$ denote the i-th label of the walk.*

The sequence feature space is defined by one feature for each possible label sequence. In particular, for any given length n and label sequence $s = s_1, \ldots, s_{2n+1}$; $s \in \mathcal{S}_n$, the corresponding feature value for every graph G is:

$$\phi_s(G) = \sqrt{\lambda_n}\,|\{w \in \mathcal{W}_n(G), \forall\, i :\ s_i = l_i(w)\}|$$

Definition 5. *We denote the direct product of two graphs $G_1 = (\mathcal{V}_1, \mathcal{E}_1), G_2 = (\mathcal{V}_2, \mathcal{E}_2)$ by $G_1 \times G_2$. The vertex and edge set of the direct product are respectively defined as:*

$$\mathcal{V}(G_1 \times G_2) = \{(v_1, v_2) \in \mathcal{V}_1 \times \mathcal{V}_2 : (label(v_1) = label(v_2))\}$$
$$\mathcal{E}(G_1 \times G_2) = \{((u_1, u_2), (v_1, v_2)) \in \mathcal{V}^2(G_1 \times G_2) : (u_1, v_1) \in \mathcal{E}_1 \wedge (u_2, v_2) \in \mathcal{E}_2 \wedge (label(u_1, v_1) = label(u_2, v_2))\}$$

A vertex (edge) in graph $G_1 \times G_2$ has the same label as the corresponding vertices (edges) in G_1 and G_2.

Before giving the definition of direct product graph kernels we will now describe and interpret some properties of the product graph. The following proposition relates the number of times a label sequence occurs in the product graph to the number of times it occurs in each factor.

Proposition 3. *Let all variables and functions be defined as in definition 4 and 5. Furthermore, let G, G' be two graphs. Then*

$$\begin{aligned}&|\{w \in \mathcal{W}_n(G \times G'), \forall i : \ s_i = l_i(w)\}| \\ =&|\{w \in \mathcal{W}_n(G), \forall i : \ s_i = l_i(w)\}| \cdot |\{w \in \mathcal{W}_n(G'), \forall i : \ s_i = l_i(w)\}|\end{aligned}$$

Proof. It is sufficient to show a bijection between every walk in the product graph and one walk in both graphs such that their label sequences match.

Consider first a walk in the product graph $w^* \in \mathcal{W}_n(G \times G')$:

$$w^* = (v_1, v_1'), (v_2, v_2'), \ldots, (v_n, v_n')$$

with $(v_i, v_i') \in \mathcal{V}(G \times G')$. Now let $w = v_1, v_2, \ldots, v_n$ and $w' = v_1', v_2', \ldots, v_n'$. Clearly $w \in \mathcal{W}_n(G)$, $w' \in \mathcal{W}_n(G')$, and

$$\forall i \ : \ l_i(w^*) = l_i(w) = l_i(w')$$

The opposite holds as well: For every two walks $w \in \mathcal{W}_n(G)$, $w' \in \mathcal{W}_n(G')$ with matching label sequences, there is a walk $w^* \in \mathcal{W}_n(G \times G')$ with a label sequence that matches the label sequences of w and w'. □

Having introduced product graphs and having shown how these can be interpreted, we are now able to define the direct product kernel.

Definition 6. *Let G_1, G_2 be two graphs, let $E_\times$ denote the adjacency matrix of their direct product $E_\times = E(G_1 \times G_2)$, and let $\mathcal{V}_\times$ denote the vertex set of the direct product $\mathcal{V}_\times = \mathcal{V}(G_1 \times G_2)$. With a sequence of weights $\lambda = \lambda_0, \lambda_1, \ldots$ ($\lambda_i \in \mathbb{R}; \lambda_i \geq 0$ for all $i \in \mathbb{N}$) the direct product kernel is defined as*

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|\mathcal{V}_\times|} \left[\sum_{n=0}^{\infty} \lambda_n E_\times^n \right]_{ij}$$

if the limit exists.

Proposition 4. *Let ϕ be as in definition 4 and $k_\times$ as in definition 6. For any two graphs G, G', $k_\times(G, G') = \langle \phi(G), \phi(G') \rangle$*

Proof. This follows directly from proposition 3. □

To compute this graph kernel, it is then necessary to compute the above matrix power series. See section 6.1 for more details.

6 Extensions and Computation

This section shows how the limits of matrix power series can efficiently be computed, how the concepts presented above can be extended to graphs with transition probabilities, and how a kernel based on non-contiguous label sequences can be computed.

6.1 Computation of Matrix Power Series

In this section we show how matrix power series of the type $\lim_{n\to\infty}\sum_{i=0}^{n}\lambda_i E^i$ can efficiently be computed for some choices of λ.

Exponential Series Similar to the exponential of a scalar value ($e^b = 1 + b/1! + b^2/2! + b^3/3! + \ldots$) the exponential of the square matrix E is defined as

$$e^{\beta E} = \lim_{n\to\infty}\sum_{i=0}^{n}\frac{(\beta E)^i}{i!}$$

where we use $\frac{\beta^0}{0!} = 1$ and $E^0 = \mathbf{I}$. Feasible exponentiation of matrices in general requires diagonalising the matrix. If the matrix E can be diagonalized such that $E = T^{-1}DT$ we can easily calculate arbitrary powers of the matrix as $E^n = (T^{-1}DT)^n = T^{-1}D^nT$ and for a diagonal matrix we can calculate the power component-wise $[D^n]_{i,i} = [D_{i,i}]^n$. Thus $e^{\beta E} = T^{-1}e^{\beta D}T$ where $e^{\beta D}$ is calculated component-wise. Once the matrix is diagonalized, computing the exponential matrix can be done in linear time. Matrix diagonalization is a matrix eigenvalue problem and such methods have roughly cubic time complexity.

Geometric Series The geometric series $\sum_i \gamma^i$ is known to converge if and only if $|\gamma| < 1$. In this case the limit is given by $\lim_{n\to\infty}\sum_{i=0}^{n}\gamma^i = \frac{1}{1-\gamma}$. Similarly, we define the geometric series of a matrix as

$$\lim_{n\to\infty}\sum_{i=0}^{n}\gamma^i E^i$$

if $\gamma < 1/a$, where $a \geq \min\{\Delta^+(G), \Delta^-(G)\}$ as above. Feasible computation of the limit of a geometric series is possible by inverting the matrix $\mathbf{I} - \gamma E$. To see this, let $(\mathbf{I} - \gamma E)x = 0$, thus $\gamma Ex = x$ and $(\gamma E)^i x = x$. Now, note that $(\gamma E)^i \to 0$ as $i \to \infty$. Therefore $x = 0$ and $\mathbf{I} - \gamma E$ is regular. Then $(\mathbf{I} - \gamma E)(\mathbf{I} + \gamma E + \gamma^2 E^2 + \cdots) = \mathbf{I}$ and $(\mathbf{I} - \gamma E)^{-1} = (\mathbf{I} + \gamma E + \gamma^2 E^2 + \cdots)$ is obvious. Matrix inversion is roughly of cubic time complexity.

6.2 Transition Graphs

In some cases graphs are employed to model discrete random processes such as Markov chains [7]. In these cases a *transition probability* is assigned to each edge.

We only consider the case that the transition probability does not change over time. Such graphs will be called *transition graphs*. In transition graphs vertices are often called *states*. We denote the probability of going from vertex u to v by $p_{(u,v)}$. More precisely this denotes the probability of the process being in state v at time $t+1$ given that it was in state u at time t. Usually, transitions are without loss, i.e., $\forall\, u \in \mathcal{V} : \sum_{(u,v)\in\mathcal{E}} p_{(u,v)} = 1$. In some cases, there is a probability p_{stop} that the process stops at any time.

In order to deal with graphs modeling random processes, we replace the adjacency matrix E of a graph by the transition matrix R with $[R]_{ij} = p_{(\nu_i,\nu_j)}(1-p_{\text{stop}})$ if $(\nu_i,\nu_j) \in \mathcal{E}$ and $[R]_{ij} = 0$ otherwise. Without loss of generality we assume $p_{(\nu_i,\nu_j)} > 0 \Leftrightarrow (\nu_i,\nu_j) \in \mathcal{E}$. Before we can apply the kernel introduced in the previous section to transition graphs we have to redefine the functions $\Delta^+(G), \Delta^-(G)$ to be the maximal row and column sum of the matrix R of a graph G, respectively. Clearly $\Delta^+(G), \Delta^-(G) \leq 1$.

If we use the transition matrix R instead of the adjacency matrix, we get to a similar interpretation. $[R^n]_{ij}$ determines then the probability of getting from vertex ν_i to vertex ν_j in n steps. The interpretation of $[LR^nL^\top]_{ij}$ is a bit more complicated. If we divide by the number of times label ℓ_i occurs, however, interpretation becomes easy again. Thus consider $[LR^nL^\top]_{ij}/[LL^\top]_{ii}$. This is the probability that having started at any vertex labeled ℓ_i and taking n steps, we arrive at any vertex labeled ℓ_j. The division by $[LL^\top]_{ii}$ can be justified by assuming a uniform distribution for starting at a particular vertex with label ℓ_i.

A graph with some transition matrix R and stopping probability $p_{\text{stop}} = 0$ can be interpreted as a *Markov chain*. A vertex v with a transition probability $p_{(v,v)} = 1$ in a Markov chain is called *absorbing*. An absorbing Markov chain is a Markov chain with a vertex v such that v is absorbing and there is a walk from any vertex u to the absorbing vertex v. It is known [7] that in absorbing Markov chains the limit of R^n for $n \to \infty$ is $\mathbf{0}$. If we define $N = \mathbf{I} + R + R^2 + \cdots$ then $[N]_{ij}$ is the expected number of times the chain is in vertex ν_j given that it starts in vertex ν_i. N can be computed as the inverse of the matrix $\mathbf{I} - R$.

In the case of graphs with transition probabilities on the edges, the edges in the product graph have probability $p_{(u_{12},v_{12})} = p_{(u_1,v_1)}p_{(u_2,v_2)}$ where $u_{12} = (u_1,u_2) \in \mathcal{V}(G_1 \times G_2)$ and $v_{12} = (v_1,v_2) \in \mathcal{V}(G_1 \times G_2)$. Let p_1, p_2 denote the stopping probability in graphs G_1, G_2 respectively. The stopping probability p_{12} in the product graph is then given by $p_{12} = 1 - (1-p_1)(1-p_2)$. A similar interpretation to proposition 3 can be given for graphs with transition probabilities by replacing the cardinality of the sets of walks with the sum over the probabilities of the walks.

6.3 Non-contiguous Label Sequences

The (implicit) representation of a graph by a set of walks through the graph suggests a strong relation to string kernels investigated in literature [14]. There, the similarity of two strings is based on the number of common substrings. In contrast to the direct product kernel suggested in section 5, however, the substrings need not be contiguous.

In this section we will describe a graph kernel such that the similarity of two graphs is based on common non-contiguous label sequences. We will consider only edge-labeled graphs in this section. A similar technique can be used for fully-labeled graphs, however, its presentation becomes more lengthy.

Before defining the desired feature space we need to introduce the wildcard symbol '?' and the function $match(l, l') \Leftrightarrow (l = l') \vee (l =?) \vee (l' =?)$. In the following 'label' will refer to an element of the set $\mathcal{L} \cup \{?\}$.

Definition 7. *Let $\mathcal{S}_{n,m}$ denote the set of all possible label sequences of length n containing $m \geq 0$ wildcards. Let λ be a sequence $\lambda_0, \lambda_1, \ldots$ of weights ($\lambda_n \in \mathbb{R}; \lambda_n \geq 0$ for all $n \in \mathbb{N}$) and let $0 \leq \alpha \leq 1$ be a parameter for penalizing gaps. Furthermore, let $\mathcal{W}_n(G)$ denote the set of all possible walks with n edges in graph G and let $\mathcal{W}(G) = \bigcup_{i=0}^{n} \mathcal{W}_n(G)$. For a given walk $w \in \mathcal{W}(G)$ let $l_i(w)$ denote the label of the i-th edge in this walk.*

The sequence feature space is defined by one feature for each possible label sequence. In particular, for any given n, m and label sequence $s = s_1, \ldots, s_n \in \mathcal{S}_{n,m}$, the corresponding feature value is

$$\phi_s(G) = \sqrt{\lambda_n \alpha^m} \, |\{w \in \mathcal{W}(G), \forall i : \ match(s_i, l_i(w))\}|$$

We proceed directly with the definition of the non-contiguous sequence kernel.

Definition 8. *Let G_1, G_2 be two graphs, let $G_\times = G_1 \times G_2$ be their direct product, and let G_o be their direct product when ignoring the labels in G_1 and G_2. With a sequence of weights $\lambda = \lambda_0, \lambda_1, \ldots$ ($\lambda_i \in \mathbb{R}; \lambda_i \geq 0$ for all $i \in \mathbb{N}$) and a factor $0 \leq \alpha \leq 1$ penalizing gaps, the non-contiguous sequence kernel is defined as*

$$k_*(G_1, G_2) = \sum_{i,j=1}^{|\mathcal{V}_\times|} \left[\sum_{n=0}^{\infty} \lambda_n \left((1-\alpha) E_\times + \alpha E_o \right)^n \right]_{ij}$$

if the limit exists.

This kernel is very similar to the direct product kernel. The only difference is that instead of the adjacency matrix of the direct product graph, the matrix $(1-\alpha)E_\times + \alpha E_o$ is used. The relationship can be seen by adding – parallel to each edge – a new edge labeled # with weight $\sqrt{\alpha}$ in both factor graphs.

Note, that the above defined feature space contains features for 'trivial label sequences', i.e., label sequences that consist only of wildcard symbols. This can be corrected by using the kernel $k_*(G_1, G_2) - \sum_n \lambda_n \alpha^n E_o^n$ instead.

7 Discussion and Related Work

This section briefly describes the work most relevant to this paper. For an extensive overview of kernels on structured data, the reader is referred to [5].

Graph kernels are an important means to extend the applicability of kernel methods to structured data. Diffusion kernels on graphs [12] allow for the computation of a kernel if the instance space has a graph structure. This is different from the setting in this paper where we consider that every instance has a graph structure.

A preliminary version of the label pair graph kernel has been presented in [4]. The feature space of this kernel is based on the distance of all labeled vertices in graphs. The dimensionality of the feature space is the number of different labels squared. In applications, where this is not sufficiently discriminative, inner products in a feature space where each feature counts the number of times a path with a given label sequence is homomorphic to a subgraph can be used. Such kernels can be extended to cover graphs with a transition probability associated to each edge.

A kernel function on transition graphs has previously been described in [9][3]. In the feature space considered there, each feature corresponds to the probability with which a label sequence is generated by a random walk on the direct product graph. There transition graphs with a uniform distribution over all edges leaving the same vertex are considered and convergence is guaranteed by assuming a non-zero halting probability. In section 6 we showed how our graph kernels can be extended to cover general transition graphs. The kernel proposed in [9] is a special case of this extended kernel.

Tree kernels have been described in [2] and compute the similarity of two trees based on their common subtrees. However, the trees considered there are restricted to labeled trees where the label of each vertex determines the number of children of that vertex. Furthermore, a fixed order of the children of each vertex is assumed. In this case [2] devises an algorithm for computing the kernel with quadratic time complexity. The graph kernels suggested in the paper at hand can clearly also be applied to trees but do not require any restriction on the set of trees. They are thus a more flexible alternative.

String kernels that base the similarity of two strings on the common non-contiguous substrings are strongly related to the non-contiguous graph kernel with geometric weights. Such string kernels have been investigated deeply in literature, see for example [17, 14]. However, using a similar idea on graphs – as done with the non-contiguous graph kernel – is new.

The main differences between string kernels and graph kernels for strings are which features are used and how the label sequences are matched. String kernels use a label sequence without wildcards and count how often this sequence or a sequence created by inserting wildcard symbols occurs. Graph kernels for strings use a label sequence with wildcards and count how often this sequence occurs. Consider, for example, the two strings ‘art’ and ‘ant’. For string kernels the common substrings are ‘a’, ‘t’, and ‘at’ – ‘at’ occurs with a gap of length 1 in both strings. For direct product graph kernels the common sequences are only ‘a’ and ‘t’. For non-contiguous graph kernels the common sequences are ‘a’, ‘t’, ‘a?’, ‘?t’, ‘a??’, ‘??t’, and ‘a?t’. The most interesting difference between string kernels

[3] An extended version of this paper appeared recently [10].

and non-contiguous graph kernels is the feature 'at' and 'a?t', respectively. Now consider the string 'at'. The sequence 'a?t' does not occur in 'at', while the substring 'at' obviously occurs in 'at'.

8 Conclusions and Future Work

In this paper we showed that computing a complete graph kernel is at least as hard as deciding whether two graphs are isomorphic, and that the problem of computing a graph kernel based on common (isomorphic) subgraphs is *NP*-hard. Therefore, we presented alternative graph kernels that are conceptually based on the label sequences of all possible walks in the kernel. Efficient computation of these kernels is made possible by the use of product graphs and by choosing the weights such that a closed form of the resulting matrix power series exists.

The advantage of the label pair graph kernel is that the feature space can be computed explicitly and thus linear optimization methods for support vector machines can be applied. Therefore, learning with this kernel can be very efficient. The advantage of the direct product kernel and the non-contiguous graph kernel is the expressivity of their feature spaces. Both definitions are based on the concept of graph products.

We have shown that the direct graph product can be employed to count the number of contiguous label sequences occurring in two graphs and that it can be to extended to count the number of non-contiguous label sequences occurring in two graphs.

We believe that such graph kernels can successfully be applied in many real-world applications. Such applications will be investigated in future work. One step towards real world applications is an experiment in a relational reinforcement learning setting, described in [6]. There we applied Gaussian processes with graph kernels as the covariance function. Experiments were performed in blocks worlds with up to ten blocks with three different goals. In this setting our algorithm proofed competitive or superior to all previous implementations of relational reinforcement learning algorithms, although it did not use any sophisticated instance selection strategies.

Real world experiments will apply graph kernels to different molecule classification tasks from bioinformatics and computational chemistry.

Acknowledgments

Research supported in part by the EU Framework V project (IST-1999-11495) *Data Mining and Decision Support for Business Competitiveness: Solomon Virtual Enterprise* and by the DFG project (WR 40/2-1) *Hybride Methoden und Systemarchitekturen für heterogene Informationsräume*. The authors thank Tamás Horváth and Jan Ramon for valuable discussions.

References

1. B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.
2. M. Collins and N. Duffy. Convolution kernels for natural language. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, Cambridge, MA, 2002. MIT Press.
3. R. Diestel. *Graph Theory*. Springer-Verlag, 2000.
4. T. Gärtner. Exponential and geometric kernels for graphs. In *NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*, 2002.
5. T. Gärtner. Kernel-based multi-relational data mining. *SIGKDD Explorations*, 2003. to appear.
6. T. Gärtner, K. Driessens, and J. Ramon. Graph kernels and gaussian processes for relational reinforcement learning. In *Proceedings of the 13th International Conference on Inductive Logic Programming*, 2003. submitted.
7. R. Gray. *Probability, Random Processes, and Ergodic Properties*. Springer-Verlag, 1987.
8. W. Imrich and S. Klavžar. *Product Graphs: Structure and Recognition*. John Wiley, 2000.
9. H. Kashima and A. Inokuchi. Kernels for graph classification. In *ICDM Workshop on Active Mining*, 2002.
10. H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th International Conference on Machine Learning*, 2003. to appear.
11. J. Köbler, U. Schöning, and J. Toràn. *The Graph Isomorphism Problem: Its Structural Complexity*. Progress in Theoretical Computer Science. Birkhäuser, 1993.
12. R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In C. Sammut and A. Hoffmann, editors, *Proceedings of the 19th International Conference on Machine Learning*, pages 315–322. Morgan Kaufmann, 2002.
13. B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, 2002.
14. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.
15. B. Schölkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, 2002.
16. S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, 1997.
17. C. Watkins. Kernels from matching operations. Technical report, Department of Computer Science, Royal Holloway, University of London, 1999.