Faster Computation of the Direct Product Kernel for Graph Classification

Nikhil S. Ketkar, Lawrence B.Holder, Diane J. Cook

Abstract— **The direct product kernel, introduced by Gärtner et al. for graph classification, is based on defining a feature for every possible label sequence in a labelled graph and counting how many label sequences in two given graphs are identical. Although the direct product kernel has achieved promising results in terms of accuracy, the kernel computation is not feasible for large graphs. This is because computing the direct product kernel for two graphs is essentially computing either the inverse of or by diagonalizing the adjacency matrix of the direct product of these two graphs. For two graphs with adjacency matrices of sizes m and n, the adjacency matrix of their direct product graph can be of size mn in the worst case. As both matrix inversion or matrix diagonalizing in the general case is $O(n^3)$, computing the direct product kernel is $O((mn)^3)$. Our survey of data sets in graph classification indicates that most graphs have adjacency matrices of sizes in the order of hundreds which often leads to adjacency matrices of direct product graphs (of two graphs) having sizes in the order of thousands.**

In this work we show how the direct product kernel can be computed in $O((m+n)^3)$. The key insight behind our result is that the language of label sequences in a labeled graph is a regular language and that regular languages are closed under union and intersection.

I. Overview

In the past, machine learning research has focused on attribute-valued data or data that is naturally expressed as a single table. Although these methods have achieved great success in a variety of real world domains, data in a majority of domains has an inherent structure which prevents it from being expressed as attribute-valued data and hence new approaches for dealing with such data have been developed. The two main approaches for dealing with such data are based on representing such data in first-order logic and graphs. A variety of problems based on graphical representation of structured data have been studied in the past. The problem that this work focuses on is that of graph classification which is learning to classify separate, individual graphs in a graph database into two or more categories.

The problem of graph classification was first studied by Gonzales et al. [1] who proposed the SubdueCL algorithm for the task and had promising initial results. The approach was based on a greedy search for sub-graphs which distinguish one class of graphs from all other classes. Since then, a variety of new algorithms and approaches based on extending existing attribute-valued algorithms have been studied. Deshpandey et al. [2] applied the FSG system to mine frequent sub-graphs in a graph database which were represented as a feature vector, and support vector machines were then applied to classify these feature vectors. Nguyen et al. [3] proposed the DT-CLGBI algorithm which learns a decision tree for graph classification in which each node is associated with a sub-graph and represents an existence/nonexistence test. Kudo et al. [4] proposed an approach based on boosting decision stumps where a decision stump is associated with a sub-graph and represents an existence/nonexistence test. Gärtner et al. [5] proposed an approach based on using support vector machines for the task of graph classification by developing graph kernels. Two kernels, namely, the walk-based (direct product) kernel and the cycle-based graph kernel were proposed in this work. Kashima et al. [6] recently proposed another walk-based graph kernel.

In this work we focus on the direct product kernel proposed by Gärtner et al. [5]. The direct product kernel takes as input two graphs and returns an inner product of the vectors of label sequences corresponding to walks in the two input graphs. Intuitively, this can be seen as a count of the identical walks that can be taken in both of the graphs. Computing the direct product kernel for two graphs is essentially computing either the inverse of or by diagonalizing the adjacency matrix of the direct product of these two graphs.

While this kernel has achieved promising results in terms of accuracy, the kernel computation is not feasible for large graphs. Our survey of data sets in graph classification indicates that most graphs have adjacency matrices of sizes in the order of hundreds which often leads to adjacency matrices of direct product graphs (of two graphs) having sizes in the order of thousands. For two graphs with adjacency matrices of sizes m and n, the adjacency matrix of their direct product graph can be of size mn in the worst case. As both matrix inversion or matrix diagonalizing in the general case is $O(n^3)$, computing the direct product kernel is $O((mn)^3)$.

Our contribution in this work is that we show how the direct product kernel can be computed in $O((m+n)^3)$. The key insight behind our result is that the language of label sequences in a labeled graph is a regular language and that regular languages are closed under union and intersection.

The rest of the paper is organized as follows. First, we present a formulation of the graph classification problem and survey the various approaches to address the problem. The next section deals with the alternative feasible kernel computation. Next, the current method for this kernel computation and the lack of its feasibility is discussed by surveying various graph data sets.

II. Graph Classification: Problem and Approaches

First, we formulate the graph classification problem. A directed edge-labeled graph is a triple $g = (V, E, \alpha)$ where

978-1-4244-2765-9/09/$25.00 ©2009 IEEE

V is the set of vertices, $E \subseteq V \times V$ is a set of directed edges and α is the edge labelling function $\alpha : E \rightarrow \Sigma$ where Σ is the alphabet of edge labels. Note that, although our formulation and results are based on directed, edge label graphs for simplicity, they are trivially extended to undirected, mixed (directed and undirected edges), vertex labelled and various other types of graphs. Given a set of training examples $T = \{\langle x_i, y_i \rangle\}_{i=0}^{L}$ where $x_i \in \mathcal{X}$ is a vertex labeled graph and $y_i \in \{+1, -1\}$, the graph classification problem is to induce a mapping $f : \mathcal{X} \rightarrow \{+1, -1\}$.

A. SUBDUE

The SubdueCL algorithm proposed by [7] is the pioneering algorithm for the graph classification problem. The key aspect of the algorithm is the greedy, heuristic search for subgraphs present in positive examples and absent in the negative examples. The hypothesis space of Subdue consists of all the connected subgraphs of all the example graphs labeled positive.

Subdue performs a beam search which begins from subgraphs consisting of all vertices with unique labels. The subgraphs are extended by one vertex and one edge or one edge in all possible ways, as guided by the input graphs, to generate candidate subgraphs. Subdue maintains the instances of subgraphs (in order to avoid subgraph isomorphism) and uses graph isomorphism to determine the instances of the candidate substructure in the input graph. Candidate substructures are evaluated according to classification accuracy or the minimum description length principle introduced by [8].

The length of the search beam determines the number of candidate substructures retained for further expansion. This procedure repeats until all substructures are considered or the user imposed computational constraints are exceeded. At the end of this procedure the positive examples covered by the best substructure are removed. The process of finding substructures and removing positive examples continues until all the positive examples are covered.

The model learned by Subdue thus consists of a decision list each member of which is connected graph. Applying this model to classify unseen examples involves conducting a subgraph isomorphism test; if any of the graphs in the decision list are present in the example, it is predicted as positive, if all the graphs in the decision list are absent it the example, it is predicted as negative.

Two additional features of the algorithm are inexact graph matching and incorporation of background knowledge. Subdue can perform an inexact graph isomorphism in the candidate evaluation stage. Here, two candidate subgraphs are evaluated to be identical if they differ by a value which falls below the user specified threshold. The intuition behind this feature is to make the algorithm more robust to noise in the examples. The incorporation of background knowledge involves providing SubdueCL with predefined substructures as background knowledge. Subdue uses this background knowledge by preprocessing the examples and compressing each of the user defined substructures which form the background knowledge into a single vertex. The intuition behind this feature is to efficiently search the space of subgraphs by utilising the information provided by a domain expert about important structural features.

B. Frequent Subgraph Mining in Conjunction with SVMs

The approach introduced by [9] for graph classification involves the combination of work done in two diverse fields of study, namely, frequent subgraph mining and support vector machines. First, we present a quick overview of the work done on the frequent subgraph mining problem.

The frequent subgraph mining problem is to produce the set of subgraphs occurring in at least ϵ of the given n input example graphs (which are referred to as transactions). The initial work in this area was the AGM system proposed by [10] which uses the apriori level-wise search approach. The FSG system proposed by [11] takes a similar approach and further optimizes the algorithm for improved running times. The gSpan system proposed by [12] uses DFS codes for canonical labeling and is much more memory and computationally efficient than the previous approaches. The most recent work on this problem is the Gaston system proposed by [13] which efficiently mines graph datasets by first considering frequent paths which are transformed to trees which are further transformed to graphs. Contrasting to all these approaches to frequent subgraph mining which are complete, the systems Subdue by [14] and GBI by [15] are based on heuristic, greedy search.

The key idea in combining frequent subgraph miners and SVMs in order to perform graph classification is to use a frequent subgraph mining system to identify frequent subgraphs in the given examples, then to construct feature vectors for each example where each feature is the presence or absence of a particular subgraph and train a support vector machine to classify these feature vectors.

The model produced by this approach thus consists of a list of graphs and a model produced by the SVM. Applying this model to classify unseen examples involves conducting a subgraph isomorphism test; a feature vector for the unseen example is produced wherein each feature represents the presence or absence of a graph in the list in the unseen example and this feature vector is classified as positive or negative by the model produced by the SVM.

C. Frequent Subgraph Mining in Conjunction with AdaBoost

The approach proposed by [16] involves combining aspects of frequent subgraph miners and AdaBoost in a more integrated way that FSG+SVM approach by [9] discussed before. Broadly speaking, the approach involves boosting decision stumps where a decision stump is associated with a graph and represents an existence/nonexistence test in an example to be classified.

The novelty of this work is that the authors have adapted the search mechanism of gSpan which is based on canonical labelling and the DSF code tree for the search for such decision stumps. The key idea behind canonical labelling and the DSF code tree in gSpan is to prune the search space by avoiding the further expansion of candidate subgraphs that have a frequency below the user specified threshold as no supergraph of a candidate can have a higher frequency than itself. This idea cannot be directly applied to graph classification as the objective of the search is not to find frequent subgraphs but subgraphs whose presence or absence distinguishes positive examples form the negative ones. [16] prove a tight upper bound on the gain any supergraph $g\prime$ of a candidate subgraph g can have. Using this result, the proposed algorithm uses the search mechanism of gSpan, calculates and maintains the current highest upper bound on gain τ and prunes the search space by avoiding the further expansion of candidate subgraphs that have a gain lower that τ. The boosting of these decision stumps is identical to the meta learning algorithm AdaBoost introduced by [17].

D. DT-CLGBI

The approach proposed by [18] involves combining aspects of frequent subgraph mining system GBI [15] and decision trees. The approach induces a decision tree where every node is associated with a graph and represents an existence/nonexistence test in an example to be classified.

As described before, the GBI system performs a heuristic, greedy search. In this approach, a variant of the GBI system, B-GBI proposed by [19] which deals with overlapping candidate subgraphs is used for feature generation. Broadly speaking, the approach involves a typical decision tree algorithm except that B-GBI is invoked to generate features at each node, the gain of each feature is computed on the basis of how the existence the feature graph splits the examples at that node and this the procedure is recursively applied until pure nodes with examples only from a single class are reached. In order to avoid over fitting, pessimistic pruning identical to C4.5 by [20] is performed.

III. Direct Product Kernel

Intuitively, the direct product kernel is is based on defining a feature for every possible label sequence in a labelled graph and counting how many label sequences in two given graphs are identical. So basically, the direct product kernel takes as input two graphs and outputs a count of the identical walks that can be taken in both of the graphs (refer Figure 1).

To define the direct product kernel we need some more notions and notation. A walk w in a graph g is a sequence of edges $e_1, e_2, ...e_n$ such that for every $e_i = (u, v)$ and $e_{i+1} = (x, y)$, $v = x$ is obeyed. Every walk is associated with a sequence of edge labels $\alpha(e_1), \alpha(e_2), ...\alpha(e_n)$

An adjacency matrix M_g of graph g is defined as,

$$[M_g]_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

A direct product of two graphs $g_1 = (V_1, E_1, \alpha_1)$ and $g_2 = (V_2, E_2, \alpha_2)$ (with identical edge label alphabet Σ) $g_1 \times g_2$ is defined as,

1) $V_{g_1 \times g_2} = \{(v_1, v_2) \in V_1 \times V_2\}$

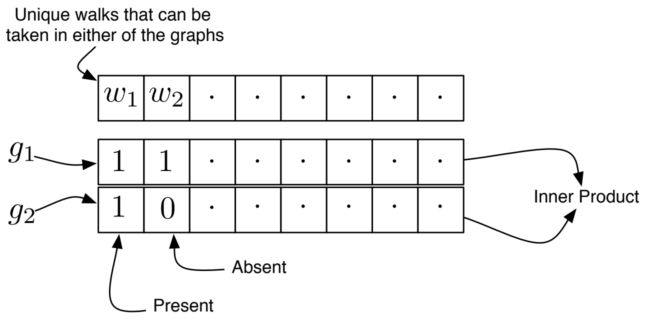

Fig. 1. Feature Space

2) $E_{g_1 \times g_2} = \{((u_1, u_2), (v_1, v_2)) \in E_1 \times E_2\}$ such that,
   a) $(u_1, v_1) \in E_1$
   b) $(u_2, v_2) \in E_2$
   c) $\alpha_1((u_1, v_1)) = \alpha_2((u_2, v_2))$

An important observation here that taking a walk on a direct product graph $g_1 \times g_2$ is equivalent to taking an identical walk on graphs g_1 and g_2. Stated differently, this means that we can take a certain walk on $g_1 \times g_2$ if and only if there exists a corresponding identical walk in both g_1 and g_2. For two graphs $g_1 = (V_1, E_1, \alpha_1)$ and $g_2 = (V_2, E_2, \alpha_2)$ (with identical edge label alphabet Σ) let $M_{g_1 \times g_2}$ be the adjacency matrix of their direct product graph $g_1 \times g_2$. With a sequence of weights $\lambda_0, \lambda_1, ...$ such that $\lambda_i \in \mathbb{R}$ and $\lambda_i \geq 0$ for all $i \in \mathbb{N}$, the direct product kernel $k_{\times(g_1,g_2)}$ is defined as,

$$k_{\times(g_1,g_2)} = \sum_{i,j=1}^{|V_{g_1 \times g_2}|} \left[\sum_{\ell=0}^{\infty} \lambda_\ell M^{\ell}_{g_1 \times g_2} \right]_{ij}$$

if the limit exists.

Intuitively, the direct product kernel computes the powers of the adjacency matrix of the direct product graph $M_{g_1 \times g_2}$ and sums them. This is equivalent to counting the identical walks that can be taken in both the input graphs. This is because any walk in $g_1 \times g_2$ corresponds to an identical walk in both g_1 and g_2 and the ℓ^{th} power of $M_{g_1 \times g_2}$ captures all walks of length ℓ in $M_{g_1 \times g_2}$.

IV. Faster Alternative Computation

In this section we present, our main contribution, a faster alternative to computing the direct product kernel. The key insight behind this is based on showing that the language of edge label sequences corresponding to walks in a graph is a regular language.

Lemma 1 Let $w = e_1, e_2, ...e_n$ be a walk in graph $g = (V, E, \alpha)$ and let $s = \alpha(e_1), \alpha(e_2), ...\alpha(e_n)$ be the sequence of edge labels corresponding to each of the edges in the walk. Let $\mathcal{L}$ be the language of all such sequences corresponding to walks in g. Then, $\mathcal{L}$ is a regular language.

Proof: Construct a finite automaton $M = (Q, \Sigma, \delta, q, F)$ as follows.

The set of states Q is constructed by introducing a state s_v corresponding to every vertex v in g. Additionally, two states s_0 and s_F are introduced in Q. The alphabet Σ is the same as the alphabet of the edge labels. The set of transitions δ is constructed by introducing a transition from state s_u to s_v on symbol $\alpha(e)$ for every directed edge $e = (u, v)$ in g. Additionally, ϵ-transitions are introduced from the state s_0 to every other state in s_v in Q except s_F each of which corresponds to a vertex in g. Also, ϵ-transitions are introduced from every state in Q except for s_0 and s_F, each of which corresponds to a vertex in g to the state s_F. Set the start state $q = s_0$ and the final state $F = s_F$.

By construction, M accepts $\mathcal{L}$ and hence $\mathcal{L}$ is regular. ■

We now show that the language of edge label sequences corresponding to walks in a direct product graph is basically the intersection of the two regular languages of edge labels corresponding to walks in the two graphs and is itself a regular language as regular languages are closed under intersection.

Lemma 2 Let $g_{g_1 \times g_2}$ be the direct product graph of g_1 and g_2. Let $\mathcal{L}_{g_1 \times g_2}$ be the language of all sequences of edge labels corresponding to walks in $g_{g_1 \times g_2}$. Similarly, let $\mathcal{L}_{g_1}$ and $\mathcal{L}_{g_2}$ be languages of all sequences of edge labels corresponding to walks in g_1 and g_2 respectively. Then, $\mathcal{L}_{g_1 \times g_2} = \mathcal{L}_{g_1} \cap \mathcal{L}_{g_2}$ is regular.

Proof: From the definition of direct graph product, taking a walk on a direct product graph $g_1 \times g_2$ is equivalent to taking an identical walk on graphs g_1 and g_2. Each of these walks has a corresponding edge label sequence associated with it. As $\mathcal{L}_{g_1 \times g_2}$ is the language of all sequences of edge labels corresponding to walks in $g_{g_1 \times g_2}$, $\mathcal{L}_{g_1}$ and $\mathcal{L}_{g_2}$ are languages of all sequences of edge labels corresponding to walks in g_1 and g_2 respectively, thus $\mathcal{L}_{g_1 \times g_2} = \mathcal{L}_{g_1} \cap \mathcal{L}_{g_2}$ follows. ■

We now introduce the notion of union of the two languages corresponding to the sequence of edge labels corresponding to walks in the two graphs. Let $g_1 = (V_1, E_1, \alpha_1)$ and $g_2 = (V_2, E_2, \alpha_2)$ (with identical edge label alphabet Σ) then the union of the corresponding languages $\mathcal{L}_{g_1}$ and $\mathcal{L}_{g_2}$ is denoted by $\mathcal{L}_{g_1 \cup g_2}$.

Lemma 3 $\mathcal{L}_{g_1 \cup g_2}$ is regular.

Proof: Follows from the definitions and the property the regular languages are closed under union. ■

We can now show a result that gives us a relation between the sizes of the languages considered so far.

Lemma 4 Let $g_{g_1 \times g_2}$ be the direct product graph of g_1 and g_2. Let $\mathcal{L}_{g_1 \times g_2}$ be the language of all sequences of edge labels corresponding to walks in $g_{g_1 \times g_2}$. Let $g_{g_1 \cup g_2}$ be the union graph of g_1 and g_2. Let $\mathcal{L}_{g_1 \cup g_2}$ be the language of all sequences of edge labels corresponding to walks in $g_{g_1 \cup g_2}$. Similarly, let $\mathcal{L}_{g_1}$ and $\mathcal{L}_{g_2}$ be languages of all sequences of edge labels corresponding to walks in g_1 and g_2 respectively. Then, $|\mathcal{L}_{g_1 \times g_2}| = |\mathcal{L}_{g_1}| + |\mathcal{L}_{g_2}| - |\mathcal{L}_{g_1 \cup g_2}|$.

Proof: Follows from Lemmas 1, 2 and 3 and the property that regular languages are closed under union and intersection. ■

This result can be easily extended to subsets of these languages which place a restriction on the size of the sequences in the language.

Lemma 5 Let $\mathcal{L}^{\ell}_{g_1}$, $\mathcal{L}^{\ell}_{g_2}$ $\mathcal{L}^{\ell}_{g_1 \times g_2}$ and $\mathcal{L}^{\ell}_{g_1 \cup g_2}$ be subsets of languages $\mathcal{L}_{g_1}$, $\mathcal{L}_{g_2}$ $\mathcal{L}_{g_1 \times g_2}$ and $\mathcal{L}_{g_1 \cup g_2}$ such that they do not contain sequences longer than ℓ. Then $\mathcal{L}^{\ell}_{g_1}$, $\mathcal{L}^{\ell}_{g_2}$ $\mathcal{L}^{\ell}_{g_1 \times g_2}$ and $\mathcal{L}^{\ell}_{g_1 \cup g_2}$ are regular and $|\mathcal{L}^{\ell}_{g_1 \times g_2}| = |\mathcal{L}^{\ell}_{g_1}| + |\mathcal{L}^{\ell}_{g_2}| - |\mathcal{L}^{\ell}_{g_1 \cup g_2}|$ holds.

Proof: Clearly $\mathcal{L}^{\ell}_{g_1}$, $\mathcal{L}^{\ell}_{g_2}$ $\mathcal{L}^{\ell}_{g_1 \times g_2}$ and $\mathcal{L}^{\ell}_{g_1 \cup g_2}$ are finite languages by definition and hence regular. The result follows from Lemma 4 and the property that regular languages are closed under union and intersection. ■

The problem thus reduces to counting the number of strings in a regular language of length no more than ℓ. It has been shown that this problem can be solved in $O(n^3)$ [21] where n is the number of states in the finite automata for the regular language. This approach is also based on diagonalising the adjacency matrix of the finite automata. Note here that in order to compute $|\mathcal{L}^{\ell}_{g_1 \times g_2}|$ we compute $|\mathcal{L}^{\ell}_{g_1}| + |\mathcal{L}^{\ell}_{g_2}| - |\mathcal{L}^{\ell}_{g_1 \cup g_2}|$. The largest finite automata and hence the largest adjacency matrix to be diagonalized is $|\mathcal{L}^{\ell}_{g_1 \cup g_2}|$. The previous approach in essence dealt with $|\mathcal{L}^{\ell}_{g_1 \times g_2}|$. The key difference from the previous computation is that now we are dealing with finite automata corresponding to the union of two regular languages which grows as $O(m + n)$ for automatas with sizes m and n (for the two input graphs) instead of the finite automata corresponding to the intersection of the two languages which grows as $O(mn)$. This implies that direct product kernel can be computed in $O((m + n)^3)$ instead of $O((mn)^3)$.

V. Experiments

We compare the proposed alternative kernel computation to the approximation of the kernel by matrix multiplication. Figure 9 shows the training time for the Mutagenesis and PTC datasets while approximating the kernel value using matrix multiplication. Figure 10 shows the time for the alternative computation (evaluation the inverse of the union matrix). The results indicate that although the alternative computation is expensive as compared to approximation for small walks, it is comparable when compared to approximation for long walks. Longer walks, in general, lead to higher accuracy(refer Figure 8) but after a certain walk length, we have diminishing returns in the sense that the extra expense of computation does not buy us better predictive accuracy. So the predictive accuracy flattens out at a particular walk length where approximation by matrix multiplication turns to be cheaper. In general, it must be noted that approximation of the kernel by matrix multiplication may turn out to be cheaper that the alternative computation in practice for smaller walk sizes.

VI. Significance of Faster Kernel Computation

In this section we focus of how the direct product kernel is currently computed and how this is infeasible for real world data sets.

Gärtner et al., 2003, describe two ways in which the direct product kernel can be computed, the first based on matrix diagonalizing and the second based on matrix inversion. The key issue in computing the kernel is that of computing the powers of the adjacency matrix of the direct product graph.

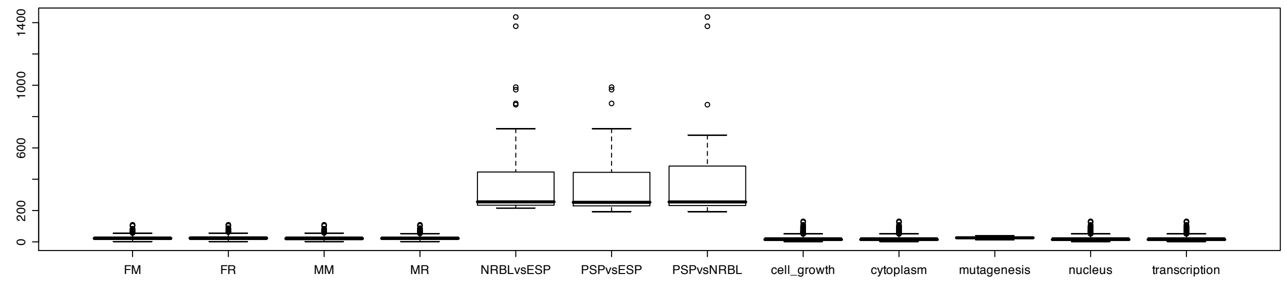

Fig. 3. Number of vertices

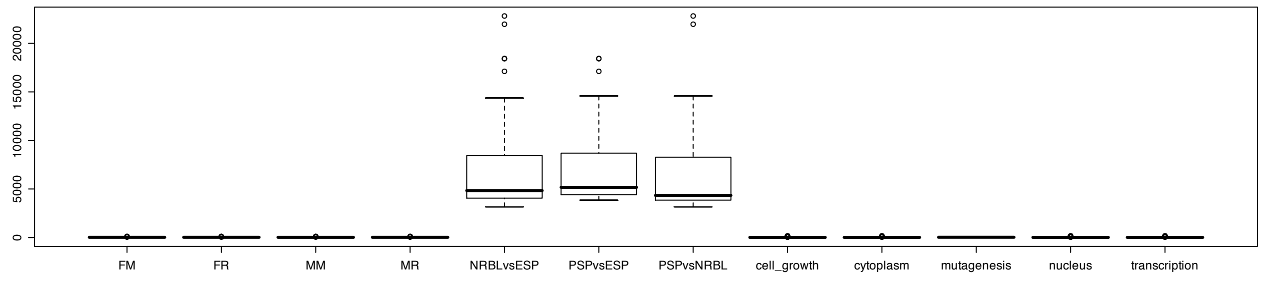

Fig. 4. Number of edges

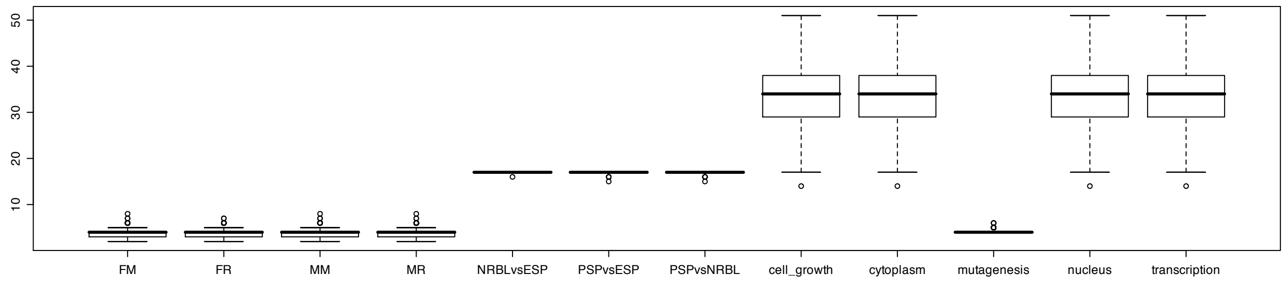

Fig. 5. Number of vertex labels

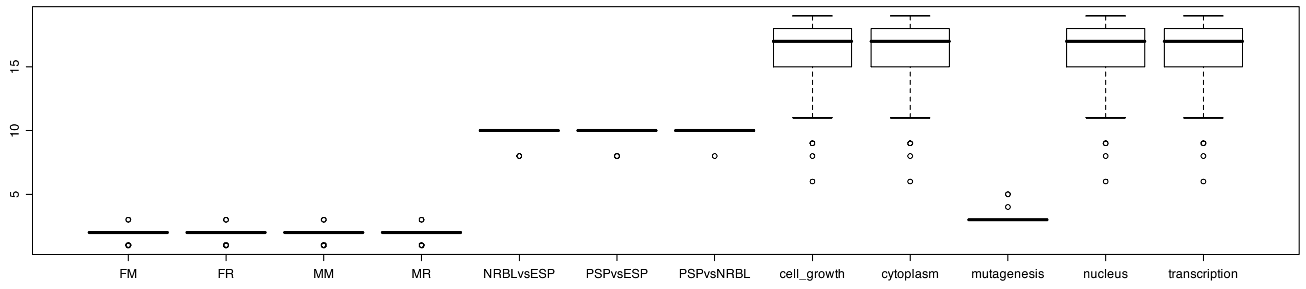

Fig. 6. Number of edge labels

Dataset	Mutag	PTC				Proteins		Gene			
								Function		Localization	
Task	Mutag	MR	FR	MM	FM	PSP vs ESP	NRBL vs ESP	Cell Growth	Transcrip.	Nucleus	Cyto.
# Pos	125	152	121	129	143	13	21	275	271	367	192
# Neg	63	192	230	207	206	42	42	587	591	495	670
# Vertices (Total)	4893	8792	9115	8416	8811	19349	24131	18258	18258	18258	18258
Average Vertices (Per Example)	26.02	25.55	25.96	25.04	25.31	351.8	383.032	21.81	21.81	21.81	21.81
# Edges (Total)	5243	8931	9312	8533	8941	390755	445665	19149	19149	19149	19149
Average Edges (Per Example)	27.88	25.96	26.52	25.39	25.69	7104.64	7074.05	22.21	22.21	22.21	22.21
Average Degree	1.07	1.01	1.02	1.01	1.01	20.19	18.46	1.04	1.04	1.04	1.04
# Vertex Labels	8	19	20	21	19	20	20	339	339	339	339
# Edge Labels	4	4	4	4	4	2	2	10	10	10	10

Fig. 7. Properties of graph classification data sets

The approach based on matrix diagonalizing involves diagonalizing the adjacency matrix of the direct product graph. If $M_{g_1 \times g_2}$ can be expressed as $M_{g_1 \times g_2} = T^{-1}DT$, then $M^n_{g_1 \times g_2} = (T^{-1}DT)^n$. Then, $M^n_{g_1 \times g_2} = T^{-1}D^nT$ and computing arbitrary powers of the diagonal matrix D can be performed in linear time. It follows that the hardness of computing the direct product kernel is equivalent to that of diagonalizing the adjacency matrix of the direct product graph. Matrix diagonalizing is $O(m^3)$ where m is the size of the matrix. The second approach based on matrix inversion involves inverting a matrix equal in size to the product graph (for details refer to Gärtner et al., 2003). The point to note here is that this approach involves inverting a matrix equal in size to the adjacency matrix of the direct product graph, and matrix inversion is $O(m^3)$ where m is the size of the matrix.

Now, as the kernel computation involves either the diagonalizing or the inversion of a matrix equal to the size of the adjacency matrix of the direct product graph, the complexity of this computation depends on the size of the direct product graph. From the definition of the direct product graph in the previous section it can be seen that the size of the adjacency matrix of the direct product graph for two graphs with adjacency matrices of sizes m and n, is $m \times n$ in the worst case. It follows that the complexity of the kernel computation is $O((mn)^3)$.

In order to investigate if this computation is feasible for real world graph classification tasks, we conducted a survey of graph classification data sets and studied their properties. Our survey included chemical compounds from the Mutagenesis data set [22] and the PTC data set, protein data from SCOP [23] and gene interaction data from the KDD Cup 2001 dataset.

The Mutagenesis data [22] set has been used as a benchmark data set in graph classification for many years. The data set has been collected to identify mutagenic activity in a compound based on its molecular structure.

The Predictive Toxicology Challenge (PTC) [24] data set has also been used in the literature for several years. The PTC carcinogenesis databases contain information about chemical compounds and the results of laboratory tests made on rodents in order to determine if the chemical induces cancer. The data consists of four categories: male rats MR, female rats FR, male mice MM, or female mice FM.

The Proteins data set comes from the Structural Classification of Proteins (SCOP) and has also been used to evaluate graph-based supervised learning methods. The graph

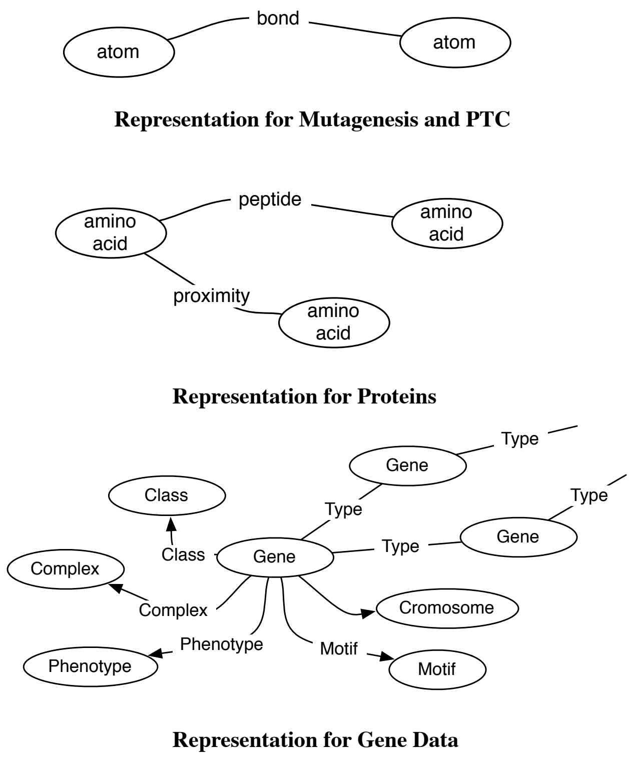

Fig. 2. Graph Representation

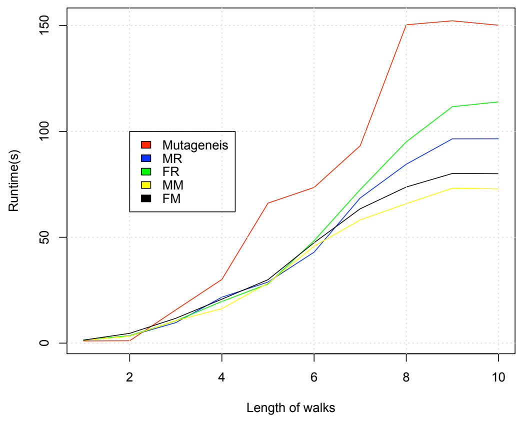

Fig. 9. Approximation using matrix multiplication

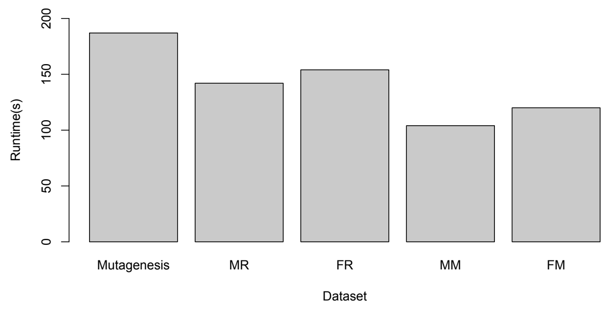

Fig. 10. Alternative Computation

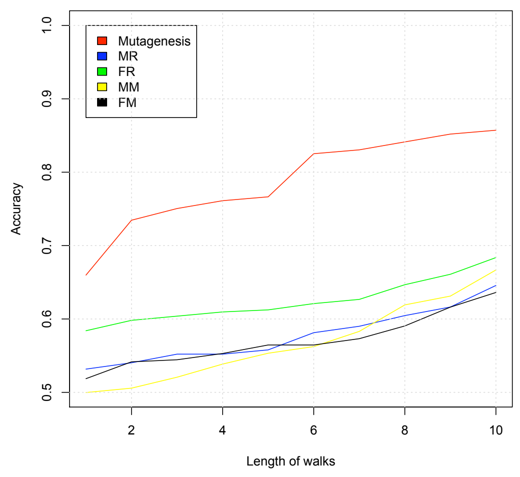

Fig. 8. Accuracy vs. Length of Walk

classification data sets studied were constructed from three SCOP protein families. [25] The first family is the nuclear receptor ligand-binding domain of proteins (NRBL) from the all-alpha class. The second family is the prokaryotic serine proteases (PSP) from the all-beta class. The third family is the eukaryotic serine proteases (ESP). PSP and ESP belong to the same superfamily.

The Gene interaction data comes from the 2001 KDD Cup [26]. The data set consists of information about the various genes of a particular organism. There are two tasks: predict the functions and localizations of the proteins encoded by the genes. For the function prediction task, we use only two functions: cell growth and transcription. For the localization task, we use only two locations: nucleus or cytoplasm.

Each of the data sets were represented as graphs and the schematic representation of the representation is shown in Figure 2. Box-and-whisker plots of the number of vertices, number of edges, number of vertex labels and number of edge labels of the graphs in these datasets are illustrated in Figures 3, 4 and 5, 6 respectively. Additional properties of the data sets are shown in Figure 7. Note that SCOP graphs have more than 300 vertices on average, which will definitely make the computation of the kernel for these graphs infeasible. Also, note the high average degree for SCOP and the high number of unique labels in the Gene data set which might lead to an increased number of unique walks.

Although there can be a number of ways in which to represent this data with a graph-based representation it can be seen that most graphs will have adjacency matrices of sizes in the order of hundreds which will lead to adjacency matrices of direct product graphs (of two graphs) having sizes in the order of thousands. It is therefore not feasible to compute the kernel using the approaches described in this section for most real world data sets.

An argument against the infeasibility of this computation is that although computing the kernel for longer walks is infeasible, the kernel can be easily computed for shorter walks by direct matrix multiplication which might lead to a sufficiently high accuracy. Such an approximation was also used by Gärtner et al. for all the experiments in the original paper that introduced this kernel. We performed experiments to analyse the effect of the length of the walk on the classification accuracy. Experiments were performed of two of the four data sets discussed above namely, Mutagenesis and PTC. The result of these experiments are shown in Figure 8. As it can be seen, longer walks can lead to higher accuracy in certain cases and hence faster computation is essential in achieving higher accuracy.

VII. Conclusion

In this work we show how the direct product kernel can be computed in $O((m+n)^3)$ instead of $O((mn)^3)$ for two graphs with adjacency matrices of sizes m and n. As real world datasets in graph classification have graphs with adjacency matrices of sizes in the order of hundreds this result is important in applying the direct product kernel to real world data sets.

The observation that the language of label sequences in a labeled graph is a regular language is important as it can be the starting point for applying a number of results from automata theory to compute graph kernels. We pursue this as a part of our future work.

References

[1] J. A. Gonzalez, L. B. Holder, and D. J. Cook, “Graph-based relational concept learning.” in *ICML*, 2002, pp. 219–226.

[2] M. Deshpande, M. Kuramochi, G. Karypis, and M. U. M. D. O. C. SCIENCE, “Frequent Sub-Structure-Based Approaches for Classifying Chemical Compounds,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 8, pp. 1036–1050, 2005.

[3] P. Nguyen, K. Ohara, A. Mogi, H. Motoda, and T. Washio, “Constructing Decision Trees for Graph-Structured Data by Chunkingless Graph-Based Induction,” *Proc. of PAKDD*, pp. 390–399, 2006.

[4] T. Kudo, E. Maeda, and Y. Matsumoto, “An application of boosting to graph classification.” in *NIPS*, 2004.

[5] T. Gärtner, P. A. Flach, and S. Wrobel, “On graph kernels: Hardness results and efficient alternatives,” in *COLT*, 2003, pp. 129–143.

[6] H. Kashima, K. Tsuda, and A. Inokuchi, “Marginalized kernels between labeled graphs.” in *ICML*, 2003, pp. 321–328.

[7] J. Gonzalez, L. Holder, and D. Cook, “Graph-based relational concept learning,” *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.

[8] J. Rissanen, *Stochastic Complexity in Statistical Inquiry Theory*. World Scientific Publishing Co., Inc., River Edge, NJ, 1989.

[9] M. Deshpande, M. Kuramochi, and G. Karypis, “Frequent Sub-Structure-Based Approaches for Classifying Chemical Compounds,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 8, pp. 1036–1050, 2005.

[10] A. Inokuchi, T. Washio, and H. Motoda, “An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data,” *Principles of Data Mining and Knowledge Discovery: 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000: Proceedings*, 2000.

[11] M. Kuramochi and G. Karypis, “Frequent subgraph discovery,” *Proceedings of the 2001 IEEE International Conference on Data Mining*, pp. 313–320, 1929.

[12] X. Yan and J. Han, “gSpan: Graph-based substructure pattern mining,” *Proc. 2002 Int. Conf. on Data Mining (ICDM02)*, pp. 721–724, 2002.

[13] S. Nijssen and J. Kok, “A quickstart in frequent structure mining can make a difference,” *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 647–652, 2004.

[14] D. Cook and L. Holder, “Substructure Discovery Using Minimum Description Length and Background Knowledge,” *Journal of Artificial Intelligence Research*, vol. 1, pp. 231–255, 1994.

[15] H. Motoda and K. Yoshida, “Machine learning techniques to make computers easier to use,” *Artificial Intelligence*, vol. 103, no. 1-2, pp. 295–321, 1998.

[16] T. Kudo, E. Maeda, and Y. Matsumoto, “An application of boosting to graph classification,” *Advances in Neural Information Processing Systems*, vol. 17, pp. 729–736, 2005.

[17] Y. Freund and R. Schapire, “Experiments with a new boosting algorithm,” *Machine Learning: Proceedings of the Thirteenth International Conference*, vol. 148, p. 156, 1996.

[18] P. Nguyen, K. Ohara, A. Mogi, H. Motoda, and T. Washio, “Constructing Decision Trees for Graph-Structured Data by Chunkingless Graph-Based Induction,” *Proc. of PAKDD*, pp. 390–399, 2006.

[19] T. Matsuda, H. Motoda, T. Yoshida, and T. Washio, “Mining Patterns from Structured Data by Beam-Wise Graph-Based Induction,” *Discovery Science: 5th International Conference, DS 2002, Lübeck, Germany, November 24-26, 2002: Proceedings*, 2002.

[20] J. Quinlan, *C4. 5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[21] B. Ravikumar and G. Eisman, “Weak minimization of dfa: an algorithm and applications,” *Theor. Comput. Sci.*, vol. 328, no. 1-2, pp. 113–133, 2004.

[22] A. Srinivasan, S. Muggleton, M. J. E. Sternberg, and R. D. King, “Theories for mutagenicity: A study in first-order and feature-based induction.” *Artif. Intell.*, vol. 85, no. 1-2, pp. 277–299, 1996.

[23] L. L. Conte, B. Ailey, T. J. P. Hubbard, S. E. Brenner, A. G. Murzin, and C. Chothia, “Scop: a structural classification of proteins database.” *Nucleic Acids Research*, vol. 28, no. 1, pp. 257–259, 2000.

[24] C. Helma, R. King, S. Kramer, and A. Srinivasan, “The Predictive Toxicology Challenge 2000-2001,” pp. 107–108, 2001.

[25] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha, “Mining protein family specific residue packing patterns from protein structure graphs.” in *RECOMB*, 2004, pp. 308–315.

[26] J. Cheng, C. Hatzis, H. Hayashi, M.-A. Krogel, S. Morishita, D. Page, and J. Sese, “KDD cup 2001 report,” *SIGKDD Explorations*, vol. 3, no. 2, pp. 47–64, 2002.