Expressivity versus Efficiency of Graph Kernels

Jan Ramon[1] and Thomas Gärtner[2,3]

[1] Department of Computer Science, K.U.Leuven, Belgium
[2] Fraunhofer Institut Autonome Intelligente Systeme, Germany
[3] Department of Computer Science III, University of Bonn, Germany
jan.ramon@cs.kuleuven.ac.be
Thomas.Gaertner@ais.fraunhofer.de

Abstract. Recently, kernel methods have become a popular tool for machine learning and data mining. 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. In this paper we study the trade-off between expressivity and efficiency of graph kernels. First, we motivate the need for this discussion by showing that fully general graph kernels can not even be approximated efficiently. We also discuss generalizations of graph kernels defined in literature and show that they are either not positive definite or not very useful. Finally, we propose a new graph kernel based on subtree patterns. We argue that while a little more computationally expensive, this kernel is more expressive than kernels based on walks.

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 [10]. 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 complexity of such kernels, one line of existing research has concentrated on special kinds of graphs, in particular, trees [2] or strings [11, 9] which results in efficient kernels, but loses most of the power of general graphs. Others [4, 7] 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.

More recently [5] answered the questions whether it is possible to define kernels that take the entire structure of graphs into account. While those kernels can be defined, computing them is hard. For that reason, alternative graph kernels based on common walks are investigated that, for example, allow for gaps in the label sequences corresponding to the walks.

In this paper we show several other results concerning graph kernels. Complete graph kernels are those graph kernels that distinguish between two graphs if and only if they are not isomorphic. It is known that computing complete graph kernels is at least as hard as deciding whether two graphs are isomorphic. In this paper we will show that even approximating k with a constant bound on the approximation-error is as hard as deciding whether two graphs are isomorphic. Also we review the basic ideas of previous work on graph kernels and show that these ideas can not directly be generalized to more expressive graph kernels. We show this by demonstrating that straight-forward generalizations do either lead to non positive definite graph kernels or to trivial feature spaces. These results motivate the search for other, more expressive graph kernels.

The remainder of this paper is structured as follows: In Section 2 we review some basic definitions. In Section 3 we show that complete graph kernels can not be efficient. In Section 4 we discuss a general framework for graph kernels based on common subgraphs, and discuss consequences of generalizing previously studied graph kernels. In Section 5 we propose a new instantiation of this framework, a graph kernel that uses the count of subtrees as features. Finally, in Section 6 we give conclusions and directions for further work.

2 Graphs

We first review a few basic definitions and introduce some notations that will be used in the sequel of this paper. For a more in-depth discussion of graphs and related concepts the reader is referred to [3, 8].

Labeled directed graphs Generally, a *graph* G is described by a finite set of *vertices* $\mathcal{V}$, a finite set of *edges* $\mathcal{E}$. Therefore, a graph is commonly denoted $G(\mathcal{V}, \mathcal{E})$. For *labeled* graphs there is additionally a set of labels $\mathcal{L}$ along with a function *label* assigning a label to each edge and vertex. For unlabeled graphs, we will assume $label(v)$ has the same value for all vertices. We will denote the space of all graphs with $\mathcal{G}$. We will sometimes assume some enumeration of the vertices and labels in a graph, i.e., $\mathcal{V} = \{\nu_i\}_{i=1}^{n}$ where $n = |\mathcal{V}|$ and $\mathcal{L} = \{\ell_r\}_{r\in\mathbb{N}}$ [1]. For *undirected* graphs, each edge is a set containing two vertices. For *directed* graphs without parallel edges each edge is a tuple consisting of the initial and terminal vertex of the edge $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Edges (v, v) in a directed graph are called *loops.*

[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.

Some special graphs, relevant for the description of graph kernels are walks, paths, cycles, trees and forests. A *walk*[2] w of a graph $G(\mathcal{V}, \mathcal{E})$ 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 graph is *acyclic* if no subgraph of a graph is a cycle. A *subforest* is an acyclic subgraph; a *subtree* is an connected subforest. We denote the set of all graphs by $\mathcal{G}$.

A graph is isomorphic to another graph if there is an edge (and label) preserving bijection between all vertices in one graphs and all vertices in the other graph. A graph is homomorphic to another graph if there is an edge (and label) preserving surjection between all vertices in one graphs and all vertices in the other graph.

We also need to define some functions describing the neighborhood of a vertex v in a graph $G(\mathcal{V}, \mathcal{E})$: $\delta^+(v) = \{u : (v, u) \in \mathcal{E}\}$ and $\delta^-(v) = \{u : (u, v) \in \mathcal{E}\}$. Here, $|\delta^+(v)|$ is called the *outdegree* of a vertex and $|\delta^-(v)|$ the *indegree*.

Product Graphs Product graphs [6] 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. Here is the definition of the direct product graph of two labeled graphs as given in [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:

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

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. The graphs G_1, G_2 are called the factors of $G_1 \times G_2$.

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

3 Complete Graph Kernels

Given some set, there are many possible ways to define positive definite kernels on it. The optimal choice not only depends on the structure of the data but also on the concept class and the data itself. The concept class is the class of all relevant concepts. The concept that is used to determine the class of examples belongs to the concept class but is unknown to the learning system. There exists a wide range of learning theory results putting bounds on the number of examples needed to learn certain concept classes assuming that a kernel evaluation can be performed in unit time. However, for graphs this is not evident. In this section, we show this by proving that kernels that approximate kernels that map graphs on separated points in feature space, are necessarily hard to compute. First, we need to define positive definite kernel, complete graph kernel, and distance induced by a kernel. A thorough discussion on kernels and kernel based learning is given in [10].

Positive Definite Kernel Let $\mathcal{X}$ be a set. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *positive definite kernel* on $\mathcal{X}$ if, for all $n \in \mathbb{Z}^+$, $x_1, \ldots, x_n \in \mathcal{X}$, and $c_1, \ldots, c_n \in \mathbb{R}$, it follows that $\sum_{i,j \in \{1,\ldots,n\}} c_i \, c_j \, k(x_i, x_j) \geq 0$.

Complete Kernel 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 , k is called a *complete* graph kernel.

Induced Distance Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive definite kernel on any set $\mathcal{X}$ and let $x, y \in \mathcal{X}$. Then, $d_k(x, y) = \sqrt{k(x,x) - 2k(x,y) + k(y,y)}$ is the distance induced by k. Note that for the map $\phi : \mathcal{X} \to \mathcal{H}$ such that $\langle \phi(x), \phi(y) \rangle = k(x, y)$ for all x, y, it holds that $d_k(x, y) = \|\phi(x) - \phi(y)\|$. If k is complete, d_k is a distance, otherwise d_k is a pseudo-distance.

A proposition similar to the following result has been presented in [5]

Proposition 1. *Let $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ be a complete graph kernel. If there is an algorithm such that the time needed to compute $k(G_1, G_2)$ for any graphs $G_1(\mathcal{V}_1, \mathcal{E}_1), G_2(\mathcal{V}_2, \mathcal{E}_2) \in \mathcal{G}$ is bounded by a function $f(|\mathcal{V}_1|, |\mathcal{E}_1|, |\mathcal{V}_2|, |\mathcal{E}_2|)$, then there is an algorithm that decides whether two graphs G_1 and G_2 are isomorphic and runs in time $\mathcal{O}(f(|\mathcal{V}_1|, |\mathcal{E}_1|, |\mathcal{V}_2|, |\mathcal{E}_2|))$.*

Proof. Let ϕ be the map corresponding to the complete kernel k. 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, G' are isomorphic. Hence, to decide whether G_1 and G_2 are isomorphic, it is sufficient to evaluate the three kernel expressions $k(G_1, G_1)$, $k(G_1, G_2)$ and $k(G_2, G_2)$ and do a simple addition, which can clearly be done in time $\mathcal{O}(f(|\mathcal{V}_1|, |\mathcal{E}_1|, |\mathcal{V}_2|, |\mathcal{E}_2|))$ □

As deciding graph isomorphism is known to be hard and the previous theorem shows that evaluating a complete kernel is equally hard, one can conclude that no efficiently computable complete kernel on $\mathcal{G}$ exists. We can even prove a stronger result: it is impossible to define an efficiently computable function that approximates closely a complete kernel.

Proposition 2. *Let $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ be a complete graph kernel such that for any two non-isomorphic graphs G_1 and G_2, $d_k(G_1, G_2) \geq d$. Let k' be a function that approximates k such that $\forall x, y : |k(x, y) - k'(x, y)| < d/8$. Then, if there is an algorithm such that the time needed to compute $K(G_1, G_2)$ for any graphs $G_1, G_2 \in \mathcal{G}$ is bounded by a function $f(|\mathcal{V}_1|, |\mathcal{E}_1|, |\mathcal{V}_2|, |\mathcal{E}_2|)$, then there is an algorithm that decides whether two graphs G_1 and G_2 are isomorphic and runs in time $\mathcal{O}(f(|\mathcal{V}_1|, |\mathcal{E}_1|, |\mathcal{V}_2|, |\mathcal{E}_2|))$.*

Proof. Assume that such a k' exists. Then, we could for any two graphs G_1 and G_2 compute $d_{k'}(G_1, G_2)$ in time $\mathcal{O}(f(|\mathcal{V}_1|, |\mathcal{E}_1|, |\mathcal{V}_2|, |\mathcal{E}_2|))$. We then have

$$\begin{aligned} & d_{k'}(G_1, G_2) - d_k(G_1, G_2) \\ &= \big(k'(x,x) - 2k'(x,y) + k'(y,y)\big) - \big(k(x,x) - 2k(x,y) + k(y,y)\big) \\ &< d/2. \end{aligned}$$

Therefore, it is sufficient to compare $d_{k'}(G_1, G_2)$ to $d/2$ to decide whether the graphs are isomorphic or not: if $d_{k'}(G_1, G_2)$ is smaller than $d/2$, $d_k(G_1, G_2)$ can not be larger than d and the graph are isomorphic while if it is larger than $d/2$, the distance induced by k between the graphs can not be zero. □

This theorem says in fact that there is no efficiently computable function that approximates a complete kernel sufficiently well to be able to distinguish between near (in feature space) but non-isomorphic graphs. Apart from the hardness of graph isomorphism, one can apply other computational complexity results on graphs to kernels. E.g. one can not expect to be able to define graphs kernels to learn efficiently concepts which are known to be hard to compute (such as containing subgraphs).

4 Graph Kernels based on Common Subgraphs

In this section we first briefly introduce the general idea behind previous approaches to define efficient graph kernels. Then we present you results indicating that straight-forward generalizations of these ideas do not result in positive definite functions. This motivates our search for other graph kernels, presented in the next section.

We will consider only those subgraphs that do not contain isolated vertices. To be able to give different weights to different sizes of subgraphs we assume a sequence $\lambda_0, \lambda_1, \ldots$ of weights with $\lambda_i > 0$.

4.1 General Approach

The general idea of graph kernels defined so far is to measure common subgraphs of two graphs. Conceptually, the feature map ϕ of the graph kernel k with $k(G, G') = \langle \phi(G), \phi(G') \rangle$ has one feature ϕ_h for every graph h from some given set H.

The graph kernel mentioned in the previous section can then be described by:

$$\Phi_h(G) = \sqrt{\lambda_{|\mathcal{E}(h)|}} \Big| \{g \text{ is subgraph of } G : h \text{ is isomorphic to } g\} \Big| \tag{1}$$

As computing this kernel is *NP*-hard, alternative kernels are based on homomorphism

$$\phi_h(G) = \sqrt{\lambda_{|\mathcal{E}(h)|}} \Big| \{g \text{ is subgraph of } G : h \text{ is homomorphic to } g\} \Big| \tag{2}$$

4.2 Common Walks

In [4, 7, 5] kernels are considered where H is the set of paths. The kernel between two graphs can then efficiently be computed as a matrix power series of the adjacency matrix of their direct product graph. This principle can be formulated as follows:

$$f_H(G, G') = \sum_{h \in H} \lambda_{|\mathcal{E}(h)|} \Big| \{g \text{ is subgraph of } G \times G' : h \simeq g\} \Big| \tag{3}$$

To be precise, not all kernels considered in [4, 7, 5] fit directly in this framework. This is because for some kernels the cardinality of the set of walks is replaced by the probability of observing a random walk with a label sequence corresponding to this walk; for other kernels only some particular labels along the walks are compared and not all. Both modifications are, however, not conceptually different from this framework.

The computation of f_H does not necessarily require to check graph isomorphism, as the summation can be over all subgraphs of $G \times G'$ that satisfy some property. That property must be the characteristic property of H. When counting common walks (H is the set of walks) f_H is a positive definite kernel function corresponding to the inner product under the map $\phi_h, h \in H$ [5]. For appropriate weight sequences f_H can be computed in cubic time.

4.3 Common Paths, Trees, and Forests

In this section we consider functions f_H where H is some set of acyclic graphs, either all paths, all trees, or all forests. We call these functions a subpath, subtree, and subforest function, respectively.

Proposition 3. *The subpath, subtree, and subforest functions are not positive definite.*

Proof. Let G be a single loop and let G' be a graph with two vertices and a single edge connecting these vertices. Then $G \times G$ and G are isomorphic, as well as $G' \times G'$, $G \times G'$, and G'. In G the only path is the trivial path consisting of just one vertex. G' has one edge, so it has three paths: two trivial ones consisting of one vertex and one consisting of the two vertices and the edge connecting them. Hence,

$$k(G,G) - 2k(G,G') + k(G',G') = \lambda_0 - 2(2\lambda_0 + \lambda_1) + 2\lambda_0 + \lambda_1 = -\lambda_0 - \lambda_1 < 0$$

$\square$

4.4 Common Graphs

In this section we consider the function f_H where H is the set of all graphs and μ is the set cardinality. We call this function the subgraph function.

Proposition 4. *The subgraph function is positive definite.*

Proof. Let G, G' be any two graphs. The number of subgraphs of their product graph is $2^{|\mathcal{E}(G \times G')|}$. The number of edges of a given label in the product graph is simply the product of the number of edges of that label in each factor. As kernels are closed under products and as $a^x, a \geq 1$ can be written as the limit of a polynomial series with positive coefficients, the subgraph function is positive definite. $\square$

Although the subgraph function is positive definite, it is not likely to be useful in practice, as it does not take the structure of the graphs into account

5 Tree-structured Pattern Kernels

In section 3, we argued that generally usable graph kernels can not even be approximated efficiently. All graph kernels investigated in literature so far are conceptually based on some measure of the walks in two graphs that have some or all labels in common. Computation of these kernels is made possible by using the direct product graph and computing a closed form of the limit of a matrix power series of the adjacency matrix of the product graph. For such kernels, one can easily find pairs of graphs which are mapped to the same point in feature space, e.g. Figure 1 shows two directed graphs and Figure 2 shows two undirected graphs whose images in the Hilbert space are the same. Such graphs can not be distinguished by any kernel machine using a walk-based graph kernel.

In section 4 we then showed that functions based on counting paths, trees, and forests in the product graph are not positive definite. In this section we will push the limit of efficiently computable graph kernels by describing a method for counting the number of common *subtree patterns* in two graphs. Roughly, the subtree patterns we consider are rooted subgraphs such that there is a tree homomorphic to the subgraph, and the number of distinct children of both root nodes in the pattern and tree are the same.

Formally, let $G(\mathcal{V},\mathcal{E}) \in \mathcal{G}$ be a graph. If $r \in \mathcal{V}$, then r is a *subtree pattern* of G *rooted* at r. If $t_1, t_2, \ldots, t_n$ are subtree patterns of G rooted at respectively $r_1, r_2, \ldots, r_n$ (with all r_i different), and if $(r,r_1),(r,r_2),\ldots(r,r_n) \in \mathcal{E}$, then $r(t_1, t_2, \ldots, t_n)$ is a subtree pattern of G rooted at r. r is also called the parent node of the nodes r_i of the nodes r_i of the subtree pattern.

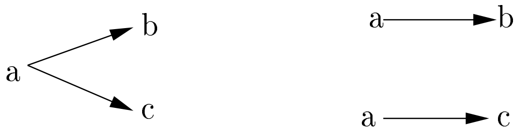

Fig. 1. Directed graphs mapped to the same point in walks feature space

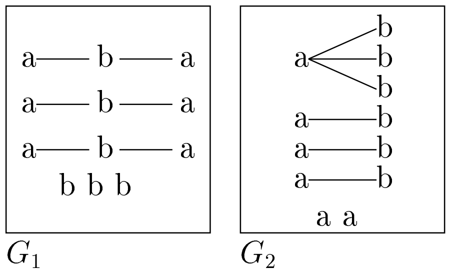

Fig. 2. Undirected graphs mapped to the same point in walks feature space

Every subtree pattern has a tree-structured signature just as every walk had a signature represented by the sequence of labels of the vertices in the sequence. So for each possible subtree pattern signature, we associate a feature whose value is the number of times that a subtree of that signature occurs in the graph.

Let $G_1(\mathcal{V}_1,\mathcal{E}_1), G_2(\mathcal{V}_2,\mathcal{E}_2) \in \mathcal{G}$ be two graphs. We will denote the weighted count of pairs of subtrees of the same signature of height less than or equal to h, with the first one rooted at $r \in \mathcal{V}(G_1)$ and the second one rooted at $s \in \mathcal{V}(G_2)$ with $k_{r,s,h}$. Now, if $h = 1$ and $label(s) = label(r)$ we have $k_{r,s,h} = 1$. If $h = 1$ and $label(s) \neq label(r)$ we have $k_{r,s,h} = 0$. For $h > 1$, one can compute $k_{r,s,h}$ as follows:

– Let $M_{r,s}$ be the set of all matchings from $\delta^+(r)$ to $\delta^+(s)$, i.e.

$$M_{r,s} = \Big\{ R \subseteq \delta^+(r) \times \delta^+(s) \mid \big(\forall (a,b),(c,d) \in R : a = c \Leftrightarrow b = d\big) \\ \wedge \big(\forall (a,b) \in R : label(a) = label(b)\big) \Big\}$$

– Compute

$$k_{r,s,h} = \lambda_r \lambda_s \sum_{R \in M_{r,s}} \prod_{(r',s') \in R} k_{r',s',h-1}$$

Here, λ_r and λ_s are positive values smaller than 1 to cause higher trees to have a smaller weight in the overall sum.

Given two graphs $G_1(\mathcal{V}_1, \mathcal{E}_1), G_2(\mathcal{V}_2, \mathcal{E}_2) \in \mathcal{G}$, we can then define the subtree-pattern kernel of G_1 and G_2 by

$$k_{tree,h}(G_1, G_2) = \sum_{r \in V_1} \sum_{s \in V_2} k_{r,s,h}.$$

Proposition 5. *$k_{tree,h}$ is a positive definite kernel.*

Proof. The feature space induced by this kernel contains one feature for each subtree pattern signature. The value of such a feature is the sum of the weights of all its occurrences in the graph. An occurrence of a subtree pattern $r(t_1, \ldots, t_n)$ has weight $weight(r(t_1, \ldots, t_n)) = \lambda_r \sum_i weight(t_i)$ and the weight of a trivial subtree pattern r is 1. We have defined the feature space explicitely, $k_{tree,h}$ is now equal to the inner product in this space and hence $k_{tree,h}$ is positive definite. □

One can also define

$$k_{tree}(G_1, G_2) = \lim_{h \to \infty} k_{tree,h}(G_1, G_2).$$

For suitable λ_r, λ_s (i.e. causing the sum to remain bounded for increasing values of h) this limit will converge. Since $k_{tree,h}$ is already a kernel for every h, in practice $k_{tree,h}$ can be used and for sufficiently large h it will be a good approximation of k_{tree}.

The graphs in Figure 2 are not equal for this kernel k_{tree}. The feature space of k_{tree} includes the feature space with all walks, but also includes features such as the subtree pattern $a(b, b, b)$ which has value 0 for G_1 and value 1 for G_2. The computational complexity is higher than the complexity for kernels based on walks. This is mainly due to the summation over $M_{r,s}$. However, if the graph is not too connected or if there is sufficient diversity in the labels of the vertices, this extra cost will be fairly low.

6 Conclusions and Further Work

In this paper we discussed several aspects of graph kernels and the trade-off between efficiency and expressivity. We improved a result that shows that no efficiently complete graph kernels exist. We presented a framework for kernels based on common subgraphs and argued that many existing kernels fit into this framework. We argued that kernels using linear patterns to define feature will not suffice in all practical cases and proposed a tree-pattern based graph kernel which has a strictly larger feature space. We also pointed out that some kernels that count weighted patterns which do not allow vertices to repeat (walks, subtrees and forests) in the product graph are not positive definite and often hard to compute and that using all subgraphs (without weighting) results in a kernel which is not very useful.

There are several possible directions for further work. First, there exists much work on learnability results for related representation languages such as inductive logic programming. Adapting these results to use graphs as representations could give more insight on efficiently learnable classes of graph concepts and in particular provide hints at classes for which efficient kernels could exist. Second, as many different kernels on graphs are proposed, it would be useful to compare them in more detail. One possible direction is to empirically evaluate on which kind of applications which kernel performs better.

Acknowledgments Jan Ramon is a post-doctoral fellow of the Katholieke Universiteit Leuven. 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 Peter Flach, Tamás Horváth, Stefan Kramer and Stefan Wrobel.

References

1. B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In David 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, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop*, 2003. to appear.
6. W. Imrich and S. Klavžar. *Product Graphs: Structure and Recognition*. John Wiley, 2000.
7. H. Kashima and A. Inokuchi. Kernels for graph classification. In *ICDM Workshop on Active Mining*, 2002.
8. B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, 2002.
9. 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.
10. B. Schölkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, 2002.
11. C. Watkins. Kernels from matching operations. Technical report, Department of Computer Science, Royal Holloway, University of London, 1999.