

Protein function prediction via *faster* graph kernels*

K. M. Borgwardt¹, S.V.N. Vishwanathan², Nic Schraudolph² and H.-P. Kriegel¹

¹ Institute for Computer Science, Ludwig-Maximilians-University of Munich,
Oettingenstr. 67, 80538 Munich, Germany

² Statistical Machine Learning Program, National ICT Australia,
Canberra, 0200 ACT, Australia

Abstract

Kernel functions on graphs have been defined over recent years. In earlier work, we have employed random walk graph kernels for predicting protein function from graph representations that integrate both protein sequence and structure. While yielding good protein function prediction results, random walk graph kernels suffer from a high computational complexity of $O(n^6)$ where n is the number of nodes in the input graphs. In this paper, we present an approach for speeding up graph kernels. It is based on the observation that random walks on a graph can be regarded as a dynamical system and makes use of conjugate gradient.

1 Introduction

Random walk kernels are based on the idea to count the number of matching walks in two input graphs. Gärtner et al. [4] define an elegant approach to determine all pairs of matching walks in two input graphs $G_1 = (V_1, A_1)$ and $G_2 = (V_2, A_2)$ via a direct product graph G_\times :

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{k=0}^{\infty} \lambda_k A_\times^k \right]_{ij},$$

where A_\times is the adjacency matrix of $G_\times = (V_\times, A_\times)$, defined via

$$\begin{aligned} V_\times(G_1 \times G_2) &= \{(v_1, w_1) \in V_1 \times V_2 : \\ &\quad \text{label}(v_1) = \text{label}(w_1)\} \\ A_\times(G_1 \times G_2) &= \{((v_1, w_1), (v_2, w_2)) \in V^2(G_1 \times G_2) : \\ &\quad (v_1, v_2) \in A_1 \wedge (w_1, w_2) \in A_2 \\ &\quad \wedge (\text{label}(v_1, v_2) = \text{label}(w_1, w_2))\}, \end{aligned}$$

and λ_k must be chosen appropriately for k_\times to converge. If we define λ_k as λ^k for all $k \in \{1, \dots, \infty\}$, we obtain a geometric series converging for $\lambda < \frac{1}{a}$ where $a \geq \min\{\text{maximal in-degree of } G_\times, \text{maximal out-degree of } G_\times\}$:

*This work was supported in part by National ICT Australia and by the German Ministry for Education, Science, Research and Technology (BMBF) under grant no. 031U112F within the BFAM (Bioinformatics for the Functional Analysis of Mammalian Genomes) project which is part of the German Genome Analysis Network (NGFN). National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

$$k_{\times}(G_1, G_2) = \sum_{i,j=1}^{|V_{\times}|} \left[\sum_{k=0}^{\infty} \lambda^k A_{\times}^k \right]_{ij} = \mathbf{e}^{\top} (\mathbf{1} - \lambda A_{\times})^{-1} \mathbf{e} \quad (1)$$

where \mathbf{e} denotes the vector of all ones, and $\mathbf{1}$ denotes the identity matrix. A direct computation of the matrix inversion is prohibitively expensive. If we assume that the number of vertices in G_1 and G_2 is n , then construction of the product adjacency matrix A_{\times} requires $O(n^2)$ time. Recall that the inversion of a matrix is cubic in its dimensions. Hence, inversion of $(\mathbf{1} - \lambda A_{\times})^{-1}$ requires $O(n^6)$ time. The purpose of this short note is to show how Conjugate Gradient, and Sylvester equation based methods can be used to considerably speed up computation of these graph kernels.

2 Protein Graph Kernels

Using Gärtner’s approach and ideas in [5], this kernel is redefined in [2] to measure partial similarities between walks that are not completely identically labeled. Node and edge labels along the walks are compared via kernel functions.

This kernel is then applied to the task of protein function prediction [2]. Proteins are classified into functional classes, first into enzymes and non-enzymes, second -if they are enzymes - into one of the 6 EC top level classes. For this purpose, proteins are modeled as graphs, in which nodes represent secondary structure elements and edges represent neighborhood within the 3D structure or along the amino acid chain. Comparing these graphs via the modified random walk kernel and classifying them via Support Vector Machines leads to function prediction accuracies that compare favorably with state-of-the-art approaches.

As noted before, computation of the random walk kernel is costly. In [2], problems with up to 1200 protein graphs are considered, yet scaling up these to large datasets of tens of thousands of proteins poses a practical problem when dealing with a runtime in $O(n^6)$.

Furthermore, inverting a large product graph adjacency matrix often leads to memory problems; as a consequence, experiments using random walk graph kernels only consider walks up to a fixed length only. Therefore, we need to look for efficient methods to compute these graph kernels. We describe such an approach in the next section.

3 Efficient Computation of Graph Kernels

We begin by introducing some notation. Given a matrix $A \in \mathbb{R}^{n \times m}$, we use $A_{:,i}$ to denote the i -th column of A , $A_{i,:}$ to denote the i -th row and $A_{i,j}$ to denote the (i, j) -th element of A . The linear operator $\text{vec} : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^{nm}$ flattens the matrix [1]. In other words,

$$\text{vec}(A) := \begin{bmatrix} A_{:,1} \\ A_{:,2} \\ \vdots \\ A_{:,m} \end{bmatrix}.$$

while the vec^{-1} operator reconstructs the matrix from a flattened matrix.

Given two matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{l \times k}$ the Kronecker product $A \odot B \in \mathbb{R}^{nl \times mk}$ is defined as [1]

$$A \odot B := \begin{bmatrix} A_{1,1}B & A_{1,2}B & \dots & A_{1,n}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n,1}B & A_{n,2}B & \dots & A_{n,m}B \end{bmatrix}$$

Observe that the adjacency matrix of a product graph may be thought as a tensor product of the adjacency matrix of the individual graphs. In other words,

$$A_{\times}(G_1 \times G_2) = A(G_1) \odot A(G_2).$$

In more detail, the tensor product $A(G_1) \odot A(G_2)$ is the adjacency matrix of a product graph in which each row (and each column) represents one pair of nodes from G_1 and G_2 . Entries in this adjacency matrix are 1, if the corresponding nodes are adjacent in both G_1 and G_2 .

Now consider the following equation:

$$M = SMT + U, \tag{2}$$

where $S, T, U \in \mathbb{R}^{n \times n}$ are given and we need to solve for $M \in \mathbb{R}^{n \times n}$. Equations of this form are known as Sylvester equations, and can be readily solved in $O(n^3)$ time with freely available code [3].

We show how the problem of solving the Sylvester equation can be systematically converted into the problem of computing the graph kernel (1). We begin by first rewriting the above equation as

$$\text{vec}(M) = \text{vec}(SMT) + \text{vec}(U). \tag{3}$$

Now, using the well known identity (see proposition 7.1.9) [1]

$$\text{vec}(SMT) = (T^{\top} \odot S) \text{vec}(M), \tag{4}$$

we can rewrite (3) as

$$(\mathbf{1} - T^{\top} \odot S) \text{vec}(M) = \text{vec}(U), \tag{5}$$

where $\mathbf{1}$ denotes the identity matrix of appropriate dimensions. In order to compute M we need to solve the $n^2 \times n^2$ system

$$\text{vec}(M) = (\mathbf{1} - T^{\top} \odot S)^{-1} \text{vec}(U). \tag{6}$$

Multiplying both sides of the above equation by $\text{vec}(U)$ we obtain

$$\text{vec}(U) \text{vec}(M) = \text{vec}(U) (\mathbf{1} - T^{\top} \odot S)^{-1} \text{vec}(U).$$

In order to recover (1) we only need to set $U \in \mathbb{R}^{n \times n}$ to be the matrix of all ones, $T = A(G_1)$ and $S = A(G_2)$. In other words computing the graph kernels is equivalent to solving the Sylvester equation. As pointed out above, solving the Sylvester equations can be done in $O(n^3)$ time and hence is significantly faster than the $O(n^6)$ time required by the direct approach.

But, if S and T are sparse then (6) can be computed even more cheaply. By exploiting sparsity, and using (4) we can compute $(\mathbf{1} - T^\top \odot S) \text{vec } X$ for any matrix X rather cheaply. This naturally suggests two strategies: We can iteratively solve for a fixed point of (6) as is done in [6]. But the main drawback of this approach is that the fixed point iteration might take a long time to converge. On the other hand, $\text{rank}(T^\top \odot S) = \text{rank}(T) \text{rank}(S)$, and hence one can employ a Conjugate Gradient (CG) solver which can exploit both the sparsity of as well as the low rank of S and T and provide guaranteed convergence [7]. Let $\{\lambda_i\}$ denote the eigenvalues of S and $\{\mu_j\}$ denote the eigenvalues of T . Then the eigenvalues of $S \odot T$ are given by $\{\lambda_i \mu_j\}$ [1]. Therefore, if the eigenvalues of S and T are bounded (as they are in the case of a graph) and rapidly decaying then the eigenvalues of $S \odot T$ are bounded and rapidly decaying. In this case a CG solver will be able to efficiently solve (6). It is worthwhile reiterating that both the above approaches are useful only when S and T are sparse and have low (effective) rank. For the general case, solving the Sylvester equation directly is the recommended approach.

4 Outlook and Future Work

As more and more graph data become available in biology, e.g. molecular structures and protein interaction networks, graph classification will gain even more importance over coming years. Within this process, faster graph kernels, sped up by conjugate gradient, will further increase the attractiveness of kernel methods in graph classification in bioinformatics. We are currently testing out the ideas outlined in this paper to large scale datasets and hope to report results very soon.

References

- [1] D. S. Bernstein. *Matrix Mathematics*. Princeton University Press, 2005.
- [2] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H. Kriegel. Protein function prediction via graph kernels. In *Proc. of Intelligent Systems in Molecular Biology (ISMB)*, Detroit, USA, 2005.
- [3] J. D. Gardiner, A. L. Laub, J. J. Amato, and C. B. Moler. Solution of the Sylvester matrix equation $AXB^\top + CXD^\top = E$. *ACM Transactions on Mathematical Software*, 18(2):223–231, 1992.
- [4] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In B. Schölkopf and M. Warmuth, editors, *Sixteenth Annual Conference on Computational Learning Theory and Seventh Kernel Workshop, COLT*. Springer, 2003.
- [5] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, Washington, DC, United States, 2003.
- [6] H. Kashima, K. Tsuda, and A. Inokuchi. Kernels on graphs. In K. Tsuda, B. Schölkopf, and J. Vert, editors, *Kernels and Bioinformatics*, Cambridge, MA, 2004. MIT Press.
- [7] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.