Partially-Observed Graph Abstractions for Authorship Identification and Process Interference Prediction

by

Rudolf Plesch

B. Applied Science, The University of British Columbia, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

THE FACULTY GRADUATE AND POSTDOCTORAL STUDIES
(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

February 2017

© Rudolf Plesch, 2017

Abstract

Pairwise interactions between objects can be modeled as a graph, which is a set of nodes and edges that each connect a pair of nodes. We consider the problem of predicting whether edges exist between nodes based on other pairs of nodes that we have observed. From a partially-observed graph we extract several neighbourhood and path features, then evaluate several machine learning algorithms for predicting whether edges will exist between unobserved nodes. K-Nearest Neighbours was found to be the best classifier. We propose the novel use of path on a weighted graph as a feature used for prediction.

We apply this abstraction to predicting collaboration between authors in an online publication database. The unweighted graph contains an edge if two authors collaborated and the weighted graph encodes the number of collaborations. Prediction error rates were less than 3% under ten-fold cross-validation.

We also apply this abstraction to predicting whether processes running on the same hardware will compete for resources. The unweighted graph contains an edge if a process executed in more time than when running with another than by itself. The weighted graph had an edge weight that was the increase in execution time. Prediction error rates were less than 14% under ten-fold cross-validation.

Preface

I designed the graph abstraction for the authorship identification. My supervisor Sathish Gopalakrishnan prompted me to consider the process interference application.

The work in Chapter 3 was part of a group project by Hootan Rashtian and Rudolf Plesch. I performed the link prediction work presented in the chapter in its entirety. Our project neatly divided between the link prediction component and a text mining component. Hootan Rashtian contributed the text mining component.

The work in Chapter 4 is part of a group project with Naga Raghavendra Surya and Rudolf Plesch. I identified the research problem, adapted the graph abstraction to this purpose, and implemented the machine learning algorithm. He created the data set and found some of the benchmarks. That project presented a different method of creating the weighted graph, which I've since refined, improving the results.

These works have not been published.

Table of Contents

Abstract ii

Preface iii

Table of Contents iv

List of Tables vi

List of Figures vii

Glossary viii

Acknowledgments ix

1 Introduction 1

2 Inference on Partially Observed Graphs 5

2.1 Feature Selection 6

2.1.1 Neighbourhood Features 6

2.1.2 Path Features 7

2.2 Statistical Analysis 7

2.3 Relationship to Matrix Completion 8

3 Application to Authorship Identification 10

3.1 Graph Construction 11

3.2 Results 12

3.2.1 Feature Distribution 12
3.3 Conclusion 14

4 Application to Process Interference 16
4.1 Related Work 17
4.2 Graph Construction 19
4.3 Results 23
4.3.1 Feature Distribution 24
4.4 Conclusion 25

5 Conclusion 28

Bibliography 30

A List of Benchmarks 34

List of Tables

Table 3.1 Average Classification Error Rate (CER) over several runs of cross-validation for all classifiers. 12

Table 4.1 Density of graph and CER for several choices of SC threshold T. CER is the fraction of correct predictions over the test set. . 24

List of Figures

Figure 3.1 Scatter plot of the Adamic/Adar score and number of common neighbours for the authorship collaboration graph. Red points are collaborating pairs and blue points are non-collaborating pairs. 13

Figure 3.2 Scatterplot of Katz score and weighted shortest path for the data set. Red points are collaborating pairs and blue points are non-collaborating pairs. 14

Figure 3.3 Scatterplot of unweighted and weighted shortest path for the data set. Red points are collaborating pairs and blue points are non-collaborating pairs. 15

Figure 4.1 Process of creating unweighted contention graph. Red pairs were classified as interfering, while green ones were not. . . . 20

Figure 4.2 Process of creating weighted contention graph. The edge weight is the inverse of the Slowdown Coefficient (SC). 21

Figure 4.3 The interference graph generated when setting T=2.5. 22

Figure 4.4 Slowdown Coefficients distribution of a typical application. . 23

Figure 4.5 Scatter plot of the Adamic/Adar score and number of common neighbours for the process interference graph. Red points are pairs that interfered and blue points are pairs that did not. . . . 25

Figure 4.6 Scatter plot of the weighted and unweighted shortest path for the process interference graph. Red points are pairs that interfered and blue points are pairs that did not. 26

Glossary

SGD Stochastic Gradient Descent

KNN K-Nearest Neighbours

SVM Supported Vector Machines

SC Slowdown Coefficient

LLC Last-Level Cache

CER Classification Error Rate, the fraction of misclassified objects in the test set.

Acknowledgments

I'd like to thank Tibi, Carmen, and Talise; Sathish Gopalakrishnan for letting me explore and asking so little of me; Alex Bouchard-Côté, Jozsef Nagy, and Mark Bottrill for showing me what a great teacher can do. What I've achieved attests poorly to their teaching.

Chapter 1

Introduction

A graph is an abstraction that can encode pairwise interactions between objects. It is defined as a set of objects, called nodes, and a set a set of edges which link two nodes together. Graphs can also be weighted, where a real-number is assigned to each edge, known as a weight. A partially-observed graph is one where we have observes a set of pairs of nodes and we know whether there are edges between the nodes or not. Based on these observations, we extract some features from the graph and attempt to predict whether edges exist between unobserved pairs of nodes. This problem has been studied in the field of social networks, where it is known as link prediction [18][8][16]. In this domain, the questions researchers usually pose is whether two users of a social network who are not linked are likely to become linked. These applications have only considered unweighted graphs. In this work we propose building a weighted graph and using weighted paths as a new feature for predicting these interactions. We also apply these methods to two novel applications: authorship collaboration in a scientific publication database and process interference on a multi-core system.

In each of the applications we encode the objects in the application as node and the event we are trying to predict as an edge on a graph. We then observe some interactions between objects and we construct two graphs: a weighted and an unweighted one. We extract some topological features of the graph and attempt to predict whether edges exist between the nodes we have not observed in the graph. Using 10-fold cross-validation, we evaluate several machine learning algorithms

for predicting the existence of edges. We also evaluate the topological graph features we extracted to show which have the most predictive power.

The problem of predicting whether edges exist in a partially-observed graph is closely related to the mathematical study of matrix completion, which seeks to complete a matrix using only observations of a few entries[13]. While we do not apply matrix completion methods to our problems, we discuss the how matrix completion applies to our problem and give some directions for future research.

Our first application of link prediction is predicting collaborations in an online publication database. The link prediction will be used to augment traditional text mining methods in a system used to disambiguate author names in a database. Ambiguous author names can exist because authors can sometime use full names, sometimes initials, and inconsistent use of accents. They can also occur when multiple authors share a name. Disambiguating author names is an important problem in online publication databases because it enables accurate accumulation of papers by author.

Currently, authorship identification is preformed by mining the text of a work for certain features. This does not apply, however, to scientific publications which often have several authors, some of which may not have contributed to the paper text. We encoded co-authored publications as edges on an unweighted graph and recurring co-authorships as decreasing edge weights on a weighted graph. Our link prediction error rate was less than 3%.

The second application is predicting whether two user processes that are running on one piece of hardware will interfere with each other. This is an important industry problem because much of today's computation is performed in large datacentres instead of on dedicated hardware. For example, services like Amazon's AWS sell computation time to users and promise a number of cores and an amount of memory, this may be less than the number of cores and memory of a unit of hardware in the datacentre and the provider may choose to put several users' jobs on one piece of hardware. Running multiple processes on a single piece of hardware can reduce the hardware needed for computations, but it risks that the processes running on the hardware will compete for resources and the performance will degrade below an acceptable level. Using a method to predict which jobs can be co-located on a piece of hardware would be valuable for the service provider. Sim-

ilar benefits can be achieved in a distributed computation framework like Spark[31] and MapReduce[7] which distribute parallel computations between hardware in a datacentre.

Previous efforts towards solving this problem rely on complex instrumentation, such as hardware counters to count cache misses, to predict whether they will interfere when co-located on the same hardware. Some solutions propose modifying the virtual memory system of the operating system[26]. Others propose changing hardware to split the processor cache between several processes[5][23]. Both of these changes would be very difficult for a service provider to implement in a data centre. In contrast to these complex solutions, the one we propose requires only observations of interference between pairs of processes to make predictions about which would interfere. Rather than using complex instrumentation to detect interference, we used an easy-to-collect quality of service measure to detect interference: the increase of execution time of processes when co-located over when running with no other processes on the hardware. To the best of our knowledge, nothing like this method has been attempted.

We modeled the processes as nodes on a graph and we added edges to an unweighted graph if the co-located processes' runtime increase was over a threshold. We added the inverse of this increase as the weight of an edge in an unweighted graph. For several levels of the interference threshold, we achieved good prediction accuracy.

This algorithm can be used as a flexible framework for modeling interference. For example, encoding a combination of slowdown and increase in Last-Level Cache (LLC) misses as an edge on the graph may also potentially be effective as a predictor of interference. This should be viewed as a tradeoff between prediction accuracy and simplicity of gathering the data needed to build the graph. The novel use of this graph abstraction for process interference allows the use of graph algorithms to solve other problems. For example Xie and Loh consider clustering of processes [29]. A minimum cut of the graph is a clustering of the processes.

We first present the partially-observed graph abstraction, showing the features we extracted from the weighted and unweighted graphs. We also show how this problem relates to matrix completion. Next we describe the application of the this abstraction to the authorship collaboration network and show the prediction results

and scatter plots of the features to show correlations and predictive power of the features. Finally, we apply the graph abstraction to process interference prediction. We show how we generated the data set and the resulting graphs and scatter plots of the features.

Chapter 2

Inference on Partially Observed Graphs

A graph is described by a set of nodes and a set of edges, each of which connect two nodes in the node set. This abstraction can be used to model interactions between pairs of objects, denoted by the existence of an edge between them. If a real-numbered weigh is assigned to an edge, then the edges can also encode the degree that an interaction occurs. The former is referred to as a unweighted graph, while the latter is referred to as a weighted graph.

A partially observed graph is a graph where we assume that we know whether edges exist or not between only a subset of the pairs of nodes in the graph. Based on this information we extract some topological features of the graph and attempt to predict whether edges exist or not between pairs of edges that we have not observed. This is also known in the literature, especially in the context of social network graphs, as link prediction[18][8][16]. It is also closely related to the problem of matrix completion.

This problem can be approached as a two-class classification problem, where pairs of nodes are either classified as having an edge between them or not based on topographical features of the graph. There are several satisfactory surveys on this subject, including [9], and [18], however, to the best of our knowledge, it is a novel contribution of this work to propose extracting path features from a weighted graph.

2.1 Feature Selection

We constructed two graphs to represent the problems of interest. The first is an unweighted graph which encodes the only the interactions between the pairs of object. We also constructed a weighted graph with positive real-numbered weights assigned to each edge. The edge weights were assigned in a method specific to the application. From these graphs, we extracted several computationally inexpensive path and neghbourhood features commonly used for link prediction.

2.1.1 Neighbourhood Features

In the following descriptions, let $\Gamma(n)$ denote the neghbourhood of node n, which is the set of nodes that it is directly connected to with an edge.

Number of common neighbors For nodes n and m:

$$|\Gamma(n) \cap \Gamma(m)|$$

Jaccard similarity of neighbors For nodes n and m:

$$\frac{|\Gamma(n) \cap \Gamma(m)|}{|\Gamma(n) \cup \Gamma(m)|}$$

Number of common neighbors of neighbors For nodes n and m:

$$\left| \bigcup_{x \in \Gamma(n)} \Gamma(x) \cap \bigcup_{y \in \Gamma(m)} \Gamma(y) \right|$$

Adamic/Adar For nodes n and m:

$$\sum_{z \in \Gamma(m) \cap \Gamma(n)} \frac{1}{\log |\Gamma(z)|}$$

The Adamic/Adar measure from the above list warrants some explanation. It was proposed By Adamic and Adar in [1] as a computationally cheaper alternative to the PageRank algorithm[21]. This is similar to common neighborhood, but weights

common neighbors with a smaller degree more highly. The intuition behind this is that a less popular common neighbour may carry more information than a popular one.

2.1.2 Path Features

Path features are also used in literature to solve the link prediction problem, although they are more computationally intensive. Shortest path on an unweighted graph is commonly used[18][8]. However, to the best of our knowledge using shortest path on a weighted graph is had not been considered. As described in Section 3.1 and Section 4.2 this method lends itself well to both of our applications. In the cases that there was no path between two nodes, we set the shortest path to 1,000,000,000, which is much larger than any path in the graph.

A family of more computationally intensive set of path features that have been considered in literature are PageRank[21], SimRank[12], and Katz centrality[14]. From these methods we chose to consider Katz centrality because it had an exact, non-iterative algebraic formulation. Katz centrality considers all paths of a given length between a pair of nodes and assigns more weights to shorter paths by multiplying them with a decay coefficient β. The Katz centrality is computed as

$$\sum_{l=1}^{\infty} \beta^l \#(\text{paths with length } l \text{ between nodes}).$$

Alternatively this can be computed by inverting the adjacency matrix A of the graph $(I-\beta A)^{-1}-I$, giving the Katz centrality of all pairs of nodes. Since the adjacency matrix of our biggest graph was just on the border of being invertible on a powerful PC, we chose to implement this matrix inversion method, setting $\beta = 0.01$ to give a well-conditioned matrix.

2.2 Statistical Analysis

The prediction of edges emerging in a graph is a two-category classification problem: determining, given a set of graph characteristics, whether an edge will exist between a pair of nodes or not. A common approach for evaluating models in machine learning is to withhold a part of the data set for testing, train the model on the

remaining data, then evaluate the predictive performance of the model on the withheld data. To evaluate our models, we used 10-fold cross-validation. We divided the set of node pairs randomly into ten approximately equally-sized sets. We used each set once as a test set while training the algorithm on the remaing sets. For each training set, we build both the unweighted and the weighted graphs from only those pairs in the set, then computed all the features for these observed graphs.

We evaluated several parametric and non-parametric models for predicting the emergence of edges in the graph. Simple linear models like Linear Discriminant Analysis and Generalized Linear Models do not work in this application because some of the features we considered are highly, for example the Adamic/Adar score and the number of common neighbours, are highly correlated. This causes the matrices that these algorithms would have to invert are very ill-conditioned because the vectors representing two highly correlated features are nearly co-linear. A parametric model that we applied to our problem is Stochastic Gradient Descent (SGD), which is not affected by these colinearity problems because it uses gradients rather than matrix inversion. Non-parametric models that we considered are K-Nearest Neighbours (KNN), Random Forests, Boosted Stumps, and Supported Vector Machiness (SVMS). For all the classifiers, we report the Classification Error Rate (CER) averaged over all the folds of cross-validation.

We used the implementations of the machine learning models given in the scikit-learn Python library[22].

2.3 Relationship to Matrix Completion

If one considers the representation of a graph as a symmetric adjacency matrix, then predicting the existence of edges based on observations of some edges is a matrix completion problem. This problem appears many fields including recommendation systems and signal processing or sparse sensing. The problem formulation, described most clearly by Kalofolias at al.[13], is given a matrix $M \in \mathbb{R}^{n \times m}$ which is assumed to be low-rank and a set Ω of observations of entries of M such that

$$M_{i,j} : (i,j) \in \Omega \subseteq \{1,...,n\} \times \{1,...,m\},$$

find

$$\min_{X \in \mathbb{R}^{n \times m}} \text{rank}(X) \text{ subject to } X_{i,j} = M_{i,j} \forall (i,j) \in \Omega. \tag{2.1}$$

The problem in Equation 2.1 is NP-hard, so the minimization performed in practice [3][24] is to instead minimize

$$\min_{X \in \mathbb{R}^{n \times m}} \text{tr}((XX^T)^{1/2}) \text{ subject to } X_{i,j} = M_{i,j} \forall (i,j) \in \Omega \tag{2.2}$$

which is convex, but not smooth. The advantage of this approach is that efficient algorithms for solving these optimizations exists and [3] gives a bound for the number of observations needed to reconstruct the matrix and [24] slightly improves on these bounds.

In the recommender systems application, the matrix M has m rows that represent users and n rows that represent items, for example items in a online store. The entry of $M_{i,j}$ represents user i's rating of item j, whether explicitly observed or inferred from the matrix completion. The recommended system completes this matrix in the hope that it can then recommend users items that they will rate highly. The low-rank assumption of the underlying matrix is justified by the assumption that there exist similar users producing similar rows in M and similar items, producing similar columns in M.

If we force M to be square and symmetric, we can adapt this method to inferring the adjacency matrix of a graph. This has not been done, to the best of our knowledge, likely because the low-rank assumption of the adjacency matrix does not apply to general graphs. However, in both of the applications considered below, our assumption on the outset is that the processes or authors behave similarly meaning the underlying graph is structured into some clusters and suggesting that the adjacency matrix may be low-rank. Király and Tomioka[15] also give necessary and sufficient conditions for reconstructing matrices of arbitrary rank. Applying these methods to reconstructing the adjacency matrix of a graph with some structure would be an interesting direction for future work.

Chapter 3

Application to Authorship Identification

We first applied the partially observed graph abstraction to the graph formed by authors collaborating on publications in an online publication repository. This was some preliminary work towards the problem of disambiguating names of authors that share names or initials. Applying the link prediction to disambiguation of names was preformed by a colleague and is orthogonal to this work, but the link prediction algorithm we developed is extremely effective and is presented below. An intuitive solution to author name disambiguation is to use an author's writing style as proposed by De Vel et al.[6] and Stamatatos[25]. However these methods don't apply when a document is written by several authors, also some given authors may not even have written any part of the document while still having a legitimate contribution to the publication. Using link prediction for this task is attractive because it can determine whether an author collaborated on a publication while only considering past collaborations.

There has been ample work in the field of link prediction in social networks, see [8], [16], and [18] for comprehensive surveys of the field. Because these techniques have only been used to predict links emerging in social and other networks, these authors have not considered a notion of recurring links. In our application, however, recurring collaborations are also of interest. We encode multiple collaborations as parallel edges on a graph and show that the same link prediction

methodology can be used to predict recurring interactions with very good accuracy.

3.1 Graph Construction

We built a graph from a snapshot of the arXiv high energy physics publications database. This was used for a data mining competition in 2003 and is freely available from Cornell University. From the file containing the abstracts of all the papers, we extracted the names of the authors that collaborated on each paper. We stripped all the accents and other special characters from the names and treated all authors with the same surnames and initials as the same person. This approximation was necessary because there was no automatic way of resolving this ambiguity and manual resolution was prohibitively difficult. In practice, few of these cases occurred. We also ignored papers in the database with a single author, since they were irrelevant to our analysis. After completing this process, our database contained 9066 authors, forming 17390 collaborating pairs.

We constructed both an unweighted graph, where we added an edge if two authors had collaborated at any point, and a weighted graph, which encoded how many times a pair of authors had collaborated. Reasoning that more frequent collaboration between authors signaled a closer relationship, we modeled recurring collaborations as parallel unit resistances in an electric circuit, each new collaboration lowering the edge weight. The final edge weight was the inverse of the number of collaborations. This means that shortest paths among frequently-collaborating groups have smaller weights. Note that a pair of authors may have collaborated in the test and training sets, in this case the training graphs will have edges between these two authors and the test of the algorithm will attempt to predict whether this pair of authors collaborated in the test set. This differs from the link prediction problem on social networks, which doesn't as whether a pair of people on the network that are known to have a link will link again. The features were calculated with these edges included and the prediction algorithm was queried for pairs that already had edges between them. It is a novel contribution of this work to show that predictions can be made accurately in this scenario.

3.2 Results

We varied the parameters of all the classification algorithms under consideration and present the best results, except in the case of KNN, where we present results for both 10 and 50 neighbours, since both performed very well. The CER for all the classifiers considered are given in Table 3.1.

Tree-based classifiers: Random forests and boosted stumps were the worst-performing classifiers. Changing the number of trees and the depth of trees used did not dramatically improve performance. An interesting observation is that in every cross-validation fold, the error of boosted trees and random forests was identical. They likely made errors on the same data points.

SVM: These classifiers showed consistently mediocre performance in all tests.

SGD: This algorithm showed very good classification error rate in many cross validation tests, reaching as low as 4%. However, as sometimes happens with this classifier, convergence problems in a few test led to 25% error rate.

KNN: This was by far the best classifier, using a single nearest neighbor produced a classification error rate of 4%. Increasing the number of neighbor lowered the classification error rate until to plateaued at 50 neighbors.

Classifier	SVM	SGD	Random Forest	Boosted Stumps	KNN 10	KNN 50
CER	0.09289	0.07244	0.1031	0.1031	0.04927	**0.02975**

Table 3.1: Average CER over several runs of cross-validation for all classifiers.

3.2.1 Feature Distribution

Analyzing the distribution of features of the pairs of collaborating authors in the data set shows which features were most important for predicting new collaborations. The scatter plot in Figure 3.1 shows the distribution of Adamic/Adar score and number of common neighbours. These two features are highly correlated. The

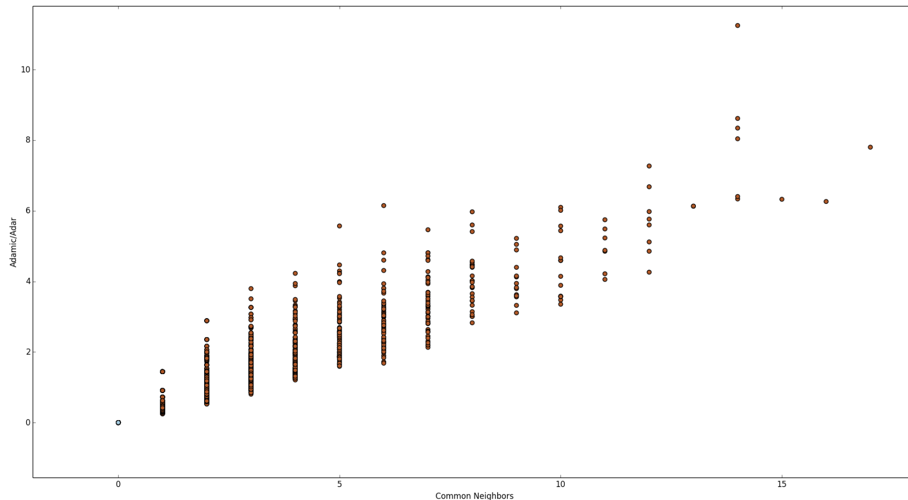

Figure 3.1: Scatter plot of the Adamic/Adar score and number of common neighbours for the authorship collaboration graph. Red points are collaborating pairs and blue points are non-collaborating pairs.

common neighbours of neighbours and the Jaccard similarity of the neighbours are also correlated to these features. This suggests that eliminating all but one of these features from the model would not affect its performance. It is also a property of this data set that overwhelmingly nodes without common neighbours did not collaborate. This is an important feature of the data set. It shows that is made up of densely-connected communities, this is the main reason that prediction is so accurate for this data set.

The Katz score is a very effective classifier. As can be seen in Figure 3.2, exclusively pairs with a low Katz score did not collaborate in the test set. This should be weighed against the computational complexity of inverting the adjacency matrix of the graph, which is required for calculating the Katz score. Figure 3.2 shows a strange bimodal distribution of Katz scores. This it probably another quirk of the data set, perhaps happening because of some densely-connected cliques. The Katz score is also surprisingly uncorrelated to the weighted shortest path between nodes.

Figure 3.3 shows the Scatterplot of weighted and unweighted shortest paths

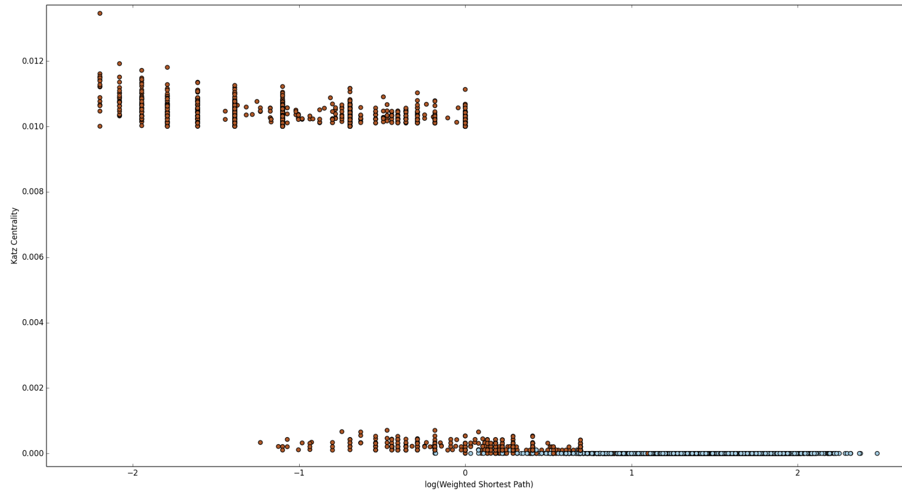

Figure 3.2: Scatterplot of Katz score and weighted shortest path for the data set. Red points are collaborating pairs and blue points are non-collaborating pairs.

between nodes. These features are unsurprisingly correlated, but there is a clear trend that among pairs with the same unweighted shortest paths, ones with smaller weighted shortest paths are more likely to collaborate, justifying the novel use of unweighted shortest path as a feature.

3.3 Conclusion

In this chapter we adapted link prediction on social networks to predict new and recurring collaborations between authors in a snapshot of the arXiv high-energy physics publication database. It is a novel contribution of this work to show that the standard link prediction machinery can be used with no modifications to predict recurring interactions between agents in a network. We encoded publications as edges of an unweighted graph and used standard link prediction methods to extract features from this graph. It is also a novel contribution of this work to encode repeated publications as decreasing edge weight in a weighted graph and use weighted shortest path as a feature to predict edges. We show that weighted and unweighted shortest path are only weakly correlated and that the addition of

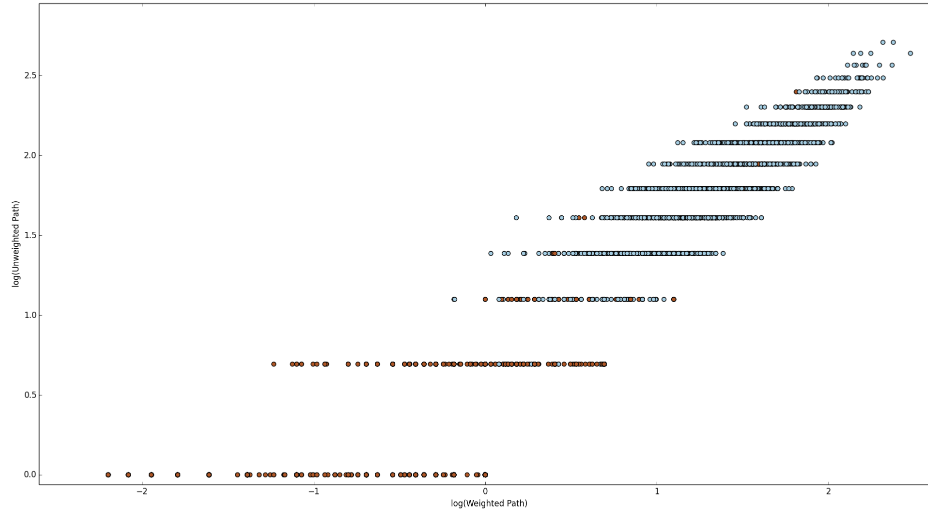

Figure 3.3: Scatterplot of unweighted and weighted shortest path for the data set. Red points are collaborating pairs and blue points are non-collaborating pairs.

weighted shortest path is justified for prediction accuracy.

We assessed several machine learning algorithms and determined that the relatively computationally inexpensive KNN is most efficient for predicting new interactions on this graph. Prediction error was less than 3%. Future work includes finding better features and optimizing machine learning algorithms.

Such accurate link prediction can be used to disambiguate author names in an online without mining the text of a publication. Our future work is combine link prediction and textual features to build a complete solution to disambiguating author names in a publication database.

Chapter 4

Application to Process Interference

Recent years have seen an increase in computation performed in large data centres where heterogeneous computing jobs are scheduled on networked or otherwise interconnected pieces of hardware, referred to as nodes. One application of this system is distributed computation frameworks. For example many big data applications like Spark [31], Microsoft's Dryad [11], and Google's MapReduce [7] seek to distribute parallel workloads over many computing nodes in a datacenter. Another family of example are systems like Amazon's AWS and Microsoft's Azure which give users access to virtual machines which run on nodes in a data centre. Finally, Amazon's Lambda Serverless Compute architecture would be a good application of this framework. It gives users a service which abstracts away the server or container in which an application is running. The user simply provides a set of callback that is triggerd by a set of events. Amazon automatically handles placing the jobs that implement these callbacks on compute nodes in its datacentres.

It is tempting for service providers to schedule several independent jobs on one node, since this would allow them to increase the number of jobs that can run in the same data centre. However, scheduling several jobs on one node causes these nodes to compete for resources. The jobs can compete for resources like network, memory capacity, and hard drive I/O, all which are common to all processors and cores on a node. In a multicore processor, even when all the threads required

by the jobs can be placed on a processor simultaneously, there are still several shared resources on the chip like LLC, memory bus, and pre-fetching hardware. Scheduling applications on the same node that compete for resources can cause the performance of these applications to degrade to an unacceptable level. Because of the variety of resource requirements of different jobs predicting which jobs can be co-located is not trivial, even with advanced profiling of jobs, because performance of applications can decrease due to a complex interplay of resource contentions.

In this application we develop a completely novel algorithm that predicts slow-down of co-located jobs using a graph model and the techniques developed above. Our algorithm defines degradation as an increase in the execution time and encodes it as an edge in a graph. This allows us to predict which jobs we can schedule on the same node without modifying the operating system of the node or the users' jobs to gather any profiling data. The algorithm can also be directly extended to encode any performance degradation measure as an edge in a graph. The strength of our algorithm is that it treats the user applications as a black box. A service provider looking to use this method does not have to implement complex hardware support for monitoring system events or complex resource management algorithms at an operating system level to attempt to reduce resource contention. The service provider would also not need to modify the users' application to gather profiling data.

4.1 Related Work

It is common in previous literature to consider contention on the LLC as the only measure of contention. See [23] for an example where two applications are said to interfere if any of them experience more LLC misses compared to when they are not co-located. Note, however, that performance of an application can degrade for more reasons than LLC contention. Note also that increase in LLC misses beyond a threshold can also be used as the criterion for adding a edge into our graph and all the same machinery can be used. Other researchers have attempted to classify applications using schemes like Stack Distance Completion [4] and Animal Classes [29]. A partitioning of the interference graph also induces a clustering of the applications.

Previous work on addressing the problem of resource contention in multicore systems has predominantly focused on partitioning the processor caches between the threads that are running on the processors. The most complex solution to this problem is to provide hardware support for partitioning the processor caches [5][23]. This would be very difficult for a service provider to implement because producing custom processors that support novel functionality if extremely difficult and time-consuming. It would also be difficult to change cache partitioning algorithm in hardware, which is problematic because a fixed partitioning may not be optimal for all co-located process pairs.

An alternative to hardware-based cache management is to modify the operating system to manage cache petitioning. The best investigated class of such algorithms is memory page colouring, which attempts to allocate virtual memory pages to physical pages in a reserved section of the cache [33][32][27]. Software-based online cache management algorithms can rely on hardware counters to optimize cache partitioning [26]. This is problematic because it requires modifying the virtual memory system of the operating system, which is extremely difficult[33].

Even if we don't consider the difficulty of implementing cache partitioning methods, they are still not a complete solution of eliminating contention between threads. As noted by Lin et al.[17], cache partitioning methods can shift contention away from the cache to memory bus. Also these methods do not consider other sources of contention that can arise.

The method in the most similar to ours is proposed by Zhuravlev et al.[33], which uses LLC miss rate of an application running by itself on a core to schedule applications that experience high miss rates on separate cores. This is similar to our because it gathers some a priori knowledge and uses it as a feature for predicting interactions between unobserved pairs of applications.

Another interesting approach is proposed by Herdrich et al.[10]. This mechanism simply suggests that to react to performance degradation that may be caused by cache pressure, simply decrease the processor frequency of one of the threads to reduce the cache pressure. This is clearly a suboptimal use of resources in the scenario that we propose.

Another method for addressing this problem is proposed by Mars et al.[19][20]. The bubble up system the that they propose uses static analysis of memory ac-

cess patterns of latency-sensitive applications to attempt to predict how sensitive they would be to co-locating with another, non-latency-sensitive, application. Non-time-critical candidates for pairing are co-located with another application whose sensitivity is known and the amount of degradation they cause are called “bubble scores” and are used to find safe parings. The disadvantage of this method is that it requires profiling of applications to predict which pairs are safe to co-locate. Yang et al.[30] propose a modification to this scheme which implements dynamic measurements of co-location sensitivity. The authors also use use a dynamic method of halting the non-time-critical appication when pressure on the latency-sensitive one increases. This online monitoring and measurement uses hardware counters to detect increased contention for shared resources.

Another method that uses static profiling to compute workload signature of user workloads, then assign them to computational resources is given by Vasić[28]. Like our method, several others[19, 20, 28, 30] also assume that repeated runs of the same applications will generate similar system loads and that reusing known safe pairings can amortize the effort of finding these safe pairings.

To the best of our knowledge, our solution is the only one that treats the users processes as black boxes and simply tries to find similarities based on observed interactions. This avoids the complication of gathering metrics on where contention occurs to make predictions. It also avoids modifications to operating system schedulers and virtual memory systems and even more difficult and costly changes to hardware.

4.2 Graph Construction

We crated the data set for this experiment by collecting several popular benchmark suites, including PARSEC, METIS, the NAS benchmark suite provided by NASA. To represent varied user applications, our list of benchmarks included also included Java, MATLAB, and Python programs. The full list is given in Appendix A. We considered both benchmarks that model modern big data applications such as would commonly run in a distributed environment and more typical benchmarks.

We defined a degradation in performance as follows. Consider applications i and j. Over several executions of application i with no other user process running

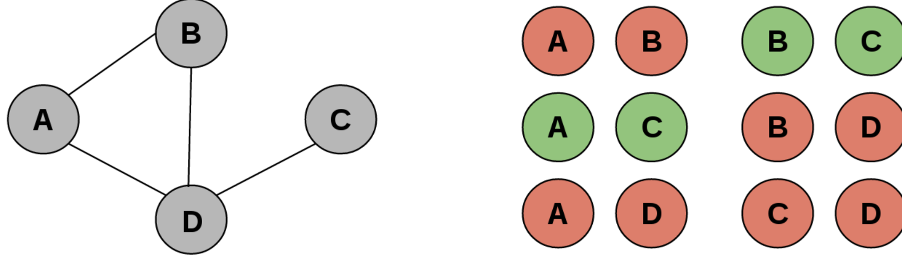

Figure 4.1: Process of creating unweighted contention graph. Red pairs were classified as interfering, while green ones were not.

on the hardware, we measure the average execution time and call it "single-run(i)". Likewise, let "co-run(i,j)" be the average run time of application i over several runs of while application j is also running on the hardware. Note that this is not equal to "co-run(j,i)". We define Slowdown Coefficient (SC) for applications i and j as

$$\text{SC}(i,j) = \frac{\text{co-run}(i,j)}{\text{single-run}(i)} \tag{4.1}$$

and

$$\text{SC}(j,i) = \frac{\text{co-run}(j,i)}{\text{single-run}(j)}.$$

We set a threshold T as a limit to the SCs of two applications beyond which we considered the slowdown to be unacceptable and the processes to interfere when co-located. Varying T caused the density of the graph to vary, we investigated several values of T to simulate various level of time-criticality of the jobs running on the system.

The first graph that we constructed to encode the interaction between processes was an unweighted graph. In this graph we simply added an edge if

$$\max(\text{SC}(i,j), \text{SC}(j,i)) > T. \tag{4.2}$$

To clarify this, consider applications A, B, C, and D. We run all pairs on the same hardware and calculate the SC according to Equation 4.1, we then insert an edge if the condition in Equation 4.2. The resulting graph is shown in Figure 4.1.

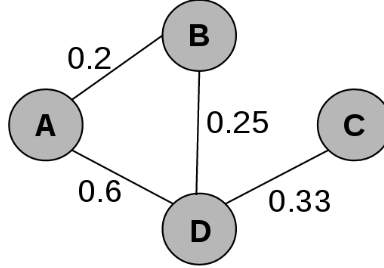

Figure 4.2: Process of creating weighted contention graph. The edge weight is the inverse of the SC.

We also constructed a weighted graph to encode the interference. The weight of an edge between two nodes prepresenting applications i and j is given by

$$w_{ij} = \begin{cases} \frac{1}{\max(\mathrm{SC}(i,j),\mathrm{SC}(j,i))} & \text{if } \max(\mathrm{SC}(i,j),\mathrm{SC}(j,i)) > T, \\ \text{no edge} & \text{otherwise} \end{cases} \tag{4.3}$$

The intuition behind using the inverse of the maximum SC is that there should be a smaller shortest path between processes that interfere with each other if they interfere more. For the same group of application in Figure 4.1, the resulting weighted graph is shown in Figure 4.2. Note that in most cases

$$\mathrm{SC}(i,j) \neq \mathrm{SC}(j,i).$$

This may seem strange, but asymmetrical interference has been noted previously in literature. Xie and Loh[29] observed that the memory access patterns of some applications made them insensitive to being paired, while others showed high sensitivity to pairing. Co-locating these applications perturbs only one application in the pair.

Our data set had 43 applications, which gives 903 possible pairs. All tests were performed on a system with an Intel Core 2 Duo running at 2.33 GHz and 2 GB of RAM. The the machine was running a Fedora 23 Linux operating system. The application were all single-threaded, meaning that there were always enough cores available to run both applications. This is a simulation of running serveral

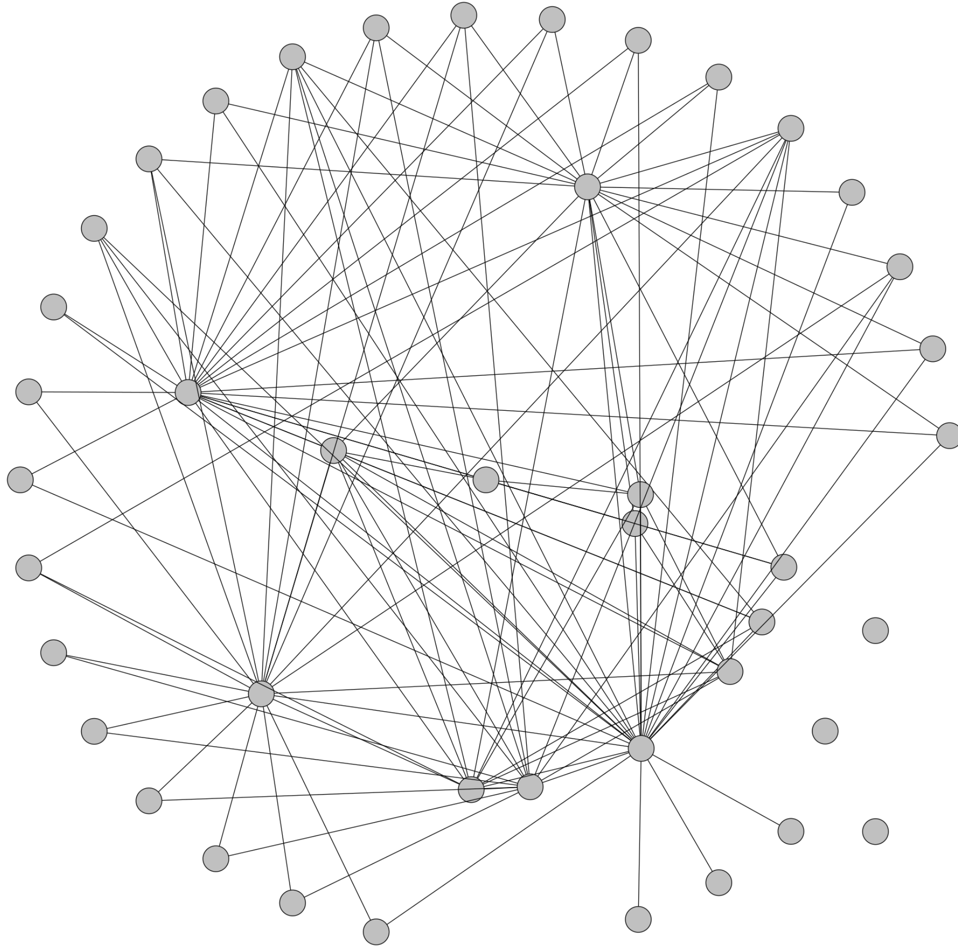

Figure 4.3: The interference graph generated when setting T=2.5.

containers in which user applications run on one physical compute node and having the sum of the cores reported by each container be the same as the number of cores on the compute node.

The interference graph generated with $T = 2.5$ is shown in Figure 4.3. As we can see, several applications interfere with almost all others, while many applications only interfere with a few and can easily be co-located.

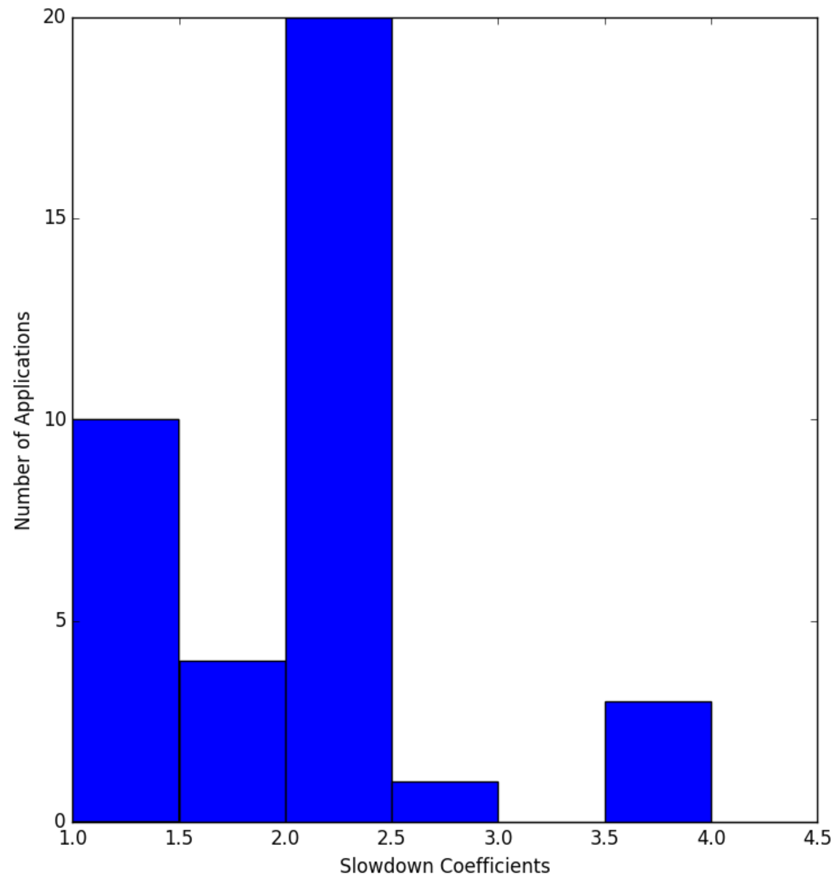

Figure 4.4: Slowdown Coefficients distribution of a typical application.

4.3 Results

After co-locating and running all the pairs of applications, we first wanted to see the distribution of SCS. We chose a typical application, and plotted a histogram of its SCS when co-located with all other applications in the data set in Figure 4.4. For ease of exposition, we binned these slowdown coefficients into intervals of size 0.5.

Studying many such histograms showed us that the SC values lay mostly between one and five. To simulate many real-world situations in which different levels of slowdown may be acceptable, we tested our machine learning algorithm on graphs created with the SC threshold T set to 1.5, 2, 2.5, and 3. Analyzing more extreme values were not interesting because in the case of $T < 1.5$ nearly all edges were present in the graph, while with $T > 3$ would result in a graph with nearly no edges. In both of these cases, accurate prediction is trivially simple since in the case of a very sparse graph a predictor that always predicts no edge can perform very well. Similarly, in a very dense graph, a predictor that always predicts an edge performs very well. Where our algorithm performs better than these predictors it demonstrates that it is making useful predictions. As in the result in Section 3.2, KNN was the best classifier. The results for all the values of T considered and the number of edges that appeared in the graph are given in Table 4.1. As we can see, the algorithm gives statistically significant predictions at $T = 2$ and $T = 2.5$. Table 4.1 also shows that after observing 75% of the pairs the model does not perform better that the trivial classifier. After observing 90%, however, it performs better.

SC Threshold	Number of Edges in Graph	Probability of Edge	Prediction Error Rate 75%	Prediction Error Rate 90%
1.5	694	0.76	0.75	0.36
2	327	0.36	0.45	**0.24**
2.5	118	0.14	0.24	**0.12**
3	88	0.097	0.20	**0.094**

Table 4.1: Density of graph and CER for several choices of SC threshold T. CER is the fraction of correct predictions over the test set.

4.3.1 Feature Distribution

As above, we can examine feature distributions to show which features were best for predicting which processes will interfere when co-located. Figure 4.5 shows the distribution of common neighbours and a Adamic/Adar score with red dots being pairs that interfered and blue dots being pairs that did not. As in the collaboration network in Section 3.2, the two features were highly correlated. An interesting in-

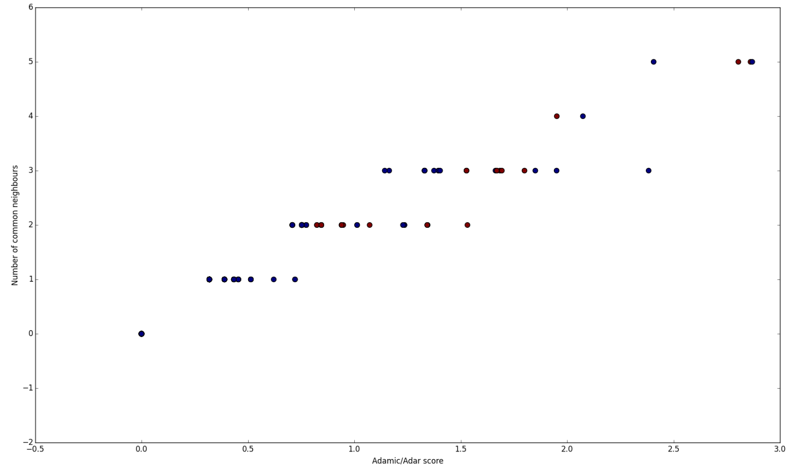

Figure 4.5: Scatter plot of the Adamic/Adar score and number of common neighbours for the process interference graph. Red points are pairs that interfered and blue points are pairs that did not.

sight is that neither feature was a good predictor of interference unlike in the author collaboration application where very few author pairs with common neighbours did not go on to collaborate, compare Figure 3.1. The Katz score and unweighted path also behaved as above.

In this application unweighted shortest path, which is a novel contribution of this work, was a powerful classifier; more so than in the authorship collaboration application. The distribution of unweighted and weighted shortest paths for the graphs are shown in Figure 4.6. The pairs towards the left the image are more likely to interfere than the nodes towards the bottom. This justifies the novel use of the weighted shortest path as a predictor of interference between processes.

4.4 Conclusion

This work proposes a novel application of partially-observed graphs to modeling interference between processes running on the same physical hardware. We ran a collection of benchmarks on a computer singly and in pairs and modeled interfer-

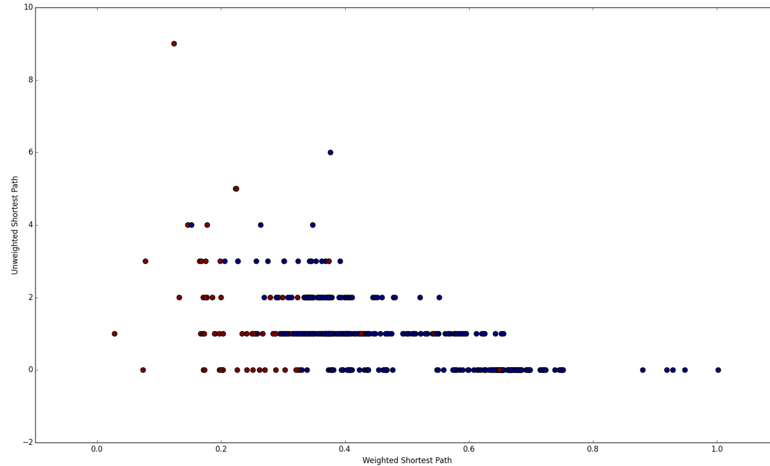

Figure 4.6: Scatter plot of the weighted and unweighted shortest path for the process interference graph. Red points are pairs that interfered and blue points are pairs that did not.

ence in pairs of processes as a large increase in the execution time of the process when co-located. We used a machine learning algorithm to predict which edges would appear in the interference graph, with promising results.

This method should be considered a framework for predicting interference between processes since the abstraction is general enough to encode any pairwise interaction of processes as a graph, then use the machine learning algorithms to predict the outcome of new interactions. For example increased LLC misses or even a combination of LLC miss and execution time increases can be encoded as an edge in the graph. This is a tradeoff between ease of implementing the required instrumentation to detect contention and a potential increase in the accuracy of predictions. Further research remains for finding good combinations of measurements.

Using the interference graph as an abstraction also enables us to use graph algorithms to interpret the observed interference data. For example, partitioning the interference graph by using a minimum cutting algorithm induces a clustering

of the applications into groups that interfere with each other. Bansal, Blum, and Chawla [2] also propose a method of clustering items on a graph by partitions that maximize similarity of items that inside the partitions. This would be a natural fit for our graph abstraction of process interference. Likewise colourings of the graph indicate the number of compute nodes required to simultaneously schedule a set of processes with no interference.

A natural direction for future work would be to attempt to determine an algorithm for running applications against other to determine a co-location scheme with minimal disruption, since users may not accept a datacenter running their applications with interference to determine an optimal scheme. Intuitively a hierarchical clustering using a series of graph cuts may be a direction to explore since exploring these clusters as a tree may yield a good co-location scheme in few tests. If the matrix completion methods discussed in Section 2.3 are effective for this application the accuracy of predictions could be bounded by the number of observations.

To the best of our knowledge, ours is the first method for predicting interference that does not require modifications to the operating system scheduler, virtual memory system, or hardware. Our method also does not require profiling of user applications. This makes it easier to implement in an existing system and it would make increasingly better predictions about which applications to co-locate.

Chapter 5

Conclusion

In this work we applied the partially-observed graph abstraction commonly used in link prediction on social networks to two novel applications with good results. For both applications we encoded interactions between objects as edges in weighted and unweighted graphs. We extracted some topographical features from these graphs and used them for a two-class classification problem in which we attempted to predict whether unobserved pairs of nodes would interact. We used 10-fold cross-validation to evaluate several model for this prediction task and found that KNN was the best classifier.

The application is predicting whether authors in a online database would co-author papers together. We modeled co-authorships as edges in a graph and predicted whether unobserved pairs of authors would interact. We achieved prediction error rate of less than 3%. Our study showed that the neighbourhood features are highly correlated and all but one can be removed without significantly changing prediction accuracy. The Katz score is a very powerful predictor, but is computationally very expensive. The novel contribution of using weighted shortest path is justified for this application. When the unweighted shortest path between pairs of authors is the same, pairs that have a smaller weighted shortest path between them are more likely to collaborate. The graph in this application is very sparse and highly structured, making prediction effective.

The next application is predicting whether two processes running on the same piece of hardware will interfere with each other by competing for shared resources.

We modeled slowdown in the processes when co-located on the same hardware as edges in a graph. The resulting predictions accuracy was also promising. This model is flexible enough to encode other measures of slowdown as edges on the graph, for example a combination of increased LLC misses and increased execution time. Modeling the interactions between processes as a graph also allows us to use graph algorithms for processing this data. For example, a minimum cut of this graph is a clustering of processes that interfere. Unlike all previous methods for predicting process inter-process interference, our method does not require complex modifications to the operating system or the hardware to implement. This means that it can be easily implemented in a datacenter without major modifications.

The prediction for the authorship collaboration was very effective because the graph is made up of many densely connected communities. This means that the underlying graph is sparse and highly structured. The process interference graph, does not display this structure, which makes prediction more difficult. This can also be seen in the feature scatter plots in Section 4.3, which are not as neatly divided between conflicting and non-conflicting pairs as in Section 3.2. Making a much larger process interference graph and varying the interference threshold to investigate if this graph exhibits similar structure to the authorship collaboration graph may reveal whether prediction accuracy can be improved.

Bibliography

[1] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003. → pages 6

[2] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004. → pages 27

[3] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009. → pages 9

[4] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *11th International Symposium on High-Performance Computer Architecture*, pages 340–351. IEEE, 2005. → pages 17

[5] S. Cho and L. Jin. Managing distributed, shared l2 caches through os-level page allocation. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 455–468. IEEE Computer Society, 2006. → pages 3, 18

[6] O. De Vel, A. Anderson, M. Corney, and G. Mohay. Mining e-mail content for author identification forensics. *ACM Sigmod Record*, 30(4):55–64, 2001. → pages 10

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. → pages 3, 16

[8] M. A. Hasan and M. J. Zaki. A survey of link prediction in social networks. In C. C. Aggarwal, editor, *Social Network Data Analytics*, pages 243–275. Springer US, 2011. ISBN 978-1-4419-8461-6. doi:10.1007/978-1-4419-8462-3_9. URL http://dx.doi.org/10.1007/978-1-4419-8462-3_9. → pages 1, 5, 7, 10

[9] M. A. Hasan, V. Chaoji, S. Salem, and M. Zaki. Link prediction using supervised learning. In *In Proc. of SDM 06 workshop on Link Analysis, Counterterrorism and Security*, 2006. → pages 5

[10] A. Herdrich, R. Illikkal, R. Iyer, D. Newell, V. Chadha, and J. Moses. Rate-based qos techniques for cache/memory in cmp platforms. In *Proceedings of the 23rd international conference on Supercomputing*, pages 479–488. ACM, 2009. → pages 18

[11] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007. → pages 16

[12] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543. ACM, 2002. → pages 7

[13] V. Kalofolias, X. Bresson, M. M. Bronstein, and P. Vandergheynst. Matrix completion on graphs. *CoRR*, abs/1408.1717, 2014. URL http://arxiv.org/abs/1408.1717. → pages 2, 8

[14] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953. ISSN 0033-3123. doi:10.1007/BF02289026. URL http://dx.doi.org/10.1007/BF02289026. → pages 7

[15] F. Király and R. Tomioka. A combinatorial algebraic approach for the identifiability of low-rank matrix completion. *arXiv preprint arXiv:1206.6470*, 2012. → pages 9

[16] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007. → pages 1, 5, 10

[17] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 367–378. IEEE, 2008. → pages 18

[18] L. L and T. Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150 – 1170, 2011. ISSN 0378-4371. doi:http://dx.doi.org/10.1016/j.physa.2010.11.027. URL http://www.sciencedirect.com/science/article/pii/S037843711000991X. → pages 1, 5, 7, 10

[19] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, pages 248–259. ACM, 2011. → pages 18, 19

[20] J. Mars, L. Tang, K. Skadron, M. L. Soffa, and R. Hundt. Increasing utilization in modern warehouse-scale computers using bubble-up. *IEEE Micro*, 32(3):88–99, 2012. → pages 18, 19

[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999. → pages 6, 7

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011. → pages 8

[23] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 423–432. IEEE Computer Society, 2006. → pages 3, 17, 18

[24] B. Recht. A simpler approach to matrix completion. *Journal of Machine Learning Research*, 12(Dec):3413–3430, 2011. → pages 9

[25] E. Stamatatos. A survey of modern authorship attribution methods. *Journal of the American Society for information Science and Technology*, 60(3): 538–556, 2009. → pages 10

[26] G. E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pages 117–128. IEEE, 2002. → pages 3, 18

[27] G. Taylor, P. Davies, and M. Farmwald. The tlb slicea low-cost high-speed address translation mechanism. *ACM SIGARCH Computer Architecture News*, 18(2SI):355–363, 1990. → pages 18

[28] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini. Dejavu: accelerating resource allocation in virtualized environments. In *ACM SIGARCH computer architecture news*, volume 40, pages 423–436. ACM, 2012. → pages 19

[29] Y. Xie and G. Loh. Dynamic classification of program memory behaviors in cmps. In *the 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects*. Citeseer, 2008. → pages 3, 17, 21

[30] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 607–618. ACM, 2013. → pages 19

[31] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010. → pages 3, 16

[32] X. Zhang, S. Dwarkadas, and K. Shen. Towards practical page coloring-based multicore cache management. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 89–102. ACM, 2009. → pages 18

[33] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *ACM Sigplan Notices*, volume 45, pages 129–142. ACM, 2010. → pages 18

Appendix A

List of Benchmarks

The benchmarks suites used in the experiments in Chapter 4 can be found at:

- METIS https://pdos.csail.mit.edu/archive/metis/
- PARSEC http://parsec.cs.princeton.edu
- NAS Parallel https://www.nas.nasa.gov/publications/npb.html
- NAS OMP https://github.com/wzzhang-HIT/NAS-Parallel-Benchmark/tree/master/NPB3.3-OMP
- NAS Java https://www.nas.nasa.gov/publications/npb.html
- HiBench https://github.com/intel-hadoop/HiBench
- LAPACK http://www.netlib.org/lapack
- Whetstone http://www.roylongbottom.org.uk/whetstone.htm
- Python 3 Benchmarks https://benchmarksgame.alioth.debian.org/u64q/python.html
- Dhrystone https://github.com/Keith-S-Thompson/dhrystone/tree/master/v2.1