NETWORK ANALYSIS AND LINK PREDICTION: EFFECTIVE AND MEANINGFUL MODELING AND EVALUATION

A Dissertation

Submitted to the Graduate School of the University of Notre Dame in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Ryan N. Lichtenwalter

Nitesh V. Chawla, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana

April 2012
Abstract
by
Ryan N. Lichtenwalter

Link prediction is succinctly stated as identifying unobserved links in a network. It has important applications ranging from recommending beneficial relationships in social networks to discovering new protein-protein interactions in biological networks. This dissertation primarily focuses on link prediction in homogeneous, single-relational networks, but it also introduces new techniques for handling multi-relational networks. We identify what we view as the three principal components of effective link prediction: high-performance algorithms and methods, proper evaluation, and computational scalability. Performance is the natural requirement that the predictive models provide usably correct outputs. Evaluation is the surprisingly difficult question of how to determine the correctness of a model in order to compare models and develop an expectation of effectiveness in real deployment. Computational scalability is necessary to obtain output from a link prediction framework in a reasonable amount of time on large networks. Each of these three components poses significant challenges, and it is reasonable to say that any of these challenges individually can stand in the way of the proliferation of link prediction techniques for useful research and domain applications.
Dedication

To the future of humanity...

...may it reach its full and brightest potential.
# CONTENTS

FIGURES .......................................................... vi

TABLES ............................................................ viii

ACKNOWLEDGMENTS ............................................... ix

SYMBOLS ........................................................... xi

CHAPTER 1: INTRODUCTION ..................................... 1
  1.1 Basic Definitions ........................................... 2
  1.2 Background .................................................. 4
  1.3 Contributions ................................................. 6
  1.4 Structure .................................................... 9

CHAPTER 2: NETWORK DATA .................................... 11
  2.1 Descriptions .................................................. 13
  2.2 Saturation .................................................... 16

CHAPTER 3: ALGORITHMS AND METHODS .................... 20
  3.1 Unsupervised Prediction ................................... 20
    3.1.1 Degree and Volume .................................... 21
    3.1.2 Common Neighbors ...................................... 21
    3.1.3 Jaccard’s Coefficient .................................. 23
    3.1.4 Adamic/Adar ............................................. 24
    3.1.5 PageRank ............................................... 24
    3.1.6 Katz ..................................................... 25
    3.1.7 Rooted PageRank ....................................... 26
    3.1.8 SimRank ............................................... 28
  3.2 The PropFlow Method ....................................... 30
  3.3 Supervised Prediction ...................................... 33
    3.3.1 Methodology ............................................ 34
### FIGURES

2.1 The hierarchical organization of the network data set families  
2.2 Network growth and saturation  
3.1 PropFlow computation example.  
3.2 Link prediction methodology  
3.3 Preferential attachment performance by scoring region  
3.4 AUC performance variation in network edge subsampling  
3.5 Class imbalance properties by geodesic distance  
3.6 Predictor ROC curve performance at $\ell = 2$  
3.7 Elements of VCP$^{3,2}$  
3.8 Subgraphs that form VCP$^{4,1}$ and the mapping of isomorphic subgraphs to VCP elements  
3.9 Subgraphs from undirected VCP$^{4,2}$ and directed VCP$^{4,1}$  
3.10 A depiction of the sets considered in the fast VCP$^{4,r}$ algorithm  
3.11 Distributional comparison of extended VCP$^{3,1}(5)$ membership for highly ranked and lowly ranked prediction outputs  
3.12 ROC and precision-recall curves for selected networks  
3.13 Link prediction timing analysis  
3.14 The first 16 elements in VCP$^{3,2}$  
3.15 Supervised classification approaches involving different temporal snapshot distributions  
3.16 Distribution of events by period  
3.17 Performance over varying temporal split locations  
3.18 Scaling trends in distinct values of VCP elements  
4.1 Link predictions for $e_{a,b}$ differ depending on source designation
| 4.2  | Effects of saturation on predictor performance | 115 |
| 4.3  | Performance at $\ell = 2$ as a function of $\tau_x$ | 117 |
| 4.4  | Effect of test set negative sampling on ROC area | 119 |
| 4.5  | ROC curve performance of preferential attachment predictor by distance | 122 |
| 4.6  | Effect of negative instance removal on prediction over increasing distance | 124 |
| 4.7  | Distribution of distances in test sets | 125 |
| 4.8  | Precision-recall curve area performance of preferential attachment predictor by distance | 127 |
| 5.1  | All required DisNet user code for graph diameter | 144 |
| 5.2  | Architectural overview of DisNet | 146 |
| 5.3  | DisNet strong scaling | 152 |
| 5.4  | DisNet performance and master burden with respect to the worker combination parameter | 153 |
| 5.5  | DisNet speedup as a function of problem size | 156 |
| 5.6  | Simplified depiction of some of the build paths in the LPmade automation script | 160 |
| 6.1  | Demographic node attribute link formation model | 171 |
| 6.2  | Usage-based node attribute link formation model | 173 |
| 6.3  | Structural link formation model based on HPLP | 174 |
| 6.4  | Structural link formation model based on directed VCP$^{3,1}$ | 175 |
| 6.5  | Structural link formation model based on undirected VCP$^{4,1}$ | 177 |
| 6.6  | VCP$^{3,2}$ key for the time-resolved multi-relational model | 178 |
| 6.7  | Temporal link formation model based on undirected VCP$^{3,2}$ | 179 |
| 6.8  | Link formation model based on all feature families | 181 |
TABLES

2.1 NETWORK DATA SET CHARACTERISTICS .............................. 12
3.1 PERFORMANCE RANKS FOR LINK PREDICTORS .................. 48
3.2 COUNTS OF ENUMERATED VCP SUBGRAPHS ..................... 55
3.3 CARDINALITY OF $VCP_{i,j}^{n,r}$ ................................. 58
3.4 DISTRIBUTIONAL DIVERGENCE OF VCP ELEMENT PREVALENCE ................................................. 72
3.5 COMPARATIVE AUROC PERFORMANCE OF VCP ............... 75
3.6 COMPARATIVE AUPR PERFORMANCE OF VCP .................... 76
3.7 DEFINED CHARACTERISTICS OF SELECTED LONGITUDINAL NETWORKS .............................................. 90
3.8 AUROC PERFORMANCE FOR LONGITUDINAL VCP PREDICTIONS ......................................................... 95
3.9 AUPR PERFORMANCE FOR LONGITUDINAL VCP PREDICTIONS ......................................................... 96
3.10 CARDINALITY OF $VCP_{n,r}^{n,r}$ FOR $n \leq 4, r \leq 6$ ........ 97
3.11 COMPARISON OF $VCP_{4,r}^{4,r}$ AUROC PERFORMANCE BEFORE AND AFTER FEATURE SELECTION ............... 101
3.12 COMPARISON OF $VCP_{4,r}^{4,r}$ AUPR PERFORMANCE BEFORE AND AFTER FEATURE SELECTION .................. 102
5.1 OPTIMAL DATA STRUCTURE AND REQUIRED SLOC FOR (A) process_vertex AND (B) combine_data ROUTINES .... 142
6.1 NODE ATTRIBUTES IN CASE STUDY ............................. 169
6.2 DEMOGRAPHIC NODE ATTRIBUTES ............................. 170
6.3 USAGE NODE ATTRIBUTES ..............................
ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Professor Nitesh V. Chawla, and his research group for their support. Professor Chawla has always been understanding of personal needs and fervently supports the well-being of his students. Notre Dame is a community that offers constructive criticism and cooperation rather than rivalry and denigration, and the 2007-2012 DIAL research group is a fine example of that. Specifically, I would like to thank Nathan Regola for his friendship, excellent systems administration for the group, and general willingness to help with anything. I would also like to thank Troy Raeder, who has always been happy to apply his intelligence and mathematical intuition to deep, extended theoretical discussions.

Next, I want to thank those with the longest stake in my life and largest part in my development, my family. There is no doubt that family plays an incredible role in forging those aspects of the individual that are most important to success, and I am fortunate to have won the lottery of life in this regard. Thank you all for your boundless love. Thank you, Grandma, for your warmth, patient ear, and the solace you have always provided. Thank you, Mom, for your undying belief in me, sacrifices to raise me, and all of the lessons you have taught me in life. Thank you, Dad, for your silent strength and support, the many opportunities that you provided, and the critical education that you gave me as a child. Thank you, Brother, for challenging me, supporting me, and being a perpetual friend. Like
all families, mine has grown and shrunk throughout the years. To all the new and missing members, thank you for your guidance, teachings, support, and love.

Finally, I want to thank my beautiful wife. Thank you, Katerina, for your intelligence, creativity, patience, boundless compassion, and love. The heights and durations of happiness that you add to my life are unprecedented. Because of you, I have experienced so many wonders that I would never otherwise have encountered. Our relationship is that of which I always dreamed: one perpetual, exciting adventure.
SYMBOLS

\( G \) a graph composed of a set of vertices \( V \), and a set of edges \( E \)
\( V \) the set of vertices in a graph
\( E \) the set of edges in a graph
\( v_i \) an arbitrary vertex in the set \( V \)
\( e_{i,j} \) an arbitrary edge in the set \( E \) connecting vertices \( v_i \) and \( v_j \)
\( \Gamma(v_i) \) the set of neighbors, out-neighbors by default, connected to \( v_i \)
\( \ell \) geodesic distance, usually taken between two vertices
\( c \) the DisNet combination parameter
\( r \) the Pearson product-moment correlation coefficient
\( \rho \) the Spearman rank correlation coefficient
\( \overline{C} \) the arithmetic mean clustering coefficient of a graph
\( r_a \) the assortativity coefficient of a graph
CHAPTER 1

INTRODUCTION

Duncan Watts, one of the catalytic individuals in the explosion of study in network science, writes of the “new” science of networks [113]. He is not referring to the theory and application of graphs, because this has been a topic of study for a long time in many fields ranging from mathematics to sociology. He is instead asserting that there is much new in the study of networks because of increases in computing power and data availability, and this allows for a reexamination of much of what we know from new and different perspectives. Organizations all over the world are developing centers for research in network science including the Center for Complex Networks Research at Northeastern University, the Interdisciplinary Center for Network Science and Applications at the University of Notre Dame, multiple academic research centers within the United States Army Research Laboratory, the Network Science Center at West Point, and the Center for Network Science at Central European University. Fruits of the study of network science have resulted in the construction of new conferences such as NetSci and ASONAM and have permeated highly respected conferences in other subfields of computer science such as KDD, ICDM, and WWW to name only a few. Interdisciplinary efforts spanning computer science, physics, mathematics, biology, sociology, and economics are rampant as new applications emerge. [9] and [21] have even introduced network science to mainstream audiences with subtitles
such as *How Everything Is Connected to Everything Else and What It Means* and *The Surprising Power of Our Social Networks and How They Shape Our Lives* respectively.

As the subtitles of these books imply, the science of networks is really about understanding objects in terms of the high dimensional space defined by the relationships that connect them. These connections affect not only our social dynamics but also the broader dynamics of a variety of physical, biological, infrastructural, economic, and other systems. Whereas much of our understanding of objects in the past was primarily focused on the objects themselves and the properties that they have or develop, network science instead focuses on the links between objects. Without links we have only an annotated collection of items, but links reveal a complex, dynamic system of interactions that offers new sources of information.

1.1 Basic Definitions

Networks can be broadly and generically defined as collections of connected items. A network or graph, \( G = (V, E) \) is composed of a set of \( n \) vertices, nodes, egos, agents, or objects, \( V \) with exemplar \( v_i \), and a set of edges, links, arcs, or relations, \( E \) with exemplar \( e_{i,j} \) connecting \( v_i \) and \( v_j \). Weighted networks may have weights on vertices, indicated as \( w_i \), edges, indicated as \( w_{i,j} \), or both. They may additionally be undirected, in which case \( e_{i,j} \) implies \( e_{j,i} \), or directed, in which case it does not. When placed in dynamic contexts, even this rudimentary definition raises many possible scientific challenges. We would like to understand how these sets are composed and how they change, but the amount of information that can be encoded in a network is tremendous, and the complexity of effectively analyzing and interpreting the entirety of network data can be staggering. One
of the most fundamental aspects in most forms of network evolution relates to changes in the set \( E \), and this comes in the form of elements in \( E \) either appearing or disappearing. In seeking to understand network evolution, we must consider issues like how the properties of items in \( V \) are likely to affect the composition of \( E \) and how the properties of items in \( E \) affect the existence or absence of other items in \( E \). If elements in \( V \) and \( E \) are embedded in space, time, or other continua, we must determine how these embeddings relate to any evolutionary trends that we observe. To understand these things, to discover the real engine underlying network evolution, is to approach the grandest potential of network analysis and network science. We approach this challenge by focusing on the problem of link prediction.

*Link prediction* generally stated is the task of predicting relationships in a network. There are many specific incarnations of the problem and special cases that researchers have attempted to solve. Most effort revolves around the problem of predicting new links given a set of existing nodes and links in a homogeneous, single-relational network. These existing nodes and links may be present from a prior time period, for longitudinal data, or they may be some portion of the topology in a network the exact topology of which is difficult to measure. In the former case, link prediction is useful to anticipate future behavior. In the latter case, it can identify or substantially narrow possibilities that are difficult or expensive to determine through direct experimentation [79, 104]. Thus, even in domains where link prediction can seem impossibly difficult or offer a high ratio of false positives to true positives, it may be useful [24].

We now define link prediction more formally as the task of inferring links in a graph \( G_{t+1} \) based on observation of a graph \( G_t \). It may be that \( t + 1 \) follows \( t \)
in time, or it may be that \( t + 1 \) represents some other evolution or manipulation of the graph such as including additional links from experiments that are difficult or expensive to conduct. Link prediction stated in this manner is a binary classification problem in which links that form construct the positive class and links that do not form construct the negative class. It is possible to apply both unsupervised methods, in which an examination of \( G_t \) alone yields prediction scores, and supervised methods, in which \( G_t \) is itself decomposable into training features and labels.

1.2 Background

Link prediction is fundamentally about understanding the evolutionary processes of these networks. Theoretically it seeks to answer basic questions about how and why objects interact and systems change. Practically it offers abilities such as predicting how a system will grow or react to change, finding important hidden connections, determining useful new relationships, or finding relationships that may indicate a problem or anomaly [73, 96]. As a result, various types of link prediction have a remarkable array of applications. Security agencies can more precisely focus their efforts based on probable relationships in malicious networks that have heretofore gone unobserved [53]. In social networks, this includes recognizing or predicting potentially fruitful collaborations [10, 12, 46, 55, 64, 82]. It may also allow individuals to efficiently and effectively find companions, assistants, or colleagues [52]. In medicine and biology, link prediction can be used to find relationships and associations that exist, but which might otherwise surface only after arduous and expensive research and study on a huge selection of agents. In biology it can help to find new protein-protein interactions [5, 18, 99, 101] or
to determine genes that may be involved in disease [27, 29, 101]. There is a close relationship between link prediction in bipartite networks and collaborative filtering, an application explored briefly in [47]. The authors of [2] used a clustering approach to identify potentially missing links between semantically similar Wikipedia articles.

Another closely related problem is anomalous link discovery [96]. In general, a framework for link prediction can be converted to a framework for link discovery by performing the following procedure. Take the input network and remove from it either one existing link at a time, or a subset of existing links, then generate predictions for the removed links. Those with the lowest score are the links that the system should label as anomalous. There are many exciting applications for link prediction in accomplishing anomalous link discovery, such as security [56], and the principles involved are mostly compatible with those of link prediction. Put simply, any environment that naturally maps to a network probably has an equally coherent mapping from link prediction in that network back to an important question in the environment.

The number of potential applications of link prediction extends well beyond this exposition. As interest has heightened in link prediction, it has become more popular as a research topic. The first comprehensive view of link prediction came in 2003 with [63]. This seminal paper still stands as an excellent reference for those seeking an introduction to the problem. It introduces many of the unsupervised prediction methods we will cover. It also discusses ways potentially to enhance these methods using meta-level approaches. With the passage of time, however, data availability and quantities have vastly increased. In virtually every field of science, at least some problems are now posed and understood in terms of networks.
of entities and relationships. Since the Liben-Nowell study, researchers have drawn
techniques from data mining [4, 87, 88] to construct supervised link prediction
frameworks to conduct complex inference in the wealth of data. Many works in link
prediction focus only on topological features of the network [68, 84]. Others involve
the use of both topological features and node attributes [4, 87, 88]. More recently,
approaches have appeared that involve innovative ways of creating predictions or
innovative ways of applying predictions. For instance, link prediction has been
proposed as a method of collaborative filtering [47]. The standard link prediction
problem has been specialized into several niche areas. One of these involves a
general matrix-alignment approach that can operate over an arbitrary feature
space [100]. Another involves making predictions of new link formations without
first observing the network topology at all [57].

1.3 Contributions

We decompose the problem of effective link prediction into three principal
components: effective algorithms and methods, evaluation, and scalability. Effec-
tiveness of algorithms and methods is the natural requirement that models of link
prediction provide usably correct outputs. Evaluation is the surprisingly difficult
question of how to determine the correctness of a model in order to define an ex-
pectation of the effectiveness of deploying the model or to compare one model to
another. Scalability and computational speed are necessary to obtain the output
of a link prediction framework in a reasonable amount of time on large networks.
Each of these three components pose significant challenges, and it is fair to say
that any of these challenges individually can stand in the way of the proliferation
of link prediction techniques for useful research and domain applications.
In terms of algorithms and methods, we consider separately unsupervised algorithms and more sophisticated supervised classification models. We start by reviewing the unsupervised methods introduced by [63] and describing ways that they can be extended to apply to more types of data. In their study, the authors were able to use exact, computationally intractable methods on very small networks. We update the study by conducting experiments on much larger data sets that require more practical approximations. We introduce our own unsupervised link prediction method, called PropFlow, and demonstrate its advantages in terms of computational time and performance with respect to its peers. Next, we transition to a rigorous examination of supervised classification in link prediction. Prior work has focused on introductory ideas and performance but has left more nuanced issues pertinent to designing effective classification frameworks untouched. The rigorous study results in performance that greatly outperforms the unsupervised baselines. Our final major contribution in algorithms and methods is that of the vertex collocation profile (VCP). This is a new method for link analysis based on the frequency of the embedding of pairs of vertices in subgraph isomorphism classes. We develop the mathematics, algorithms, and application techniques necessary for effectively using this powerful new method for both link analysis and link prediction in single-relational and multi-relational networks. Then we demonstrate how the multi-relational capabilities of VCPs can incorporate information such as time to enhance the predictive capacity of network data even more than with baseline VCP-based predictions.

There are many different ways of evaluating the performance of a link predictor, and there is no established set of standards. In the worst cases, misinformed methods may give misleading or errant results, while at least the fragmentation
of evaluation methods prevents any standard of comparison. Link prediction is a challenging domain within which to evaluate and interpret performance because of its issues with prediction directionality, the variable periods over which network data can be observed to derive different results, and its extreme imbalance to illustrate. The latter issue causes many to apply test set sampling methods, many of which demonstrably lead to problems. All of these issues stand in the way of producing fair, comparable results across published methods. Perhaps even more importantly, they interfere with views of performance that indicate what we can really expect of our prediction methods in deployment scenarios. Some of the analysis and conclusions may seem obvious, but the published literature is rife with the issues we highlight making it difficult to understand results. We seek to provide a reference for important issues to consider and a set of coherent standards as recommendations to those performing link prediction research. We examine some proposed evaluation methods across several papers, occasionally highlighting weaknesses, and propose a standard method informed by class imbalance research within the data mining community. Sound evaluation matters in any domain, and link prediction includes all the issues of proper evaluation in binary classification, but it also presents several unique challenges.

Finally, we introduce DisNet and LPmade, two systems for tackling some of the computational issues involved in network analysis and link prediction in large, complex networks. DisNet automatically distributes vertex-centric computations across computational worker nodes on symmetric multiprocessing systems, grids, and clouds. It is, to our knowledge, the first published system that allows users to specify suitable network analysis algorithms in completely serial fashion while leveraging inherent natural parallelism. DisNet also includes useful features for
long running computations like worker-level result aggregation and checkpointing. LPmade is a system designed around GNU make that makes conducting network analysis and link prediction faster, easier, more efficient, and less error-prone. Many tasks in network analysis and fair, effective link prediction involve a complex series of steps. LPmade describes these steps in a complex dependency hierarchy, which enables the LPmade system to conduct dozens or hundreds of automatically parallelized computations with short, understandable one-line commands.

By covering the algorithms and methods necessary for good prediction output, the issues required for interpretable evaluation output, and integrating all of this into systems that make computations more accessible, we have significantly advanced opportunities for practical and effective application of link prediction. Domain experts unfamiliar with the complexities of data mining, evaluation, distributed systems, and network science can leverage our ideas, methods, and implementations to advance science in their respective fields.

1.4 Structure

In this chapter, we provided the reasons behind the importance of network science, particularly link prediction, and we placed our work in broad context. The following chapters align with the breakdown of link prediction into issues of predictive performance, evaluation, and scalability, and in each we first provide the relevant background information before proceeding to our contributions. In Chapter 2, we introduce and describe the data that we will use throughout the work to compute our results and support our conclusions. As we proceed through more specialized topics, we will highlight additional characteristics of interest for particular, relevant data sets. In Chapter 3, we cover in detail existing unsuper-
vised link prediction methods and introduce several new methods for performing link prediction including the PropFlow algorithm, a high-performance supervised link prediction framework called HPLP, and a novel method of analyzing topology with structural decomposition called VCP. In Chapter 4, we cover the complex issue of evaluation. We introduce some considerations that are heretofore untouched by others in the field. We also propose new concerns related to problematic accepted practices. In Chapter 5, we introduce the problem of computation in large networks. We briefly discuss several approaches to achieve computationally intensive tasks more quickly and then introduce our solution, DisNet, which recognizes the ease of exploiting vertex-independent computations for easy parallel computation including those necessary for link prediction. We also introduce the LPmade framework, which provides a raw network and link prediction library along with an elegant build system designed to lower the barrier to entry for performing link prediction in research and industry domains. Chapter 6 explores the link formation processes of one large, complex network in particular in an attempt to characterize the unique interplay of the various dimensions involved. We conclude and argue for the aggregate benefit of our work in Chapter 7.
CHAPTER 2

NETWORK DATA

In this study, we use a variety of data sets falling into a broad array of categories, visible in Figure 2.1. The categories include communication networks as represented by calls and sms, citation networks represented by dblp-cite, hepth-cite, and patents-cite, scientific collaboration networks represented by condmat, dblp-collab, hepth-collab, and patents-collab, genetic networks represented by disease-g and disease-p, and product co-purchasing networks represented by huddle. While no study of link prediction on any subset of networks can guarantee universally applicable recommendations, the breadth of this study lends substantial credence to the general findings we present. Table 2.1 will be cited several times in upcoming chapters to refer the interested reader to basic properties of specific data sets.

Figure 2.1. The hierarchical organization of the network data set families.
### TABLE 2.1

NETWORK DATA SET CHARACTERISTICS

<table>
<thead>
<tr>
<th>Name</th>
<th>Weighted</th>
<th>Directed</th>
<th>Vertices</th>
<th>Edges</th>
<th>$\bar{C}$</th>
<th>$r_a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>calls</td>
<td>✓</td>
<td>✓</td>
<td>7,786,471</td>
<td>33,292,508</td>
<td>0.127</td>
<td>0.212</td>
</tr>
<tr>
<td>condmat</td>
<td>✓</td>
<td></td>
<td>17,216</td>
<td>110,544</td>
<td>0.642</td>
<td>0.177</td>
</tr>
<tr>
<td>dblp-cite</td>
<td>✓</td>
<td>✓</td>
<td>15,963</td>
<td>344,373</td>
<td>0.128</td>
<td>-0.046</td>
</tr>
<tr>
<td>dblp-collab</td>
<td>✓</td>
<td></td>
<td>367,725</td>
<td>2,088,710</td>
<td>0.617</td>
<td>0.254</td>
</tr>
<tr>
<td>disease-g</td>
<td>✓</td>
<td></td>
<td>399</td>
<td>15,634</td>
<td>0.665</td>
<td>-0.310</td>
</tr>
<tr>
<td>disease-p</td>
<td>✓</td>
<td></td>
<td>437</td>
<td>81,158</td>
<td>0.818</td>
<td>-0.406</td>
</tr>
<tr>
<td>hepth-cite</td>
<td>✓</td>
<td>✓</td>
<td>8,249</td>
<td>335,028</td>
<td>0.352</td>
<td>0.097</td>
</tr>
<tr>
<td>hepth-collab</td>
<td></td>
<td></td>
<td>8,381</td>
<td>40,736</td>
<td>0.466</td>
<td>0.237</td>
</tr>
<tr>
<td>huddle</td>
<td>✓</td>
<td></td>
<td>4,243</td>
<td>997,008</td>
<td>0.591</td>
<td>-0.211</td>
</tr>
<tr>
<td>patents-cite</td>
<td></td>
<td>✓</td>
<td>1,461,714</td>
<td>32,418,457</td>
<td>0.106</td>
<td>0.075</td>
</tr>
<tr>
<td>patents-collab</td>
<td></td>
<td></td>
<td>1,162,227</td>
<td>5,448,168</td>
<td>0.531</td>
<td>0.141</td>
</tr>
<tr>
<td>sms</td>
<td>✓</td>
<td>✓</td>
<td>5,016,746</td>
<td>11,598,843</td>
<td>0.048</td>
<td>0.042</td>
</tr>
</tbody>
</table>

These properties are constructed from all available longitudinal data. $\overline{C}$ represents average clustering coefficient and $r_a$ represents assortativity coefficient.
2.1 Descriptions

calls is a stream of 262 million cellular phone calls from a major cellular phone service provider. We construct weighted, directed networks from the calls by creating a node $v_i$ for each caller and a directed link $e_{i,j}$ from $v_i$ to $v_j$ if and only if $v_i$ calls $v_j$. This data set is not publicly available, but it has been heavily studied and was first presented in [89].

condmat is a stream of 19,464 multi-agent events representing condensed matter physics collaborations from 1995 to 2000. We construct weighted, undirected networks from the collaborations by creating a node for each author in the event and an undirected link connecting each pair of authors. Link weights correspond to the number of collaborations two authors share. This data set was provided by Mark Newman and is a time-resolved version of the data described in [85].

dblp-cite is a citation network based on the DBLP computer science bibliography. Each researcher is a node $v_i$ and weighted directed networks are formed by viewing a citation by researcher $v_i$ of work by researcher $v_j$ as a directed link $e_{i,j}$. Link weights correspond to the number of such citations. The dblp-collab network uses the same raw data, but links are based on co-authorship collaborations. A weighted undirected link exists between $v_i$ and $v_j$ if both are an author on the same paper, but that link has a weight divided equally among the other authors. A paper with 5 authors would create 10 links each of weight 0.25. Total link weights correspond to the sum of individual link weights over a given period. The raw data from which dblp-cite and dblp-collab were constructed is available at http://kdl.cs.umass.edu/data/dblp/dblp-info.html. The authors ask that we acknowledge them thus: the Proximity DBLP database is based on data from the DBLP Computer Science Bibliography with additional preparation performed...
by the Knowledge Discovery Laboratory, University of Massachusetts Amherst.

*disease-g* is a network in which nodes represent diseases and the links between
diseases represent the co-occurrence of particular genotypic characteristics. Links
are undirected but are weighted according to the frequency of the co-occurrence.
Unlike the other networks, this network is not longitudinal. We can treat it in the
same manner by imposing a random order on the appearance of links, creating
periods with the same number of appearances, and using those periods for training
and testing.

*disease-p* is from the same source as *disease-g*. The difference is that rather
than having nodes represent diseases with links representing the co-occurrence of
genotypic characteristics, the links in *disease-p* represent the co-occurrence of
phenotypic characteristics. This network is derived from the same multi-relational
network as *disease-g*. Predictions of common expressions between diseases are
uninteresting since expressions are either observed between diseases or they are
not, so practically speaking the value of phenotypic predictions is negligible.
Nonetheless, holding out phenotypic links and subsequently predicting their presen-
tce is equally instructive for the purposes of predictor evaluation. The *disease-g*
and *disease-p* networks are described in greater detail in [28].

*hepth-cite* and *hepth-collab* are constructed in exactly the same way as
*dblp-cite* and *dblp-collab* respectively. The raw data for these networks is
a set of publications in theoretical high-energy physics. In particular, we used
a data set post-processed by the Knowledge Discovery Lab at the University of
Massachusetts for use in [80] rather than the original 2003 KDD Cup competition
data set. This form of the data set offers numerous advantages in data quality and
entity consolidation and disambiguation. Entity disambiguation is unnecessary
for other data sets that did not come from citation sources because events were recorded in terms of the actual entities themselves. For data sets in which there is some question about the mapping between names or other identifiers and actual entities, entity disambiguation work such as in [11, 77] supports a more meaningful topology and thus better prediction. The raw data from which \texttt{hepth-cite} and \texttt{hepth-collab} were constructed is available at http://kdl.cs.umass.edu/data/hepth/hepth-info.html. The authors ask that we acknowledge them thus: the PROXIMITY HEP-Th database is based on data from the arXiv archive and the Stanford Linear Accelerator Center SPIRES-HEP database provided for the 2003 KDD Cup competition with additional preparation performed by the Knowledge Discovery Laboratory, University of Massachusetts Amherst.

The \texttt{huddle} data set from [94] is transaction data gathered at a convenience store on the University of Notre Dame campus. The data was collected from June 2004 to February 2007. Products are represented by nodes, and products purchased together in the same transaction are represented by undirected links. Link weights are determined by counting the number of times two products are purchased together.

The \texttt{patents-cite} and \texttt{patents-collab} data sets are constructed from the data made available by the National Bureau of Economic Research. Nodes $v_i$ are represented by patent authors in both networks. In the citation network, a directed link $e_{i,j}$ exists if author $v_i$ produces a patent that cites a patent by author $v_j$. The weight on these links is the number of unique directional citations between the same two authors. In the collaboration network, undirected links are formed between authors who work together on the same patent. One unit of link weight is divided equally among the authors, as above, and total link weight is the sum
of all collaboration links between two people. This data has also been studied as a network in [58].

`sms` is a stream of 84 million text messages from the same source as `calls`. Senders and receivers are nodes in the network, and a transmission between them forms a directed link, sender to receiver. Like the `calls` data set, this data is not publicly available but it has been heavily studied and was first presented in [89].

2.2 Saturation

We illustrate the saturation of the networks in terms of several descriptive quantities, as shown in Figure 2.2. In these plots, the x-axis corresponds to time since the start of the stream of network events. In each plot, the reconstruction and reanalysis of a network is marked by ticks along the x-axis. Because the networks are considered after equally sized time intervals, the ticks are equal widths apart. The y-axis shows the value of a measure, such as strongly connected component (SCC) count or size of the largest SCC (LSCC), normalized by its final value. Thus all series end at a value of 1.0. The point of interest is how they converge. The `calls` and `sms` networks show a reasonable convergence, visible in terms of the convexity of the series, which indicates partial saturation. Networks that have series with areas of concavity, especially in fundamental information such as the number of nodes, are exhibiting areas of relatively accelerated growth. In some networks, the average clustering coefficient and the assortativity coefficient fluctuate particularly wildly. This is sometimes because the final values are very small, so small absolute variations result in large relative variations. Other specific reasons for this might make an interesting study unto themselves and are probably related to degree distribution and densification trends, but for
Figure 2.2. Convergence of fundamental statistics with increasing network observation time.
Figure 2.2. Convergence of fundamental statistics with increasing network observation time. (continued)
our purposes it is enough to search for cases where the curves settle gently into
their final values. This indicates that the time over which the network is observed
is sufficient to approximate closely its present underlying structure. Figure 2.2
also presents the non-longitudinal disease-g and disease-p networks to provide
contrast. Because periods from which snapshots are constructed are generated by
randomly sampling edges without replacement, the growth in the number of edges
and vertices is linear, and other measures start at or quickly approach their final
values.
CHAPTER 3

ALGORITHMS AND METHODS

There are two fundamental types of approaches: unsupervised and supervised. Unsupervised approaches rely only on characteristics of the network itself to produce a score or decision for potential links. Supervised approaches use information about subsequent formations from past periods to inform a model about potential proper classification boundaries for future periods. There are many variations of both unsupervised and supervised modeling. We start with a survey of several unsupervised topological predictors first assembled together in [63]. In addition to explaining how these predictors work, we provide some information about how they may be extended to cover networks with additional types of information. The coverage of these predictors provides the background and context for our algorithm work, which includes the PropFlow unsupervised predictor and vertex collocation profile analysis, and our classification framework, HPLP.

3.1 Unsupervised Prediction

We focus on unsupervised topological predictors covered within [63]. In that paper, many of the implementation suggestions were offered in terms of spectral graph theory and matrix operations. The data sets used in that paper consisted of only thousands of nodes, and those implementations were feasible. With even
slightly increased network sizes, they are not feasible. In addition to our slightly expanded coverage of these predictors, we offer implementation suggestions and algorithms based on computations within adjacency lists. We also include coverage of some very simple topological properties. First, such simple properties can generate useful predictions in some networks. Second, their output is useful to supervised classification models.

3.1.1 Degree and Volume

Perhaps the simplest possible topological feature for link prediction, node degree and node volume provide fundamental information about the existing connectivity of the node. The real predictive power of these simple pieces of information is generally overlooked, but their connectivity information is employed when amalgamated into other link prediction methods, such as preferential attachment [86] or Jaccard’s coefficient. Nevertheless, predictions just based on the degree of the target, for instance, can greatly outperform those from a random predictor. Node volume is implicitly used as a decomposition variable in [87] in which the authors generate predictions for the most prolific authors, those clearly most likely to make new connections. Furthermore, as we hope to demonstrate presently, simple degree and volume statistics are more useful as supervised classification features than more sophisticated methods that employ them.

3.1.2 Common Neighbors

The common neighbors [85] link prediction method is an examination of the number of nodes that are present in the first-degree neighborhood of both the source $v_i$ and the target $v_j$. In undirected networks, there is only one definition
of common neighbors. A neighbor is shared if and only if both nodes are connected to it. This means that there is no literal sense in which nodes more distant than two hops away from each other in undirected networks can have any common neighbors. One trivial extension to generalize methods based on common neighbors to predictions between more distant nodes, $d_n$, is to count nodes that are $n - m$ hops away from the source and $m$ hops away from the target. Such a procedure becomes expensive quickly. In directed networks common neighbors may be considered in the form of in-neighbors or out-neighbors. This allows for common neighbors to be applied in directed networks to nodes of an arbitrary distance in terms of a path along directed edges. So long as $v_i$ has an edge to $v_j$ and both have a relationship to $v_k$ consistent with the selected common neighbors definition, $v_j$ need not even be reachable from $v_i$. With directed edges, there are several possible definitions: a node $v_k$ is a neighbor common to $v_i$ and $v_j$ if:

- $v_i$ is connected to $v_k$ by an out-edge and $v_j$ is connected to $v_k$ by an in-edge.
- $v_i$ is connected to $v_k$ by an out-edge and $v_j$ is connected to $v_k$ by an out-edge.
- $v_i$ is connected to $v_k$ by an in-edge and $v_j$ is connected to $v_k$ by an in-edge.
- $v_i$ is connected to $v_k$ by an in-edge and $v_j$ is connected to $v_k$ by an out-edge.

The last possibility resembles the first structurally, but the two have a different meaning in the case of predicting directed edges. There are additional possibilities if we allow or require reciprocal edges. We could reduce the number of definitions to three by requiring that $v_i$ and $v_k$ are connected to the common neighbor with the same directionality or bidirectionality, and this may seem like a more reasonable definition of common neighbors, but it may also be less efficacious. It seems most sensible of all four of these scenarios in transmission networks, for instance, that
the number of out-neighbors of $v_i$ that have $v_j$ as an out-neighbor could predict the formation of a redundant link from $v_i$ to $v_j$. Such a common neighbors predictor is a degenerate case of more sophisticated path-based measures. For the sake of simplicity, our experiments are limited to case 2 in directed networks.

Common neighbors scores may consider edge weights trivially by taking the sum of the minimum shared weight value in case 2, $\sum_{k \in V} \min(w_{i,k}, w_{j,k})$, instead of the cardinality of the set intersection. The issue of using weights in common neighbors scores was explored specifically in [84] where the authors found that weighted versions of these predictors outperformed unweighted versions. In this case, the weighting scheme was to compute the arithmetic mean of the edge weights between the source and the target and their common neighbor, $\sum_{k \in V} \frac{w_{i,k} + w_{j,k}}{2}$. In reality, many functions of weights involved in connecting the common neighbor to the source and target may yield interesting scores.

We will not present, in any greater detail, the algorithms required to compute measures based on common neighbors. Such algorithms require merely performing intersection or union operations on sets that are easily available in either adjacency matrix or adjacency list network representations.

3.1.3 Jaccard’s Coefficient

Jaccard’s coefficient [97] is a common similarity measure in information retrieval. The method divides the intersection of the neighbors sets of $v_i$ and $v_j$ by the union of the neighbor sets. It is applicable to directed or undirected networks, but because it is at its core an adjustment of the common neighbors score, it is undefined for $\ell \geq 3$ on undirected networks. It is possible to convert Jaccard’s coefficient to a weighted method just as with common neighbors. In this calculation,
the denominator equivalent to the unweighted union operation is $\sum_{k \in V} w_{i,k} + w_{j,k}$.

3.1.4 Adamic/Adar

The Adamic/Adar measure [3] reckons the importance of a common neighbor $v_k$ according to the rarity of relationships between other nodes and $v_k$. It is applicable to directed or undirected networks, but it is also undefined for $n \geq 3$ on undirected networks. In adjacency matrix representations of networks, efficiently computing Adamic/Adar does not present any difficulties, but unlike the simple count of common neighbors and Jaccard’s coefficient, computing the score efficiently with adjacency list representations requires that both in-links and out-links be stored for each node because of appearance of in-degree in the denominator. It is possible to convert Adamic/Adar to take advantage of edge weights trivially by computing it as $\min \frac{\log \sum_{a \in V} w_{a,k}}{\log \sum_{a \in V} w_{a,k}}$.

3.1.5 PageRank

The PageRank algorithm of Google fame, first introduced in the academic sphere in [16], represents the importance of a node in a network based on the importance of other nodes that link to it. The final output is a probability distribution where each value represents the likelihood of a random walk reaching a particular node. This convergent measure, which is a variant of eigenvector centrality, holds intuitive potential as a link predictor. If we assume that linking to nodes that are important is desirable, an assumption implicit in preferential attachment prediction, then the PageRank of the target node represents a useful statistic.

For our experiments, we perform the original, unoptimized PageRank calcu-
ulation iteratively, checking for convergence of the vector of PageRank scores by calculating the Pearson correlation coefficient, $r$. After $r \geq 0.999$, we stop iterating and use the scores. Convergence generally requires under 10 iterations. Spearman’s $\rho$ or Kendall’s $\tau$ may both be more suitable in the sense that they are rank coefficients and we are interested in ranks, but they are more expensive to compute.

### Algorithm 1 Katz

**Input:** network $G = (V, E)$, 
$s : v_s \in V$, 
damping parameter $\beta$, 
maximum distance $\ell_{max}$

**Output:** score $S_t$ for all $\ell \leq \ell_{max}$-degree neighbors $v_t$ of $v_s$

1: $\psi \leftarrow$ empty stack
2: push($\{s, 0\}$) $\rightarrow \psi$
3: while $\psi$ not empty do
4:   $\{i, \ell\} \leftarrow$ pop($\psi$)
5:   $S_i \leftarrow S_i + \beta^\ell$
6:   if $\ell < \ell_{max}$ then
7:     for all $j : v_j \in \Gamma(v_i)$ do
8:       push ($j, \ell + 1$) $\rightarrow \psi$
9:     end for
10: end if
11: end while

### 3.1.6 Katz

The Katz score or Katz measure \[50\] is an arithmetic scalar representing the number and length of paths between the source and target node in the prediction task. The paths need not be shortest paths. To reduce the significance of long
paths, which would otherwise rapidly overwhelm shorter more significant paths due to sheer numbers in dense areas of the network, the paths are weighted according to an exponential factor $\beta$. The Katz score can be determined either through sparse matrix exponentiation or through breadth-first search. The former case is prohibitively slow on very large networks, and the latter case only offers an acceptable alternative if the range of exploration is bounded. Otherwise, it must explore all paths and not simply shortest paths, and the exploration of all paths quickly becomes intractable without memoizing optimizations.

3.1.7 Rooted PageRank

Rooted PageRank was introduced as a link predictor in [63] but is derived from the original formulation of PageRank in [16]. The subject of personalizing PageRank is studied specifically in [42]. Prediction outputs from the method correspond to the probability of visiting the target node in the prediction during a random walk from the source. The parameter $\alpha$, the probability of restarting the walk at the source, allows it to overcome a tendency it might otherwise have to get trapped in directed networks or to get caught in large dense clusters far from the source. Higher values of this parameter make the predictor more sensitive to local topological features while lower values allow it to report more meaningful scores for far-away nodes.

In practice, achieving scores in large networks involves waiting for the walks to converge. Especially with low to moderate values of $\alpha$, this may take many walk steps. In addition to the parameter, the rate of convergence depends on the size and local density of the network. In our implementation of Rooted PageRank, we perform 100,000 steps at a time, checking each time whether or not the Pearson
Algorithm 2 Rooted PageRank

**Input:** network $G = (V, E)$,
\[ s : v_s \in V, \]
walk restart parameter $\alpha$,
test step condition $T$,
correlation threshold $r_{\min}$

**Output:** score $S_t$ for all $\ell \leq \ell_{\text{max}}$-degree neighbors $v_t$ of $v_s$

1: $S_s \leftarrow 1$
2: $v_i \leftarrow v_s$
3: loop
4: if $|\Gamma(v_i)| < 1 \vee \text{rand()} < \alpha$ then
5: $v_i \leftarrow v_s$
6: else
7: $v_i \leftarrow \text{random neighbor of } v_i$
8: end if
9: $S_i \leftarrow S_i + 1$
10: if $T$ then
11: $r = \text{correlation}(S^{\text{old}}, S)$
12: if $r > r_{\min}$ then
13: return $S$
14: else
15: $S^{\text{old}} \leftarrow S$
16: end if
17: end if
18: end loop
correlation coefficient is greater than 0.9999. These particular values are only significant in the sense that we simultaneously want values that are stable and want to avoid performing the correlation test too frequently, which is much more expensive than executing a step. We determine from preliminary experimentation that these values represented a good trade-off on the larger networks, but they are otherwise arbitrary. Again, Spearman’s $\rho$ or Kendall’s $\tau$ may both be more suitable for determining convergence but are even more expensive to compute relative to walk steps.

3.1.8 SimRank

It is a common assumption in the link prediction task that two entities are related if they have something in common. In social networks, this is a principle known as homophily [81] and leveraged to achieve practical results in work such as [91]. SimRank takes the principle of homophily, which is often defined in terms of characteristics of nodes themselves and applies it to the topological embedding of the node. In the paper that introduces SimRank [48], its intuition is described clearly and succinctly thus: “two objects are similar if they are referenced by similar objects”. In terms of link prediction, the score is used such that two nodes are likely to form a link if they are both connected to nodes that are themselves connected to many of the same nodes.

Algorithm 3 provides a simple iterative procedure to calculate SimRank scores. The number of iterations to reach convergence is usually quite small; the authors observe that it is typically about 5. The memory requirements are demanding, however. Representing the SimRank matrix requires $O(|V|^2)$ space. The authors mention several ways of reducing these demands, most effectively pruning. The
Algorithm 3 SimRank

**Input:** network $G = (V, E)$, decay factor $C$, correlation threshold $r_{\text{min}}$

**Output:** score $S_{s,t}$ for every $s, t : v_s, v_t \in V$

1: for $i : v_i \in V$ do
2:    $S_{i,i} \leftarrow 1$
3:    $S_{i,i}^{\text{new}} \leftarrow 1$
4: end for
5: repeat
6:    for $i \leftarrow 1$ to $|V|$ do
7:        for $j = v_1 + 1$ to $|V|$ do
8:            $S_{i,j}^{\text{new}} \leftarrow 0$
9:            for $k : v_k \in \Gamma_{in}(i)$ do
10:                for $l : v_l \in \Gamma_{in}(j)$ do
11:                    $S_{i,j}^{\text{new}} \leftarrow S_{i,j}^{\text{new}} + S_{k,l}$
12:                end for
13:            end for
14:            $S_{i,j}^{\text{new}} \leftarrow (C \times S_{i,j}^{\text{new}})/(\deg_{in}(k) \times \deg_{in}(l))$
15:            $S_{j,i}^{\text{new}} \leftarrow S_{i,j}^{\text{new}}$
16:            $S_{i,j}^{\text{new}} \leftarrow S_{j,i}^{\text{new}}$
17:        end for
18:    swap($S, S^{\text{new}}$)
19: until correlation($S, S^{\text{new}}$) $> r_{\text{min}}$
20: return $S$
SimRank score of a node is most influenced by the nodes closest to it. Considering only subsets of the network at a time, such as $n \leq 2$ for each node, can dramatically reduce the memory requirements. Nonetheless, for highly connected nodes in dense areas of the topology, the requirements can still be restrictive. In practice, we were unable to use SimRank on the moderate or large networks due either to space or time constraints. Very recently, researchers have studied methods of computing SimRank more quickly [61, 62, 115] or relating accuracy with the number of iterations [72], and the application of their results could make SimRank a more feasible predictor in a wider range of situations. Authors of [69] have identified and proposed corrections for other limitations of SimRank in their algorithm, PageSim.

3.2 The PropFlow Method

The PropFlow predictor introduced in [68] is a path-based predictor that is insensitive to topological noise far from the source node. While it is similar to other path-based predictors such as Rooted PageRank and Katz, it outperforms them in suitable networks. The essential idea is that in many networks, such as communication and transmission networks, the flow of some resource is entirely or primarily propagated radially outward from the source. Although it may be used in any network, PropFlow has special intuitive significance as a link predictor in networks where some resource such as information flows, propagates, or cascades. In transportation networks, when a resource frequently travels from one node through neighbors to another, there is often some cost for the intermediaries. When the expected cost inherent in traveling through intermediaries overcomes the cost of establishing a new link, one can expect formation of that particular link.
In transmission networks, the measure represents the link-weighted probability that a randomly outward-propagated transmission sent by one node will reach another.

We provide a description of the procedure in Algorithm 4. In addition, Figure 3.1 demonstrates how some of the corner cases are handled. Starting from the left, source node $v_a$ has a PropFlow score of 1. Its entire output is divided among its out-neighbors, $v_b$ and $v_c$, according to the amount of weight on each edge. For each of these neighbors, their output is distributed in a similar fashion. For $v_e$ and $v_f$ with PropFlow scores of $\frac{3}{16}$, the case is more interesting. $v_e$ contributes all of its input score entirely to $v_i$. The score on $v_e$ does not include any input from $v_f$ because $v_f$ is at the same distance from the source. $v_f$ contributes half of its input to $v_i$. The contribution is unaffected by the edge traveling backward from $v_i$. The $\frac{9}{32}$ score on $v_i$ is thus the result of $1 \left( \frac{3}{16} \right) + \frac{1}{2} \left( \frac{3}{16} \right)$. Finally, the $v_i$ node has only one out-neighbor, $v_j$, and $v_j$ has only $v_i$ as an in-neighbor, so $v_i$ and $v_j$ have identical scores.
**Algorithm 4** PropFlow

**Input:** network $G = (V, E)$, 
$s : v_s \in V$

**Output:** score $S_\ell$ for all $\ell \leq \ell_{max}$-degree neighbors $v_\ell$ of $v_s$

1: insert $v_s$ into set $F$
2: $\psi \leftarrow$ empty stack
3: push $v_s \rightarrow \psi$
4: $S_s \leftarrow 1$
5: for $\ell \leftarrow 0$ to $\ell_{max}$ do
6: $\psi_{new} \leftarrow$ empty stack
7: while $\psi$ is not empty do
8: $i \leftarrow \text{pop}(\psi)$
9: $x \leftarrow S_i$
10: $y_{sum} \leftarrow 0$
11: for $j : v_j \in \Gamma(v_i)$ do
12: $y_{sum} \leftarrow y_{sum} + \text{weight}(e_{i,j})$
13: end for
14: for $j : v_j \in \Gamma(v_i)$ do
15: $S_j \leftarrow S_j + x \times \frac{\text{weight}(e_{i,j})}{y_{sum}}$
16: end for
17: if $v_i \notin F$ then
18: insert $v_i$ into $F$
19: push $v_i$ onto $\psi_{new}$
20: end if
21: end while
22: $\psi \leftarrow \psi_{new}$
23: end for
Upon completion of the algorithm, each node will have a score corresponding to the probability of receiving a randomly propagated transmission along a shortest path from the source. The higher this score, the more we might expect a link to form. At its core, PropFlow executes a breadth-first search updating all the nodes at a particular distance from the source. Because of the simple underlying search algorithm, and because it only considers shortest paths rather than all paths, PropFlow is fast compared to other path-based predictors. The memory required to complete a PropFlow computation is $O(|V|)$, and this is required to maintain the queue for breadth-first search and a data structure for recording visited nodes.

3.3 Supervised Prediction

While past studies on link prediction have focused on unsupervised single metrics, some recent works have used a supervised classification scheme, and rightly so. If one accepts the basic premise that ground truth, whether a link forms or not, is always available from prior incarnations of the network, there is no practical disadvantage to using a supervised framework. Even constructing a classification model based on one unsupervised method, say method $M$, as a single feature has the potential to outperform rankings generated by sorting the scores of method $M$ if there are multiple differentiating boundaries in the domain of scores.

Supervised algorithms are also able to capture important interdependency relationships between topological properties. While past studies simply acknowledged this fact and trained classifiers, we probe more deeply into the relevant issues so that we can fully understand how to frame the prediction problem and why a supervised framework is best for the task. We first address the question of how by examining how to best transform network data into standard data sets. We then
address the question of why by explaining that supervised approaches are adaptive and may be more general whereas unsupervised methods do not respect patterns particular to a given data set. We demonstrate that unsupervised methods cannot be trivially combined into ensembles to reduce variance and that unsupervised methods are inherently incapable of combating extreme class imbalance, a natural characteristic of link prediction in almost any network.

3.3.1 Methodology

In typical data mining tasks, there exists a training set and a testing set. Both are sets of data with a feature vector. The training set always has associated class labels. The testing set only has associated class labels for evaluation purposes. In a real deployment scenario, class labels are unavailable for the testing set at least until after prediction. Classification algorithms construct models by looking for patterns in one or more features that allow them to separate the classes. They then apply the constructed model to the features of the training data to attempt to anticipate or predict the correct class labels. In more traditional classification settings, such as detecting credit card fraud, the training and testing data has a direct natural and intuitive correspondence to reality. The class, which is binary, represents whether a transaction is fraudulent or not. Features are constructed from information about the transactions, accounts, or individuals in question. Past transactions with known fraud status provide a source of training data, and future transactions provide a source of testing data.

Link prediction is somewhat more complicated because it involves an extra abstraction layer. Features and class values do not arise out of those items, occurrences, or events directly. They instead arise from topological measurements
of a network, which is itself a representation of the items, occurrences, or events. The class is positive, with a 1-valued label, if a link forms and negative, with a 0-valued label, if it does not. Figure 3.2 illustrates the overarching framework we propose for handling both longitudinal and non-longitudinal data. In longitudinal data, the temporal dimension applies as shown in the figure. In non-longitudinal data, the set of edges can be presented in randomized order to the same overarching framework, and this random presentation will generate appropriate random holdouts of edges for training labels and testing labels. Figure 3.2a illustrates the snapshots of the network data, and Figure 3.2b focuses on a simpler representation of how data events are divided into the training features, training labels, testing features, and testing labels.

Figure 3.2a shows four separate snapshots to demarcate the networks that are built from the stream of events. The outer division separates the data used for both features in the testing data set and unsupervised predictors from the data used for class labels in the testing data set. The inner division, none of which can reside within the data used to construct class labels in the testing data set, separates the data used for training feature computation from the data used to determine class labels in training. Testing is conducted with class labels determined from exactly the same network whether the underlying predictor is unsupervised or supervised. This allows for a perfectly fair comparison between unsupervised predictors and supervised predictors. Figure 3.2b demonstrates a more precise view of the longitudinal or data volume aspects of the problem. Whether the data is longitudinal or non-longitudinal, it is best to keep corresponding longitudinal window sizes or data volumes as similar as possible between training and testing to minimize the development of problematic biases in the model.
Figure 3.2. Link prediction methodology.
3.3.2 Generality

While classifiers can generalize well to many environments in the sense that they can adjust models depending on posterior information, unsupervised methods are domain-specific. The figures presented in [64] show that predictors that serve well in one network do not necessarily serve well in all networks; our observations concur. We also find that the performance of the unsupervised methods is unstable not only from one network to the other, but from one graph-distance to another. The preferential attachment predictor is a particularly clear example in Figure 3.3. The figure shows the percentage of instances with a given score for which an unobserved potential link is subsequently observed. Intuitively, the model serves as a good predictor when low scores produce low percentages and high scores produce high percentages and an inverted predictor when the opposite is true.

In the phone network, we see that the predictions are inverted, with a higher percentage of positives falling into low scores than high scores. In the condmat network, the predictions are much better, with the highest scores corresponding to
much higher incidences of links. Finally, for both networks, we observe a similar trend with increasing geodesic distance $\ell$. The greater the distance, the better preferential attachment models the appearance of links. That is, we see lower percentages for lower scores and higher percentages for higher scores as we move from $\ell = 2$ to $\ell = 4$. This supports the intuition that preferential attachment is better as a global indicator where underlying local mechanisms such as neighbor recommendations are weaker.

3.3.3 Variance Reduction and Sampling Issues

Yet another benefit of supervised learning is that classification algorithms, especially unstable algorithms like decision trees, can benefit from reduced variance by placing them in an ensemble framework. Ensembles consist of many models that have been trained on slightly perturbed variations of the data. It is difficult or impossible to accomplish the same goal with unsupervised methods common in link prediction because the score is invariant for a given potential link. Furthermore, it is likely that network analogs to common ensemble sampling techniques are fundamentally flawed as a rough corollary of work in [105], where samples of networks with ill-behaving distributions produce new networks with different properties. Nevertheless, we wanted to explore the potential for one method of ensemble construction using unsupervised methods. To achieve the values in Figure 3.4, we construct 10 new networks, randomly selecting $p$ percent of the edges in the original network for each. Then, we compute a common neighbors score for the pair $(v_i, v_j)$ in each network and combine the scores using a summary statistic.

The figure shows that the attempt at constructing an ensemble out of an unsupervised method fails. The best AUC appears at 100%, where the network is
unsampled and there is no ensemble, which suggests that sampling the network to construct the ensemble does nothing but remove important information, a result we find unsurprising. What the figure does not show is that the $p = 100$ ROC curve dominates all ROC curves for $p < 100$, including mean and max, and that transformations of the ROC curves into precision-recall space show $p = 100$ greatly outperforms even $p = 95$. We performed these experiments only for the common neighbors classifier, but expect the same results for other unsupervised methods.

Supervised classification, on the other hand, offers many strong options for reducing variance such as bagging [14] and random forests for decision trees [15], the latter of which also increases classification efficiency. While a single classifier that incorporates several of the unsupervised methods can greatly improve classification versus those methods, variance reduction techniques can further improve it.
3.3.4 Graph Distance and Imbalance

A significant novelty of link prediction as a supervised learning problem is its extreme imbalance, which reaches past the most skewed distributions studied by the imbalance community. While unsupervised methods cannot combat this imbalance because they do not respect class distributions by definition, supervised learning schemes are able to balance data and focus on class boundaries. In this section, we will study some of the properties of that imbalance, especially as it relates to graph distance.

3.3.4.1 Sparse Networks

We proceed by constructing a formal proof of the lower bound on the class imbalance ratio for link prediction in sparse networks. The proof operates on two reasonable, almost ubiquitously satisfied assumptions. First, the network maintains the property of sparseness throughout the period of interest. Second, the network growth is limited such that the number of nodes may only double during the period of interest, although the theorem holds for any factor of growth $g$ such that $g \ll |V|$.

**Definition** Let a network $G = (V,E)$ be described as *sparse* if it maintains the property $|E| = k|V|$ for some constant $k \ll |V|$.

**Theorem 3.3.1** The class imbalance ratio for link prediction in a sparse network $G$ is $\Omega\left(\frac{|V|}{1}\right)$ when at most $|V|$ nodes may join the network.

**Proof** The number of possible links in $G$ is $|V|^2$. Then the number of missing links, $|E^c|$, is $|V|^2 - k|V| \in \Theta(|V|^2)$. Let $|V'|$ nodes and $|E'|$ links join the network. Since $|V| + |V'| \leq 2|V| \in \Theta(|V|)$, $|E| + |E'| \in \Theta(|V|)$, which requires that
$|E'| \in O(|V|)$. The number of positives is $|E'|$, and there are $|(E \cup E')^C| \in \Theta(|V|^2)$ negatives. This gives us $\Theta(|V|^3)$, equivalent to $\Omega(\frac{|V|}{1})$, as the class ratio.

Thus the imbalance issue in the general link imbalance problem becomes clear. No matter how many links we hope to anticipate, true positives ($TP$), we must accept a baseline random model that produces false positives ($FP$) such that $FP \propto TP \times |V|$. Even a model thousands of times better than random performs poorly. The severity of the problem is exacerbated by the fact that positives often represent occurrences of greater interest.

### 3.3.4.2 Vertex Neighborhoods

In link prediction, graph distance plays a primary role in determining the imbalance ratio. We define the $\ell$-degree neighborhood of a node $v_i$ as the set of nodes exactly $\ell$ hops away from $v_i$. As $\ell$ increases, the number of potential links will increase in proportion to the superlinear increase in the number of neighbors. Simultaneously, it is reasonable to expect that the new links will tend to form between nodes that are close, such as in phone where local influences such as recommendations and common neighbors pertain. This expectation holds in social networks in general and in many other network types. Figure 3.5 illustrates the imbalance behavior of the phone and condmat networks. It also demonstrates the distribution of distances between pairs of nodes for all distances where the underlying computation is feasible. It is important to note here that phone has a diameter in its largest strongly connected component of 25 while condmat has a diameter of only 19 in its largest connected component. Further, the $\ell \leq 6$-degree neighborhood of any given node in phone still includes only a moderate fraction of nodes in the network. In the much smaller condmat network, the
\( \ell \leq 6 \)-degree neighborhood of any given node often approaches the periphery and includes almost every node in the network.

The simultaneous severe increase in unformed potential links and severe decrease in links that actually form causes even more dramatic increases in imbalance ratios. **phone** imbalance goes from 131:1 at \( \ell = 2 \) to 32,880:1 at \( \ell = 4 \) and 606,926:1 at \( \ell = 6 \). **condmat** imbalance goes from 179:1 at \( \ell = 2 \) to 6,247:1 at \( \ell = 4 \). Fortunately, there is often little reason to believe that the benefit of successfully predicting links to nodes at high \( \ell \) is greater than the benefit of predicting them at low \( \ell \). For instance, a beneficial social relationship is beneficial regardless of the previous geodesic distance between the two individuals in the network. Likewise, it is important to discover the interaction between two proteins however far apart they may be in any network representation.

Given that imbalance increases so sharply between neighborhoods, and local mechanisms quickly give way to global mechanisms at higher values of \( \ell \), we suggest that each neighborhood should be treated as a separate problem in supervised

---

Figure 3.5. Class imbalance properties by geodesic distance.
learning. This also allows us to avoid the V:1 imbalance of the general problem. Additionally, in the case of the entire class of neighbor-based models, there is null output for $\ell \geq 3$ in undirected networks because there is no sense in which such nodes can have common neighbors. Many unsupervised methods have implicit or explicit adjustments for graph distance, but the fundamental distinction of neighborhood comes for free. Supervised models may benefit from a decrease in noise and, for networks in which the distance spanned by the predicted link is inconsequential, the consideration of low $\ell$ saves computational time.

The geodesic distance between two nodes in a network is perhaps the most fundamental and powerful descriptor of their relationship. For purposes of link prediction, we define a shell $\ell$ as the set of nodes reachable in precisely $\ell$ steps from some source node. This information can be used to decompose the link prediction problem in fundamental ways and bears heavily on virtually all aspects of its consideration. The distance between two nodes for consideration hints at the actual mechanisms necessary to effectively cause the formation of a link between them, the amount of computation required to generate scores with any given predictor, the imbalance ratio of the problem, and the expected performance any given predictor can achieve.

3.3.4.3 Local and Global Mechanisms

Link prediction and its associated areas of study can provide interesting insights into the mechanisms that actually drive connections between nodes in networks [87]. While there are many potential operational mechanisms driving link formation in any given graph, and these mechanisms vary from one graph to another, they may fundamentally be divided into two categories: local and global.
Depending on the type of network involved, the balance between these two fundamental types of mechanisms may shift so that one or the other dominates. Unsupervised predictors ultimately operate to differing degrees along the spectrum between these categories. In some cases, parameters to the methods may allow the operating area to shift. For instance, providing a high walk restart probability to rooted PageRank will provide better information in regions close to the source, but a low walk restart probability can allow it to provide more information at more distant nodes.

Global mechanisms of link formation are active at any scale because they are based on topological characteristics that are entirely orthogonal to distance. Preferential attachment models of link formation are a prime example. The degree of a node does not change because some other node is closer or farther away. If degree is indeed an attractive factor for new connections, then a high-degree node may exhibit an attraction across any distance so long as its degree is known to the attractee.

Local mechanisms are those that are dictated by a geodesically constrained set of influences. Recommendations by node neighbors are the iconic example. In social networks, a limited set of relatively close, strong relationships exert great influence in the choices of the ego \cite{22, 23, 37, 81}. These recommendations are most influential at small scales where they are unencumbered by the need to overcome propagation resistance many times. In terms of unsupervised link prediction, recommendation potential is most closely captured by methods that rely on information about common neighbors. The simple count of common neighbors between two nodes, Jaccard’s coefficient, and the Adamic/Adar measure all indicate the degree to which two nodes, because of their local connections, may be
influenced to make a connection to each other.

There is an important relationship between the type of the network and the mechanisms that pertain. In some networks, there is little expectation that nodes in the network have any global information. In communication networks such as networks of individuals who call, send text messages to, or email each other, the individuals in the network have no global knowledge. There is no method whereby a given individual can see that another individual 2000 miles and 5 hops away is connected to hundreds of other people, lies in a dense cluster, and initiates a communication every 5 minutes. The only knowledge nodes in such networks have is the knowledge that they acquire from communication itself, which is extremely local in terms of topology. Access to any broader knowledge either relies on propagation of that knowledge through communication events, where we can expect a rapid natural decay, or on outstanding cases where influences external to the network grant that knowledge. For instance, many people undoubtedly suspect that the president of a country has many communication contacts even if they are quite far from him in the network.

In these types of networks, the global influences are very weak. In the case in which these nodes are themselves the agents of link formation in the network, there is little reason to expect predictors reporting indicators of global mechanisms to perform well. With these weakened or non-existent global sources of attraction, the local influences hold sway and localized path predictors or neighbor-based predictors become quite good reckonings of the actual underlying mechanisms of link formation.

In other types of networks, the opposite case holds. Collaboration networks are a case in which the nodes have an extensive knowledge of the broader net-
work topology. Communication of node status in the network takes the form of broadcasts about the completion of some important paper, and these broadcasts are independent of and much less costly than the actual collaboration events that form the links. Researchers tend to know other eminent researchers in the field irrespective of how far away the two may be in the network. Working with such individuals is a great opportunity and is generally considered desirable. Even if the decision to seek out a collaboration with such individuals comes as the result of recommendations rather than directly from knowledge of the network itself, these recommendations are themselves the result of a global knowledge of the network.

In these types of networks, we expect a greater proportion of long-distance connections as the underlying global mechanisms override or compete with more local influences to collaborate more locally. We likewise expect for these long-distance connections to align well with predictors that capture the statistics that drive the global mechanisms of attraction.

Consider in Figure 3.3 an example of this behavior in the calls network, a representative of a communication network with constrained information, and the condmat network, which allows its members a global view of other members. In the calls network, we see that the predictions are inverted, with a higher percentage of positives falling into low scores than high scores. In the condmat network, the predictions are much better, with the highest scores corresponding to much higher incidences of links. Finally, for both networks, we observe a similar trend with increasing geodesic distance $\ell$. The greater the distance, the better preferential attachment models the appearance of links. That is, we see lower percentages for lower scores and higher percentages for higher scores as we move from $\ell = 2$ to $\ell = 4$. This supports the intuition that preferential attachment is
better as a global indicator where underlying local mechanisms such as neighbor recommendations are weaker.

In the case of some path-based predictors, parameters may allow for the predictor to move along the local-global spectrum. Rooted PageRank accepts a parameter $\alpha$, which is the probability of the random walk returning to the source node to restart. When this probability is very high, the random walk does not have the opportunity to explore far reaches of the network. Instead, it is practically bounded to the information available within only a few hops. In this case, rooted PageRank is a local predictor. When the probability of restarting is very low, the walk can reach great distances from the source, can get trapped in dense remote clusters in the network, and is generally less sensitive to the local topology and more able to describe distance nodes. It becomes a more global predictor.

3.3.5 Results

One of the clearest negative results is the poor quality of predictions offered by Jaccard’s coefficient. It is almost always greatly outperformed by common neighbors and Adamic/Adar. The explanation is simple. Jaccard’s coefficient is used in link prediction as a measure of how similar two nodes are in terms of their local embedding. It fails because the quantity fundamentally incorporates a sort of anti-preferential attachment. In the Jaccard’s coefficient expression, $\frac{\Gamma(v_i) \cap \Gamma(v_j)}{\Gamma(v_i) \cup \Gamma(v_j)}$, we observe that the denominator by which the measure is normalized lies between $d_i$ and $d_i + d_j$. The reduction in estimated likelihood due to the sum of the degrees is diametrically opposed to preferential attachment, which ranks its predictions by the product of degrees. In networks where preferential attachment makes sense as a predictor, poor performance by Jaccard’s coefficient is unsurprising.
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>calls</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>condmat</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>dblp-cite</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>dblp-collab</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>disease-g</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>disease-p</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>hepth-cite</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>hepth-collab</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>huddle</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>patents-cite</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>pattents-collab</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>sms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average   | 2.2 | 3.7 | 5.2 | 5.3 | 4.7 | 6.1 | 1.0 |

Ranks are for link predictors in $\ell = 2$, from best (1) to worst (7). Included are Adamic/Adar (A), common neighbors (B), Jaccard’s coefficient (C), preferential attachment (D), PropFlow (E), rooted PageRank (F), and HPLP (G).
Adamic/Adar, also in the common neighbors family, exhibits high performance across the range of networks. Unlike Jaccard’s coefficient, it uses the rarity of the shared neighbors as a factor to increase prediction rankings rather than decreasing them according to degree. In this way, it actually compresses a great deal of information about the network, the degree of every shared neighbor, into a powerful scalar normalizing factor. The degree of the shared neighbor, whether extremely low or extremely high, is not itself involved in the attachment process but does strongly indicate similarity between source and target.

Regardless of the network and the relative quality of the unsupervised methods within each network, supervised classification provides the best AUC performance. This is demonstrated by Table 3.1. The AUC performance increase underlying the ranks is on average 4% above the best unsupervised predictor, as much as 13% above the best unsupervised predictor, and is as much as 74% above the worst unsupervised predictor. Further, examination of the ROC curves in Figure 3.6 shows that for many networks supervised predictors dominate unsupervised predictors in ROC space in addition to exhibiting greater AUC. The plots also allow a few general observations. The citation networks allow better ROC curves across the predictors than the collaboration networks. The sms network has much lower ROC convex hulls than its calls cousin, which suggests that texting relationships are inherently less predictable. Despite the potential usefulness of the PageRank predictors, their predictions generally are very little better than random. The two biological networks are clear examples of how different unsupervised predictors achieve very different results from one network to another while supervision allows for much more general predictive power.
Figure 3.6. Predictor ROC curve performance at $\ell = 2$. 
Figure 3.6. Predictor ROC curve performance at $\ell = 2$. (continued)
3.4 Vertex Collocation Profiles

Prediction methods thus far have incorporated simple node attributes, dyadic properties, common neighbors, or paths. These are all in some sense approximations describing the topological embedding of two vertices relative to each other. We take a major step beyond these methods by introducing a technique for describing very precisely how two nodes are collocated in terms of the broader network.

3.4.1 Introduction

As already stated, link prediction is the task of inferring links in a graph $G_{t+1}$ based on the observation of a graph $G_t$. Link analysis, less formally defined, is the problem of identifying evolutionary processes or growth mechanisms in a network that are responsible for the formation of new relationships between nodes. We formally define a new technique for performing both link prediction and link analysis based on a restrictive representation of the local topological embedding of the source and target vertices. This idea is a generalization and extension of the triangle counting approach for multi-relational prediction in [29]. It also draws on concepts from literature on graphlets as introduced in [92] and to a lesser degree from motif analysis as discussed in [83].

Many existing link prediction models compress a selection of simple information in theoretically or empirically guided ways. By contrast the vertex collocation profile approach preserves as much topological information as possible about the embedding of the source and target vertices. It also extends naturally to multi-relational networks and can thereby encode a variety of additional information such as edge directionality. It can encode continuous quantities such as edge
weights by binning into relational categories, such as high activity and low activity. Information about the nature of relationships is maintained as structures are identified within the network. We proceed with a formal exploration of VCP, discuss its relationship to isomorphism classes, provide algorithms that formally describe VCP computations, and demonstrate the potential of VCP in link prediction and analysis as well as feasibility in terms of computational time. Fast forms of the algorithms listed herein are all implemented in C++ and integrated into the LPmade [65] link prediction software and are thus freely available on MLOSS.

3.4.2 Vertex Collocation Profiles

Formally, a vertex collocation profile (VCP), written as \( VCP^{n,r}_{i,j} \), is a vector describing the relationship between two vertices, \( v_i \) and \( v_j \), in terms of their common membership in all possible subgraphs of \( n \) vertices over \( r \) relations. A VCP element, \( VCP^{n,r}_{i,j}(x) \) is defined as a distinct embedding of \( v_i \) and \( v_j \) within a particular isomorphism class of \( n \) vertices and is represented by a uniquely addressable cell in the VCP vector. Figure 3.7 illustrates the first 16 elements of \( VCP^{3,2}_{s,t} \), where the two relations correspond to edge directionality.

In general, we can encode the connectivity in any multi-relational network of \( r \) relations with \( 2^r \) different types of connections. We use \( 2^r \) instead of \( 2^{r-1} \) because structural holes are often as important as links [17], and we consider the lack of relation as itself a type of connection. Undirected single-relational networks exhibit two types of connections: existent and nonexistent. Directed single-relational networks are similar to undirected bi-relational networks and have four types of connections: nonexistent, outward, inward, and bidirectional.

The cardinality of \( VCP^{n,r} \) depends upon the number of vertices \( n \) and the
Figure 3.7. Elements of VCP\[^{3,2}\]_{s,t}. 16 through 31 are identical to their modulo 16 counterparts except for the presence of $\epsilon_{t,a}$. 
number of types of relationship $r$ in the set of relations $\mathcal{R}$. The space grows exponentially in the number of vertices with the base as the cardinality of the power set of relations. The formula for the number of subgraphs is written in intuitive form in Equation (3.1). The multiplier accounts for the number of possible collocation structures disregarding any links between the source and the target. The multiplicand is the number of different ways two vertices with the same embedding can appear based on the different link possibilities between them.

\[
(2^r)^{\left(\frac{n(n-1)}{2}-1\right)} \times 2^{r-1}
\]  

(3.1)

We can manipulate these to achieve the simpler formula below.

\[
2^{\frac{n(n-1)r}{2}-1}
\]  

(3.2)

Table 3.2 illustrates the number of labeled subgraphs that compose the VCP given different values of $n$ and $r$. 

<table>
<thead>
<tr>
<th>$n$</th>
<th>$r$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>32</td>
<td>256</td>
<td>2048</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>2048</td>
<td>131072</td>
<td>8.4 $\times$ 10$^6$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>512</td>
<td>524288</td>
<td>5.4 $\times$ 10$^8$</td>
<td>5.5 $\times$ 10$^{11}$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>16384</td>
<td>5.4 $\times$ 10$^8$</td>
<td>1.8 $\times$ 10$^{13}$</td>
<td>5.8 $\times$ 10$^{18}$</td>
<td></td>
</tr>
</tbody>
</table>

TABLE 3.2

COUNTS OF ENUMERATED VCP SUBGRAPHS
The number of subgraphs grows at such a rate as to make the sheer size of output unmanageable for large values of \( n \) and \( r \). The rate of growth of VCPs is much slower due to superlinear increases in the isomorphisms with increasing \( n \), but nonetheless VCP cardinality also grows quickly. Fortunately, the most important information is typically located close to the source and target vertices, and many networks have few types of relationships. When the number of relationship types is high, relationships can be compressed or discarded in various ways, albeit with a loss of information.

3.4.2.1 Isomorphisms

Isomorphic subgraphs are closely related to VCP elements. In Figure 3.8, it is impossible to distinguish subgraph 1 from subgraph 2, and the prevalence of the frequency of each during counting will depend upon implementation details of the counting algorithm that determine the order in which the vertices are selected. These subgraphs map to the same VCP element, \( \text{VCP}^{4,1}_{s,t}(1) \), and the count of that element is the sum of the counts of the isomorphic subgraphs.

Isomorphisms that require a mapping between \( v_s \) and \( v_t \), for instance subgraph 1 and subgraph 8 in Figure 3.8, do not share the same VCP element even though they reside within the same isomorphism class. VCP elements ignore isomorphisms that require mapping \( v_s \) to \( v_t \) because VCP describes the local embedding of these two explicitly identified vertices. In undirected graphs, elements such as \( \text{VCP}^{4,1}_{s,t}(1) \) and \( \text{VCP}^{4,1}_{s,t}(3) \) together supply information regarding symmetry or asymmetry in the density of the embedding of \( v_s \) and \( v_t \). The distinction in directed networks is more obvious and relates to the potential significance of the difference in the local topologies of the source of a new link and its target. Figure
Figure 3.8. Subgraphs that form VCP$^{4,1}$ and the mapping of isomorphic subgraphs to VCP elements.

3.8 shows all the subgraphs pertinent to VCP$^{4,1}$ and their corresponding mappings to elements.

Counting the number of elements in VCP$^{n,r}$ is related to the complex problem of counting the number of isomorphism classes in graphs of $n$ vertices. In VCP$^{3,r}$, each enumerated subgraph maps uniquely to a VCP element. $v_s$ and $v_t$ are fixed, and there is only one permutation of the remaining vertex. In VCP$^{4,r}$, there are two mappable vertices and the number of VCP$^{4,r}$ elements is described as:

$$2^{6r-2} + 2^{4r-2}$$

(3.3)

The derivation of a general formula for $|VCP^{n,r}|$ for all $n$ and $r$ is extremely
TABLE 3.3

CARDINALITY OF VCP$_{i,j}^{n,r}$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$r$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td>4</td>
<td>32</td>
<td>256</td>
<td>2048</td>
<td>16384</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>20</td>
<td>1088</td>
<td>66560</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>120</td>
<td>91520</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>996</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>12208</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

combinatorially involved and its discussion is beyond the scope of this work. We have instead provided software that computes VCP cardinalities and subgraph-to-element mappings for $3 < n < 8$ and $1 < r < 8$. For practical purposes, Table 3.3 shows the cardinality of all VCPs with fewer than a million elements.

3.4.2.2 Addressing

We define a VCP addressing scheme similar to the isomorphism certificate addressing scheme in [54]. The subgraphs from which the elements are derived are indexed by assigning powers of $2^r$ to edges in the adjacency matrix in increasing lexicographical order starting with $e_{1,3}$ and ending with $e_{n-1,n}$. $v_s$ and $v_t$ are defined as $v_1$ and $v_2$ respectively, and $e_{1,2}$ is the edge of highest value. The value of each edge is multiplied by the index of the lexicographically ordered power set, $\mathcal{P}(\mathcal{R})$, corresponding to the ordered set of $\mathcal{R}$ relations on the edge. Figures 3.7 and 3.8 demonstrate the indexing scheme for two different values of $n$ and $r$. 

58
For any selection of vertices $v_s, v_t, v_3, ..., v_n$, this addressing scheme will map the resulting multi-relational subgraph to an index that exists within a set of indices of isomorphic structures.

We define the unique address of a VCP element as the subgraph representative with the lowest index within the corresponding isomorphism class. This addressing scheme provides a unique address for all elements in all VCPs. The addresses for elements in VCP$^{4,1}$ are provided in Figure 3.8. Because manual identification of isomorphism classes is error-prone and difficult especially as the number of subgraphs increases, we have provided a program that outputs the mapping from all subgraph indices to their corresponding element addresses for all VCPs.
3.4.2.3 Directionality

Directed networks with \( r \) relations can be treated similarly to undirected networks with \( 2r \) relations with one major caveat. The subgraph-to-element mapping differs with directed networks. Taking \( e_{x_{i,j}} \) momentarily as notation for an edge of relation \( x \) and \( 2x \) as relation \( x \) in the opposite direction, \( e_{x_{i,j}}^x \equiv e_{x_{j,i}}^x \) in undirected multi-relational networks, but \( e_{x_{i,j}}^x \equiv e_{2x_{j,i}}^{2x} \) in directed networks. In the context of larger subgraphs, this causes more isomorphic equivalences and decreases the cardinality of the VCP by comparison to its undirected pseudo-equivalent, a fact demonstrated in Figure 3.9. For instance, VCP\(^4,2\) contains 1088 elements whereas the directed variant of VCP\(^4,1\) contains only 1056 elements. Therefore, the algorithms in Section 3.4.3 and the procedures described by all provided code work with only minor adjustments, which are essentially related to the subgraph-to-element mapping. We include software to construct the mapping for VCP\(^4,1\) where there are actually two relations corresponding to directionality.

3.4.3 Algorithms

Algorithms 5 and 6 serve as reference algorithms and not as optimized or even asymptotically optimal approaches for VCP element counting. In fact, implementations of the naïve VCP\(^4,1\) algorithm fail to return within hours for networks with even thousands of nodes. Fortunately, it is possible to design much faster approaches, and we implemented these approaches and provide them as a set of C++ files. We also present a more innovative algorithm, Algorithm 7 for counting VCP\(^4,1\), which corresponds to the approach in the code for that VCP.
3.4.3.1 3-Vertex VCP

Algorithm 5 demonstrates how to calculate $VCP^{3,r}$ for the set of $r$ relations in $\mathcal{R}$. $\Phi(\mathcal{P}(\mathcal{R}), e_{x,y})$ refers to a procedure to determine the index of the multirelational edge $e_{x,y}$ in $\mathcal{P}(\mathcal{R})$, the lexicographically ordered power set of relations. This procedure can derive power set indices efficiently by setting individual bits in the index according to the presence of the relation corresponding to that bit and indexing the bits by the natural order of the relations themselves.

This naïve algorithm first determines the contribution of any edge types between $v_i$ and $v_j$, the fixed source and target vertices for the link. Then it counts subgraphs with a third vertex connected to both $v_i$ and $v_j$, connected only to $v_i$, and connected only to $v_j$. $\sigma$ represents the identity of an edge within $\mathcal{P}(\mathcal{R})$ and $\lambda$ represents the index of the subgraph. Because no isomorphisms exist with only one mappable vertex, the algorithm directly increments the VCP elements corresponding to the underlying subgraph index as contrasted to Algorithm 6 wherein the subgraph index must be mapped to an element address.

A naïve implementation iterates through each vertex in the network and determines the corresponding subgraph index by summing the edge contributions. For one free vertex, this approach has complexity $O\left(|V| \log \left(\frac{|E|}{|V|}\right)\right)$ per edge output assuming $O\left(\log \left(\frac{|E|}{|V|}\right)\right)$ neighbor search time for the average case. This complexity is probably feasible for small networks, but may require an unacceptably long time for large networks. It is simple to improve upon this approach by considering only the vertices that are neighbors, denoted by $\Gamma(v_x)$ of $v_i$, $v_j$, both, or neither and performing set operations. $VCP_{i,j}^{3,1}(0)$ is populated by subtracting $|\Gamma(v_i) \cup \Gamma(v_j)|$ from $|V| - 2$. $VCP_{i,j}^{3,1}(1)$ and $VCP_{i,j}^{3,1}(2)$ are populated by computing set differences $|\Gamma(v_i) - \Gamma(v_j)|$ and $|\Gamma(v_j) - \Gamma(v_i)|$ respectively. $VCP_{i,j}$ is computed as the
Algorithm 5 VCP$^{3,r}$

Input: network $G = (V, E)$, relations $\mathcal{R}$,
      $i : v_i \in V$,
      $j : v_j \in V$

Output: $\text{VCP}^{3,|\mathcal{R}|}_{i,j}$

1: $\sigma_{i,j} \leftarrow \Phi(\mathcal{P}(\mathcal{R}), e_{i,j})$
2: for $k : e_{i,k} \in E \land e_{j,k} \in E$ do
3:   $\sigma_{i,k} \leftarrow \Phi(\mathcal{P}(\mathcal{R}), e_{i,k})$
4:   $\sigma_{j,k} \leftarrow \Phi(\mathcal{P}(\mathcal{R}), e_{j,k})$
5:   $\lambda \leftarrow 2^{|\mathcal{R}|}\sigma_{i,j} + 2^{|\mathcal{R}|}\sigma_{j,k} + \sigma_{i,k}$
6:   $\text{VCP}^{3,|\mathcal{R}|}_{i,j}[\lambda] \leftarrow \text{VCP}^{3,|\mathcal{R}|}_{i,j}[\lambda] + 1$
7: end for
8: for $k : e_{i,k} \in E \land e_{j,k} \notin E$ do
9:   $\sigma_{i,k} \leftarrow \Phi(\mathcal{P}(\mathcal{R}), e_{i,k})$
10:  $\lambda \leftarrow 2^{|\mathcal{R}|}\sigma_{i,j} + 2^{|\mathcal{R}|}\sigma_{j,k} + \sigma_{i,k}$
11:  $\text{VCP}^{3,|\mathcal{R}|}_{i,j}[\lambda] \leftarrow \text{VCP}^{3,|\mathcal{R}|}_{i,j}[\lambda] + 1$
12: end for
13: for $k : e_{i,k} \notin E \land e_{j,k} \in E$ do
14:   $\sigma_{j,k} \leftarrow \Phi(\mathcal{P}(\mathcal{R}), e_{j,k})$
15:   $\lambda \leftarrow 2^{|\mathcal{R}|}\sigma_{i,j} + 2^{|\mathcal{R}|}\sigma_{j,k} + \sigma_{i,k}$
16:   $\text{VCP}^{3,|\mathcal{R}|}_{i,j}[\lambda] \leftarrow \text{VCP}^{3,|\mathcal{R}|}_{i,j}[\lambda] + 1$
17: end for
18: $\lambda \leftarrow 2^{|\mathcal{R}|}\sigma_{i,j}$
19: for $k : e_{i,k} \notin E \land e_{j,k} \notin E$ do
20:   $\text{VCP}^{3,|\mathcal{R}|}_{i,j}[\lambda] \leftarrow \text{VCP}^{3,|\mathcal{R}|}_{i,j}[\lambda] + 1$
21: end for
22: return $\text{VCP}^{3,|\mathcal{R}|}_{i,j}$
intersection $|\Gamma(v_i) \cap \Gamma(v_j)|$. These operations can be performed quickly especially in graphs in which adjacencies are represented as ordered lists of neighbors. This implementation has average-case complexity $O\left(\frac{|E|}{|V|}\right)$ per edge output.

3.4.3.2 4-Vertex VCP

Algorithm 6 iterates through every pair of free vertices, yielding a complexity of $O\left(|V|^2 \log\left(\frac{|E|}{|V|}\right)\right)$ from $\binom{|V|}{2}$ pairs of free vertices. This requires trillions of operations even for small networks. It is possible to reduce this time greatly by restricting consideration to known neighbors as described in the discussion of Algorithm 5, but naïve implementations of this optimization involve many expensive operations in hashes or balanced search trees.

Algorithm 7 instead uses a minimal number of set operations implemented as merge operations on ordered lists. Figure 3.10 provides an illustration of the sets mentioned in the following explanation. First, the number of connected pairs and unconnected pairs is computed once for the entire network, and these values are represented as $\chi_G$ and $\epsilon_G$ respectively. We must also track the connected pairs and unconnected pairs that exist among pairs of vertices that appear in discovered four-node subgraphs. This allows us to differentiate VCP$^{4,1}(0)$ from VCP$^{4,1}(10)$. We start by constructing a set of potential “third position” vertices, $\Gamma_3$, as $\Gamma(v_i) \cap \Gamma(v_j)$. For each member of $\Gamma_3$, we construct two disjoint sets of “fourth position” vertices, $\Gamma_4$ containing vertices reachable by our current member of $\Gamma_3$ but not contained within $\Gamma_3$, and $\Gamma_{4a}$ constructed from $\Gamma_3$ excluding the current member of $\Gamma_3$. In $\Gamma_4$, we count new connections and gaps in the configuration, and we increment the counter for the corresponding subgraph. For $\Gamma_{4a}$, we do not count connections and gaps since configurations using those set members are counted
Algorithm 6 Simple VCP$^{4,1}$

**Input:** network $G = (V,E)$,
- $i : v_i \in V$,
- $j : v_j \in V$,
- subgraph-element mapping $M$

**Output:** $VCP_{i,j}^{4,1}$

1: for $k : v_k \neq v_i \land v_k \neq v_j$ do
2: for $l : v_l \neq v_i \land v_l \neq v_j \land l > k$ do
3: $\lambda \leftarrow 0$
4: if $e_{i,k} \in E$ then
5: $\lambda \leftarrow \lambda + 1$
6: end if
7: if $e_{i,l} \in E$ then
8: $\lambda \leftarrow \lambda + 2$
9: end if
10: if $e_{j,k} \in E$ then
11: $\lambda \leftarrow \lambda + 4$
12: end if
13: if $e_{j,l} \in E$ then
14: $\lambda \leftarrow \lambda + 8$
15: end if
16: if $e_{k,l} \in E$ then
17: $\lambda \leftarrow \lambda + 16$
18: end if
19: $VCP_{i,j}^{4,1}(M(\lambda)) \leftarrow VCP_{i,j}^{4,1}(M(\lambda)) + 1$
20: end for
21: end for
22: return $VCP_{i,j}^{4,1}$
Algorithm 7 Fast VCP$^{4,1}$

**Input:** network $G = (V, E)$,
- $i : v_i \in V$,
- $j : v_j \in V$,
- subgraph-element mapping $M$,
- count of connected pairs $\chi_G$,
- count of unconnected pairs $\epsilon_G$

**Output:** $VCP_{i,j}^{4,1}$

1. $v \leftarrow 2$
2. $\chi \leftarrow 0$
3. $\epsilon \leftarrow 0$
4. $\Gamma_3 \leftarrow \Gamma(v_i) \cap \Gamma(v_j)$
5. for $k : v_k \in \Gamma_3$ do
6.   $\lambda_1 \leftarrow 0$
7.   if $e_{i,k} \in E \lor e_{k,i} \in E$ then
8.     $\lambda_1 \leftarrow \lambda_1 + 1$
9.     $\chi \leftarrow \chi + 1$
10. else
11.     $\epsilon \leftarrow \epsilon + 1$
12. end if
13. $\Gamma_4 \leftarrow \Gamma(v_k) - \Gamma_3$
14. $v \leftarrow v + |\Gamma_4|$
15. for $l : v_l \in \Gamma_4$ do
16.   $\lambda_2 \leftarrow \lambda_1$
17.   if $e_{i,l} \in E \lor e_{l,i} \in E$ then
18.     $\lambda_2 \leftarrow \lambda_2 + 2$
19.     $\chi \leftarrow \chi + 1$
20. else
21.     $\epsilon \leftarrow \epsilon + 1$
22. end if
23. if $e_{j,l} \in E \lor e_{l,j} \in E$ then
24.     $\lambda_2 \leftarrow \lambda_2 + 8$
25.     $\chi \leftarrow \chi + 1$
26. else
27.     $\epsilon \leftarrow \epsilon + 1$
28. end if
Algorithm 7 Fast VCP\(^{4,1}\) (continued)

29: \hspace{1em} \textbf{if} \ e_{k,l} \in E \lor e_{l,k} \in E \hspace{1em} \textbf{then}
30: \hspace{2em} \lambda_2 \leftarrow \lambda_2 + 16
31: \hspace{2em} \chi \leftarrow \chi + 1
32: \hspace{1em} \textbf{else}
33: \hspace{2em} \epsilon \leftarrow \epsilon + 1
34: \hspace{1em} \textbf{end if}
35: \hspace{1em} VCP\(_{i,j}^{4,1}(M(\lambda_2)) \leftarrow VCP\(_{i,j}^{4,1}(M(\lambda_2)) + 1
36: \hspace{1em} \textbf{end for}
37: \hspace{1em} \textbf{for} \ l : v_l \in \Gamma_3 \land l > k \hspace{1em} \textbf{do}
38: \hspace{2em} \lambda_2 \leftarrow \lambda_1
39: \hspace{2em} \textbf{if} \ e_{i,l} \in E \lor e_{l,i} \in E \hspace{1em} \textbf{then}
40: \hspace{3em} \lambda_2 \leftarrow \lambda_2 + 2
41: \hspace{2em} \textbf{end if}
42: \hspace{2em} \textbf{if} \ e_{j,l} \in E \lor e_{l,j} \in E \hspace{1em} \textbf{then}
43: \hspace{3em} \lambda_2 \leftarrow \lambda_2 + 8
44: \hspace{2em} \textbf{end if}
45: \hspace{2em} \textbf{if} \ e_{k,l} \in E \lor e_{l,k} \in E \hspace{1em} \textbf{then}
46: \hspace{3em} \lambda_2 \leftarrow \lambda_2 + 16
47: \hspace{3em} \chi \leftarrow \chi + 1
48: \hspace{3em} \textbf{else}
49: \hspace{4em} \epsilon \leftarrow \epsilon + 1
50: \hspace{3em} \textbf{end if}
51: \hspace{2em} VCP\(_{i,j}^{4,1}(M(\lambda_2)) \leftarrow VCP\(_{i,j}^{4,1}(M(\lambda_2)) + 1
52: \hspace{2em} \textbf{end for}
53: \hspace{1em} \zeta \leftarrow |\Gamma_3| + |\Gamma_4|
54: \hspace{1em} VCP\(_{i,j}^{4,1}(M(\lambda_1)) \leftarrow VCP\(_{i,j}^{4,1}(M(\lambda_1)) + |V| - 2 - \zeta
55: \hspace{1em} VCP\(_{i,j}^{4,1}(M(0)) \leftarrow VCP\(_{i,j}^{4,1}(M(0)) + |V| - 2 - \zeta
56: \hspace{1em} \textbf{end for}
57: \hspace{1em} \rho \leftarrow e_{i,j} \in E \lor e_{j,i} \in E
58: \hspace{1em} VCP\(_{i,j}^{4,1}(M(16)) \leftarrow VCP\(_{i,j}^{4,1}(M(16)) + \chi_G - (\chi + \rho)
59: \hspace{1em} VCP\(_{i,j}^{4,1}(M(0)) \leftarrow VCP\(_{i,j}^{4,1}(M(0)) + \epsilon_G - (\epsilon + \gamma(\rho)) - (2|V| - v)
60: \hspace{1em} \textbf{return} VCP\(_{i,j}^{4,1}
when they serve as members of $\Gamma_3$, or “third position” vertices. Likewise, we only count subgraphs with two members from $\Gamma_3$ when the member from $\Gamma_{4a}$ compares lower. This avoids multi-counting. After considering the configurations from all members of $\Gamma_4$ and $\Gamma_{4a}$, we account for structures with isolates by contributing $|V| - 2 - |\Gamma_3| - |\Gamma_4|$ to $VCP^{4,1}_{i,j}(M(\lambda_1))$. We also account for multi-counting of $VCP^{4,1}(0)$ due to duplicate consideration of gaps by subtracting the same quantity from $VCP^{4,1}_{i,j}(M(0))$. Finally, we compute $VCP^{4,1}(0)$ and $VCP^{4,1}(10)$ using our computations of vertices and gaps in the vertices we have encountered in the single $\Gamma_3$ and multiple $\Gamma_4$ sets and subtract their contributions from the contributions from the entire network. It is possible to perform the entire procedure using a few relatively inexpensive merge operations in ordered vectors or lists and entirely avoiding hashes or balanced trees. This exposition mostly describes the procedure to quickly compute $VCP^{4,1}$ albeit omitting minor implementation details. We refer more interested readers to the code itself.

3.4.4 Extension to Complex Networks

It is trivial to extend VCP algorithms to networks more complex than those on which we obtained our results. This includes any form of edge feature such
as directionality, weight, temporality, different relation types, or any information describing edges or vertex pairs that either exists categorically or can be quantized. One amenable network representation associates an ordered set of bits with each edge. Each bit corresponds to the presence of a particular relation or some Boolean descriptor for a pair of vertices. The determination of the existence of an edge for single-relational data instead becomes an evaluation of the edge as the binary-coded integral value of the ordered set of bits. This is one conceivable implementation for $\Phi(P(R), e_{x,y})$ in Algorithm 5, which replaces the constant values for all $\lambda$ updates in Algorithms 6 and 7. For most values of $r$, this can be implemented as a constant-time operation equivalent to retrieving the value of a variable, so the asymptotic cost of populating the VCP vector is unaffected. Excepting the additional costs of writing output and of allocating and deallocating the storage necessary to hold the additional multi-relational structural elements, which is inexpensively proportional to $2^r$, the computational complexity of the multi-relational extension is no greater than for single-relational networks.

3.4.5 Results

First, we illustrate how VCP can serve as a powerful link analysis and modeling tool. Then we perform a standard comparison of the link prediction efficacy of VCP and a selection of other methods. Timing results are rarely provided in link prediction work despite vast differences in the running time and feasibility of methods. For this reason and because we believe many might suspect that a completely theoretically aligned implementation of VCP is computationally unachievable, we also provide comparative timing results.
3.4.5.1 Experimental Setup

To run our experiments, we integrated VCP with the LPmade link prediction software [65]. LPmade uses a GNU make architecture to automate the steps necessary to perform supervised link prediction. This integration will allow those interested in VCP for link prediction and other purposes to test it on their networks easily.

We compare link prediction output against representatives from different predictor families established as strong by prevailing literature [64]. The unsupervised selections include the Adamic/Adar common neighbors predictor [3], the Katz path-based predictor [50], and the preferential attachment model [10, 86]. We also compare against the HPLP supervised link prediction framework contributed by [68], including the PropFlow feature. HPLP combines simple topological information such as node degree and common link predictors into a bagged random forests classification framework with undersampling, a framework that the authors showed works well.

When performing classification using VCPs, we opted for the bagged [14] random subspaces [45] implementation from WEKA 3.5 [114]. This classification scheme offers significantly lower peak memory requirements than random forests while simultaneously providing the potential to handle feature redundancy [45]. We considered presenting results with HPLP also using random subspaces, but we determined that random subspaces produced decreased or comparable performance to the original reference implementation, so we present HPLP results unmodified using random forests [15].

We used the default values from HPLP of 10 bags of 10 random forest trees, 10 bags of 10 random subspaces for VCP classifiers, and training set undersampling to
25% positive class prevalence in training. We did not change the size or distribution of the testing data. For undirected networks, we resolve \( f(v_s, v_t) \neq f(v_t, v_s) \), by computing the arithmetic mean to serve as the final prediction output. By default, LPmade includes features that consider edge weights such as the sum of incoming and outgoing link weights, and PropFlow inherently considers edge weights. We are interested in the comparative prediction performance of the link structure alone, so we ran all predictors on the networks disregarding edge weights. There are many different ways to assign edge weights to all the networks here, and the particular choice of edge weight and the precise decision about how to incorporate it into the VCPs would distract from the study.

Computing and evaluating predictions for all possible links on large, sparse networks with any prediction method is infeasible for multiple computational reasons including time and storage capacity. Link prediction within a two-hop geodesic distance provides much greater baseline precision in many networks [68, 98], so effectively predicting links within this set offers a strong indicator of reasonable deployment performance. For all compared prediction methods, we restricted the prediction task by distance and only considered performance comparisons for potential links spanning two hops within the training data due to their higher prior probability of formation and computational feasibility.

Reciprocity is an important consideration for link formation in directed networks, so when performing undirected VCP computations on directed networks, we deviate slightly from the definitions above to consider existing reciprocal links as a different relation type and accordingly double the width of the VCP feature space to include elements with and without the reciprocal link.
3.4.5.2 Link Analysis

VCPs can assist with a variety of functions regarding link analysis, and post hoc analysis of link prediction output is an interesting example. We start with the performance of the Adamic/Adar predictor in the sms network. As we show in Tables 3.5 and 3.6 it achieves AUROC performance of 0.642 and AUPR performance of 0.009410. It may be helpful to us as modelers to understand better why Adamic/Adar fails. We can do this by looking at other simple characteristics of the graph such as degree, centrality measures, or clustering coefficient, but VCPs offer a fine-grained and informative view of links as they are embedded in the broader topology.

We select the Adamic/Adar predictor and first extract the positives from the top 10 million predictions and place them in one set. We place all remaining positives in a second set. The threshold is arbitrary but serves to separate positives into two roughly equally sized sets. For the positives in each of these sets, we can very quickly compute the VCPs of our choice. For simplicity in the demonstration, we choose undirected VCP$^3,1$. Since sms is a directed network, we extend VCP$^3,1$ to include reciprocal edges between $v_i$ and $v_j$ if they exist. This procedure provides two multi-column distributions of corresponding columns. One logical first step is to compute the distributional divergences of these columns. The distributions are highly skewed, so we use Hellinger distance [49], a non-parametric measure of divergence ranging from 0 to $\sqrt{2}$. The distances are shown in Table 3.4.

We select the most divergent element, the fifth, and examine the distribution of highly ranked and lowly ranked Adamic/Adar outputs more closely in Figure 3.11. The fifth element is one in which a reciprocal link precedes the target link in the prediction. The distributions indicate that highly ranked predictions in
<table>
<thead>
<tr>
<th>Address</th>
<th>Element</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>⊗ ⊗</td>
<td>0.006</td>
</tr>
<tr>
<td>1</td>
<td>⊗ ⊗</td>
<td>0.091</td>
</tr>
<tr>
<td>2</td>
<td>⊗ ⊗</td>
<td>0.052</td>
</tr>
<tr>
<td>3</td>
<td>⊗ ⊗</td>
<td>0.063</td>
</tr>
<tr>
<td>4</td>
<td>⊗ ⊗</td>
<td>0.006</td>
</tr>
<tr>
<td>5</td>
<td>⊗ ⊗</td>
<td>0.116</td>
</tr>
<tr>
<td>6</td>
<td>⊗ ⊗</td>
<td>0.051</td>
</tr>
<tr>
<td>7</td>
<td>⊗ ⊗</td>
<td>0.106</td>
</tr>
</tbody>
</table>
Adamic/Adar tend to have more connected source vertices than lowly ranked predictions. Since having many neighbors in common tends to follow from simply having many neighbors, this is not surprising, but the greater dissimilarities of elements 1, 5, and 7 and lesser dissimilarities of 6 and 2 suggest that the connectedness of the link initiator may be more significant than that of the receiver. Adamic/Adar as a model fails to sufficiently separate links containing low-degree source vertices in this network.

In this particular case, we could have obtained the same information by examining the degree distributions of the two sets, but 4-vertex VCPs offer much more distinctive structural information with their greater complexity. This is only one of many ways to perform post hoc link prediction analysis that focuses on the causes of type 2 error in prediction output. Similar analysis could be applied to analyze type 1 error in an attempt to increase precision. Many more powerful and
imaginative variations on these techniques apply to link analysis and clustering in general.

3.4.5.3 Prediction Performance

The area under the receiver operating characteristic curve (AUROC) can be a deceptive measure of performance in scenarios with extreme imbalance \cite{1}, and area under the precision-recall curve (AUPR) exhibits higher sensitivity to performance disparities in the same scenarios \cite{30}. We will provide results for those interested in traditional AUROC, but we will also present AUPR results and will mainly restrict our analysis to those results. Tables 3.5 and 3.6 shows the comparative AUROC and AUPR performance of Adamic/Adar, Katz, preferential attachment, HPLP, and VCPs in link prediction for potential links spanning a geodesic distance of two hops.

In general, we expect the information content of VCPs to increase in the left-to-right order presented in the table. Depending on the significance of directedness in the network, the expectation of performance from VCP 3D and VCP 4U may change. We point the reader to \textit{calls, dblp-cite, dblp-collab, disease-g, disease-p, hepth-cite, huddle, patents-collab, and sms} as conformant examples. We suspect that the exceptions indicate cases in which the classification algorithm was either overfitting the training set or failed to create a sufficiently optimized model in the high-dimensional space.

In general, we expect the information content of VCPs to increase in the left-to-right order presented in the table. Depending on the significance of directedness in the network, the expectation of performance from VCP 3D and VCP 4U may change. We point the reader to \textit{calls, dblp-cite, dblp-collab, disease-g, disease-p, hepth-cite, huddle, patents-collab, and sms} as conformant examples. We suspect that the exceptions indicate cases in which the classification algorithm was either overfitting the training set or failed to create a sufficiently optimized model in the high-dimensional space.
## TABLE 3.5

COMPARATIVE AUROC PERFORMANCE OF VCP

<table>
<thead>
<tr>
<th></th>
<th>AA</th>
<th>Katz</th>
<th>PA</th>
<th>HPLP</th>
<th>VCP 3U</th>
<th>VCP 3D</th>
<th>VCP 4U</th>
<th>VCP 4D</th>
</tr>
</thead>
<tbody>
<tr>
<td>calls</td>
<td>0.698</td>
<td>0.641</td>
<td>0.424</td>
<td>0.782</td>
<td>0.802</td>
<td>0.814</td>
<td>0.834</td>
<td><strong>0.849</strong></td>
</tr>
<tr>
<td>condmat</td>
<td><strong>0.663</strong></td>
<td>0.630</td>
<td>0.585</td>
<td>0.588</td>
<td>0.637</td>
<td>-</td>
<td>0.582</td>
<td>-</td>
</tr>
<tr>
<td>dblp-cite</td>
<td>0.794</td>
<td>0.791</td>
<td>0.773</td>
<td>0.841</td>
<td>0.830</td>
<td>0.847</td>
<td>0.843</td>
<td><strong>0.868</strong></td>
</tr>
<tr>
<td>dblp-collab</td>
<td><strong>0.697</strong></td>
<td>0.623</td>
<td>0.523</td>
<td>0.691</td>
<td>0.640</td>
<td>-</td>
<td>0.695</td>
<td>-</td>
</tr>
<tr>
<td>disease-g</td>
<td>0.930</td>
<td>0.907</td>
<td>0.820</td>
<td><strong>0.970</strong></td>
<td>0.923</td>
<td>-</td>
<td>0.964</td>
<td>-</td>
</tr>
<tr>
<td>disease-p</td>
<td>0.898</td>
<td>0.920</td>
<td>0.932</td>
<td>0.922</td>
<td>0.939</td>
<td>-</td>
<td><strong>0.951</strong></td>
<td>-</td>
</tr>
<tr>
<td>hepth-cite</td>
<td>0.826</td>
<td>0.794</td>
<td>0.766</td>
<td>0.838</td>
<td>0.836</td>
<td>0.846</td>
<td>0.845</td>
<td><strong>0.851</strong></td>
</tr>
<tr>
<td>hepth-collab</td>
<td>0.606</td>
<td>0.619</td>
<td>0.547</td>
<td>0.592</td>
<td>0.598</td>
<td>-</td>
<td><strong>0.622</strong></td>
<td>-</td>
</tr>
<tr>
<td>huddle</td>
<td>0.879</td>
<td>0.875</td>
<td>0.875</td>
<td>0.877</td>
<td>0.881</td>
<td>-</td>
<td><strong>0.888</strong></td>
<td>-</td>
</tr>
<tr>
<td>patents-collab</td>
<td>0.793</td>
<td>0.671</td>
<td>0.532</td>
<td>0.800</td>
<td>0.680</td>
<td>-</td>
<td><strong>0.816</strong></td>
<td>-</td>
</tr>
<tr>
<td>sms</td>
<td>0.642</td>
<td>0.581</td>
<td>0.472</td>
<td>0.714</td>
<td>0.735</td>
<td>0.730</td>
<td>0.791</td>
<td><strong>0.802</strong></td>
</tr>
</tbody>
</table>

Predictors are Adamic/Adar (AA), Katz, preferential attachment (PA), HPLP, and VCP. For VCP, U indicates that directionality is ignored and D indicates that it is considered. Bold text indicates the maximum value in the row.
<table>
<thead>
<tr>
<th></th>
<th>AA</th>
<th>Katz</th>
<th>PA</th>
<th>HPLP</th>
<th>VCP 3U</th>
<th>VCP 3D</th>
<th>VCP 4U</th>
<th>VCP 4D</th>
</tr>
</thead>
<tbody>
<tr>
<td>calls</td>
<td>0.000505</td>
<td>0.011465</td>
<td>0.000092</td>
<td>0.018005</td>
<td>0.031655</td>
<td>0.033091</td>
<td>0.033533</td>
<td>0.035127</td>
</tr>
<tr>
<td>condmat</td>
<td>0.000195</td>
<td>0.000183</td>
<td>0.000177</td>
<td>0.007763</td>
<td><strong>0.011917</strong></td>
<td>-</td>
<td>0.008589</td>
<td>-</td>
</tr>
<tr>
<td>dblp-cite</td>
<td>0.000314</td>
<td>0.000246</td>
<td>0.000234</td>
<td>0.016030</td>
<td>0.009207</td>
<td>0.015265</td>
<td>0.011427</td>
<td><strong>0.018137</strong></td>
</tr>
<tr>
<td>dblp-collab</td>
<td>0.008777</td>
<td>0.006723</td>
<td>0.003251</td>
<td>0.007772</td>
<td>0.007152</td>
<td>-</td>
<td><strong>0.009410</strong></td>
<td>-</td>
</tr>
<tr>
<td>disease-g</td>
<td>0.221299</td>
<td>0.193863</td>
<td>0.061694</td>
<td><strong>0.466716</strong></td>
<td>0.155165</td>
<td>-</td>
<td>0.444153</td>
<td>-</td>
</tr>
<tr>
<td>disease-p</td>
<td>0.629516</td>
<td><strong>0.676419</strong></td>
<td>0.673601</td>
<td>0.390074</td>
<td>0.552765</td>
<td>-</td>
<td>0.633316</td>
<td>-</td>
</tr>
<tr>
<td>hepth-cite</td>
<td>0.003967</td>
<td>0.003784</td>
<td>0.003225</td>
<td>0.054846</td>
<td>0.046140</td>
<td>0.059245</td>
<td>0.056244</td>
<td><strong>0.063387</strong></td>
</tr>
<tr>
<td>hepth-collab</td>
<td>0.008563</td>
<td><strong>0.009328</strong></td>
<td>0.005060</td>
<td>0.006123</td>
<td>0.007197</td>
<td>-</td>
<td>0.007156</td>
<td>-</td>
</tr>
<tr>
<td>huddle</td>
<td>0.000790</td>
<td>0.000746</td>
<td>0.000745</td>
<td>0.039914</td>
<td>0.039394</td>
<td>-</td>
<td><strong>0.046803</strong></td>
<td>-</td>
</tr>
<tr>
<td>patents-collab</td>
<td>0.006962</td>
<td>0.005678</td>
<td>0.001684</td>
<td>0.006735</td>
<td>0.005564</td>
<td>-</td>
<td><strong>0.007709</strong></td>
<td>-</td>
</tr>
<tr>
<td>sms</td>
<td>0.009410</td>
<td>0.009164</td>
<td>0.002986</td>
<td>0.011594</td>
<td>0.025206</td>
<td>0.026063</td>
<td>0.027073</td>
<td><strong>0.028201</strong></td>
</tr>
</tbody>
</table>

Predictors are Adamic/Adar (AA), Katz, preferential attachment (PA), HPLP, and VCP. For VCP, U indicates that directionality is ignored and D indicates that it is considered. Bold text indicates the maximum value in the row.
disease-p, hepth-cite, huddle, patents-collab, and sms as conformant examples. We suspect that the exceptions indicate cases in which the classification algorithm was either overfitting the training set or failed to create a sufficiently optimized model in the high-dimensional space.

In 7 of the 11 networks, VCP classification offers superior AUPR performance. In a slightly different selection of 7 networks, it offers superior AUROC performance. In some of the cases in which VCP offers the best performance, the differences are quite wide. In the sms network it offers an AUPR that is 2.3 times as high as the best competitor. In the condmat network, it offers AUPR 1.53 times the nearest competitor. In two of the networks in which VCP classification does not provide the best performance, HPLP does. As an interesting side note, when weights are removed as they were to obtain these results, HPLP does not always outperform the unsupervised predictors.

Figure 3.12 shows a closer look at the performance differences. The black dashed line represents the baseline performance of a random predictor. Across all the selected networks, VCP maintains high precision longer at increasing values of recall. This is especially important in link prediction, where high precisions are so difficult to achieve.

Despite the strong and competitive performance that the VCP method of supervised prediction exhibits, it is not our intent to present the most effective possible classification scheme. Our experiences with random forests, random subspaces, and other classification techniques suggest that the potential for improvement through feature selection and alternative classification algorithms is high. Another option for potential improvement is to concatenate VCP$^{3,r}$ and VCP$^{4,r}$ into a single feature vector. VCPs contain much information, and the task is
Figure 3.12. ROC and precision-recall curves for selected networks.
simply to determine how best to leverage it to achieve whatever goals are desired.

3.4.5.4 Timing

We used two different types of machines for timing. All feature computation and VCP generation was performed serially on a quad-core Opteron 2218 running at 2.6 GHz with 1 MB cache and 4 GB of main memory. Some classification runs required more than 4 GB of memory with the specified training set undersampling and algorithm parameters, so all WEKA classification was performed on a dual quad-core Xeon E5620 running at 2.4 GHz with 12 MB cache and 32 GB of main memory. To some degree, timings are implementation dependent, and though the implementations of predictors, feature computations, and VCPs are not naïve, we cannot claim that they are fully optimized. Figure 3.13 shows the results.

Adamic/Adar, $O\left(\frac{|E|}{|V|}\right)$ per prediction, and preferential attachment, $O(1)$ per prediction, perform very few operations to generate their output. They are so inexpensive to compute that they are invisible within the same scale as Katz and the supervised prediction methods for all three networks in Figure 3.13. We note that for all three networks the total time to perform supervised link prediction with VCPs is often less than that necessary for HPLP. Most of this is due to the expense of the Katz feature, which involves finding paths up to length 5 with the aid of memoization in our implementation. Based on these results, Adamic/Adar is clearly an effective and inexpensive option for a wide variety of networks, but VCP offers significant potential for performance enhancement.

Perhaps the most interesting conclusion lies in the inconsistency of the results in terms of the breakdown of time requirements for different components. Timing is related to the interplay between the specific algorithms involved, the local den-
Figure 3.13. Timing analysis for three selected networks. Network size information is available in Table 2.1 on page 12.

sities or global density of the graph, and the raw size of the graph in terms of the number of vertices and edges. Whether the bulk of time is consumed in feature generation, VCP computation, training, or testing varies greatly across networks as does the total time for any particular method.

The VCP implementations provided are slightly limited in efficiency because of the graph implementation to which they are tied in LPmade. With a more amenable supporting graph implementation and slight changes to the selection of data structures, we expect that it would be possible to decrease the running time of the VCP vector computation itself by a factor of at least 2. Nonetheless, the computation of even VCP^4,2 is competitive in terms of running time with much simpler and less effective path-based prediction methods.

The results in Figure 3.13 show that VCP is more efficient from a cost-performance standpoint than classification based on computing and combining simpler unsupervised predictors. Further, VCP computations are naturally parallel, and the extended times for the sms network include computing VCPs for
tens of millions of vertex pairs. The extended feature vector of VCP\textsuperscript{4,2} greatly increases training time, but feature selection or the application of different training algorithms or parameters could reduce this greatly, and training is parallelizable across bags.

3.4.6 Discussion

VCP is a new method of link analysis with solid theoretical roots. We presented evidence of its utility in some applications here, but there are many possible applications. It is useful for post hoc analysis of classification output, comparative analysis of network link structure, and it competes effectively with an existing recently published method, HPLP, often outperforming it by wide margins. In well-established networks with past observational data, VCP can serve as a sensitive change detection mechanism for tracking the evolving link formation process. In addition to link prediction and link analysis for the purpose of network growth modeling, VCP can be used for link or vertex pair clustering. For instance, examining VCP distributions may offer the ability to cluster links into functional categories. Its ability to handle multiple relations naturally extends its utility into many domains and offers an alternative to the practice of combining or discarding edge types or edge directionality.

We have provided code to perform VCP subgraph-to-element mappings. The VCP computations for directed and undirected VCP\textsuperscript{3,1} and directed and undirected VCP\textsuperscript{4,1} are currently loosely integrated into the LPmade link prediction framework. The mapping code and LPmade branch containing the software are available at \url{http://mloss.org/software/view/307/}. Most of the data sets are publicly available elsewhere, but we have also published public data sets to
so that our experiments can be repeated with the same longitudinal thresholds and thus the same network saturation.

3.5 Incorporating Longitudinal Data

Link formation and growth mechanisms are typically studied in terms of nodal attributes and topology [64]. More recently, the increased attention to link prediction has led to models that incorporate a variety of disparate data sources. [59] and [20] leverage and predict positive and negative links, [57] studies using surrogate information to construct a probabilistic graph when the topology is partially or entirely missing, [112] incorporates the geospatial proximity of individuals through time in cellular phone communication networks, and [98] similarly uses places individuals visit in common for augmenting topological link prediction.

In several studies of link prediction, authors have used longitudinal data, a series of events with timestamps of varying resolution that describes a network evolving through time [61, 68, 84, 98, 102], to perform link prediction without actually using the temporal component in their models. One recent work attempts to answer the question of when rather than if a link will form in the future [107]. Also distinct are works that perform general modeling based on temporal link analysis, which has a broader supporting literature [6, 44]. It is much less common to actually consider the temporal dimension as a factor in constructing link prediction models, and this is probably due to the difficulty of creating representations and models that effectively incorporate time [44]. [64] briefly considers methods of treating temporally delineated periods. The only work of which we are aware that directly addresses the problem of predicting links using temporal information is [1]. The authors apply factorizations to third order tensors, in which the complete
link structure is contained in two of the dimensions as it exists at discrete temporal intervals indicated by the third dimension, to achieve predictions. Alternatively, they describe a method of collapsing the temporal information into a weighted two-dimensional link representation on which they subsequently perform matrix factorizations to achieve predictions.

To our knowledge, ours is the first work to use multi-relational models to encode temporal information for the purpose of performing data-driven predictions with machine learning algorithms. The essential idea is to generate a vector of quantities describing the embedding of a link in all possible unique isomorphism classes. By encoding temporal information as multiple edge types in a multi-relational graph representation, we take advantage of the power of sophisticated multi-relational modeling techniques such as VCPs to incorporate time into our models.

3.5.1 Method

From the introductory paper on VCP [67], a vertex collocation profile (VCP), written VCP\textsuperscript{\textit{n,r}}\textsubscript{\textit{i,j}} is a vector describing the relationship between two vertices, \(v_i\) and \(v_j\), in terms of their common membership in all possible subgraphs of \(n\) vertices over \(r\) relations. A VCP element, VCP\textsuperscript{\textit{n,r}}\textsubscript{\textit{i,j}}(x) is defined as a distinct embedding of \(v_i\) and \(v_j\) within a particular isomorphism class of \(n\) vertices and is represented by a uniquely addressable cell in the VCP vector. It is possible to extend the VCP approach to multi-relational networks trivially by representing edges as an ordered set of bits. In the VCP algorithms, the procedures that determine the existence of a particular edge for single-relational data instead determine the value of the edge as the binary-coded integral value of the ordered set of bits for multi-relational
Figure 3.14. The first 16 elements in VCP$^{3,2}$.

data. As a result, excepting the additional cost of allocating and deallocating the storage necessary to count the additional multi-relational structural elements, the computational complexity of the multi-relational extension is no greater than for single-relational networks. Figure 3.14 shows the first 16 VCP elements for a 2-relational network, which is the complete set of elements discounting differences in $e_{s,t}$.

Given this method for extending VCPs to multi-relational data, we can capture the power of longitudinal data by encoding the snapshot in which an event occurs as its relation type. Consistent with the approach proposed by [102] and subsequently used by [1], we also divide our data into a set of discrete snapshots. Suppose a continuous stream of event records arrives with arbitrarily fine tempo-
ral granularity. We divide the stream into chunks $s_0$ to $s_t$ sized, possibly equally, according to selected criteria. In some cases, data is generated by a non-stationary process with periodicity, and so encapsulating the periodicity consistently might be an important criterion. From the chunks, we can create corresponding graphs $G_0$ to $G_t$ such that all events involving $i$ or $j$ in $s_a$ appear as links $e_{i,j}$ between vertices $v_i$ and $v_j$ in $G_a$. When we want a graph snapshot to be more temporally expansive than the chunks, we can either compute the minimum common supergraphs of multiple graph snapshots or equivalently compute a new graph snapshot containing the events within the desired bounds.

When performing supervised learning on longitudinal data, conceptual and practical complexities arise that are not present in standard supervised learning tasks. These complexities are covered in great detail in the expansive literature on data stream mining, and many of the topics endemic to all mining tasks on longitudinal data apply here. Nonetheless, using supervised classification to perform topologically informed link prediction in an underlying longitudinal data stream is particularly complex and encapsulates hidden parameters. Figure $3.15$ demonstrates the specific methodology employed within our experiments. Like other vanilla non-longitudinal classification problems, there exists a model construction step using training data and a performance evaluation or deployment prediction step using testing data. The basic application of supervised learning to link prediction on longitudinal data is depicted in $3.15a$ as a reference. Each color in each of the training and testing time lines represents a separate network snapshot.

In the basic application, there are four snapshots: the two blue snapshots are used for constructing a feature vector. In the training time line, the red snapshot provides labels for model induction. In the testing time line, the red
Figure 3.15. Supervised classification approaches involving different temporal snapshot distributions.
snapshot is the newly observed data on which we test our model. Importantly, the red snapshot in testing must always be the same for fair model comparison regardless of changes to any other snapshots. Though slightly less important, we also maintain the red snapshot in training since varying the snapshot is likely to cause significant changes in both the prior and conditional probabilities of the link formation labels. We likewise make an effort to reflect the duration of the red snapshot in testing with the duration of the red snapshot in training to give the model the best chance at facing the same distribution on which it was trained.

Figures 3.15b and 3.15c demonstrate supervised learning with two temporal relations, and this requires six distinct snapshots total for training and testing. Figures 3.15d and 3.15e demonstrate supervised learning with three temporal relations, and this requires eight distinct snapshots. In our temporal models, we handle data granularity in two different ways. In the first, snapshot granularity remains consistent throughout the data to model. In the second, we decrease snapshot granularity exponentially with each relation and emphasize the most recent events by making that snapshot relatively short. The depictions in Figures 3.15b and 3.15c are of evenly divided relations. Those in Figures 3.15c and 3.15e are of exponentially divided relations. In Figure 3.15d, we suppose that $G_{t-6}$ is derived from the oldest available data in the stream, so we cannot extend the oldest snapshot to its full length. In such cases, it may also be wise to limit the length of the oldest testing snapshot to present the closest possible testing data to the model, and such a modification would not constitute leakage [51].

Smaller relational periods effectively place greater emphasis on occurrences in the multi-relational VCP method. Links from short periods and links from long periods both receive their own relation, so short periods with few events afford the
events greater discriminative potential. Links occurring within the same relation are also guaranteed to occur closer together in time, which may decrease noise and increase the clarity of temporally proximate evolutionary trends. In Figures 3.15b and 3.15d, events that recently occurred are given the same weight as those that occurred in the more distant past. In Figures 3.15c and 3.15e, recent events are effectively given more weight by placing fewer, more proximate events in the same relation.

3.5.2 Data

We draw our conclusions from four network data sets of moderate size. Three represent collaboration events among authors in physics or computer science. The collaboration events connect scientists when they co-author papers. The fourth represents market baskets, sets of products purchased together during checkout. The purchase events connect products when students, faculty, and staff buy them together in the same transaction at the campus convenience store at the University of Notre Dame. Table 3.7 presents some basic information about the data in the longitudinal context.

The table also indicates the periods into which events are divided. For instance, we divide the longitudinal condmat data into 6 periods each spanning 12 months. The division of data into periods is essentially arbitrary but represents our attempts to balance reasonable data sufficiency in each period with the production of enough periods to allow for experiments involving comparison of the number of snapshots and snapshot granularity.

Figure 3.16 provides the domain and distribution of events in time for each of the networks. The collaboration networks exhibit a general increase in the rate of
Figure 3.16. Distribution of events by period.
growth. The growth in the dblp-collab network is consistent with [60], but we note that the anomaly in 1994-1996 is inconsistent, and upon further examination of the data, we can report no explanation. The hepth-collab network data reduces in volume for 2002 due to incomplete information for that year. The product co-purchasing network shows a cyclical reduction during summer months, and we feel safe attributing this to the presence of fewer students on campus since the store resides in the student center.

3.5.3 Experiments

In addition to demonstrating the performance boost that incorporating time can offer with a directly comparable baseline that does not include time, we examine several considerations of link prediction with temporal data in more detail. We consider issues related to temporal granularity and recency as they apply to these data sets with an empirical investigation. This culminates in a demonstration that including time can offer marked increases in predictive accuracy even

<table>
<thead>
<tr>
<th>Network</th>
<th>Agents</th>
<th>Relationships</th>
<th>Periods x Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>condmat</td>
<td>17,000</td>
<td>110,538</td>
<td>6 × 12 months</td>
</tr>
<tr>
<td>dblp-collab</td>
<td>367,725</td>
<td>2,088,710</td>
<td>13 × 24 months</td>
</tr>
<tr>
<td>hepth-collab</td>
<td>8,381</td>
<td>40,736</td>
<td>6 × 24 months</td>
</tr>
<tr>
<td>huddle</td>
<td>4,243</td>
<td>993,288</td>
<td>11 × 3 months</td>
</tr>
</tbody>
</table>
over methods that already powerfully exploit network topology for link prediction.

3.5.3.1 Analyzing Split Points

Approaches that divide continuous longitudinal network data into discrete, bounded units face several potential fundamental choices and parameters. These include whether to maintain a consistent snapshot size, how many snapshots to use, and at what temporal locations data should be divided. We conduct a simple experiment to determine the optimal temporal location at which to divide these data sets, at least in the case of two snapshots. This experiment also tests the hypothesis that greater focus on recent events is beneficial, and the results lead to cogent theories about how to parameterize more snapshots.

Figure 3.17 shows the performance achievable at all possible splits for two snapshots given periods. Take $s_i$ as a value in the domains of the sub-figures and $s_{t+1}$ as the final chunk in the data stream as indicated in Table 3.7. Then in training the first snapshot includes chunks $[s_0, s_i)$, and the second snapshot includes $[s_i, s_{t-1}]$. The penultimate chunk, $s_t$, is reserved for training labels. For this experiment, to ensure that training and testing feature data volumes are constant, the first testing snapshot entirely subsumes the first training snapshot. In testing, the first snapshot includes chunks $[s_1, s_i+1)$, and the second snapshot includes $[s_{i+1}, s_t]$. The final chunk, $s_{t+1}$, is reserved for testing labels. The horizontal lines correspond to baselines in which supervised models are built using VCPs without any temporal discrimination.

The time-resolved multi-relational VCPs almost always outperform their non-longitudinal single-relational counterparts for some split value. The only counterexample is condmat for which VCP$^{3,1}$ outperforms VCP$^{3,2}$. In all other cases,
Figure 3.17. Performance over varying temporal split locations.
both 3-vertex and 4-vertex VCPs perform equally well or much better when pro-
vided with multiple relations encoding the time of occurrence. We additionally
observe that while VCP\(^4\) outperforms VCP\(^3\) in all cases except condmat and
nearly ties for hepth-collab AUPR and huddle AUROC, the addition of tempo-
ral discrimination into the models appears to allow 4-vertex multi-relational VCPs
significantly greater explanatory power across all networks. Consistent with our
hypothesis regarding the significance of recency, performance increases as the more
recent snapshot encompasses fewer, more temporally proximate events. The con-
firmation is particularly strong, because the split locations are both highly biased
toward recency and still offer a great performance benefit over placing all data in
a single snapshot. condmat and huddle include only \(s_{t-1}\) and \(s_{t-2}\) in the second
snapshot for the highest performing split, and dblp-collab and hepth-collab
include only \(s_{t-1}\) in the highest performing split.

3.5.3.2 Effectiveness of Temporal Data

Having demonstrated the importance of splitting periods and recency, we con-
tinue by providing the comparative performance of the temporally non-resolved
baseline with that of evenly and exponentially spaced snapshots in Tables 3.8 and
3.9. Cells with dashes indicate data sets for which there are insufficient chunks to
meaningfully differentiate even from exponential snapshot sizes with three snap-
shots. Gray cells indicate cells for which the computational cost was too high to
allow completion, a problem we will address momentarily. We consider it critical
to compare the different variations using the same volume and quality of data, so
we deviate slightly from the method displayed in Figure 3.15 in a couple minor
ways. First, all methods, whether evenly spaced or exponentially spaced use all
of the data to account for explanations of differences in performance due to variations in data. In both even and exponential cases, when there are too few or too many chunks for assignment, the oldest chunk deviates from its ideal value. This decision is based on the premise substantiated intuitively and by Figure 3.17 that this is the chunk of least significance. Second, to help account for potential periodic biases, the most recent snapshot in the exponential results starts with two rather than one underlying chunk.

These results show that temporally aware predictors universally outperform non-temporal baselines using the most directly comparable method as a baseline. Gains are consistently large when moving from no temporal awareness to division of time into two snapshots with $\text{VCP}^n_2$. They are less consistent moving from $\text{VCP}^n_2$ to three snapshots with $\text{VCP}^n_3$. Particularly, we note that condmat and hepth-collab do not offer significant gains or even suffer losses when moving to three snapshots. Here we point out the difference in data volume and cite the curse of dimensionality. As the number of distinct temporal-structural elements increases, the amount of information contained within any one element is reduced as the data falls into increasingly many fine-grained buckets. With the voluminous dblp-collab data set, there is still sufficient data to lend statistical credence to the greater number of features. Values for huddle require too much time and memory to reasonably compute, which is a limitation of this method that we address and overcome in the following section, but they also indicate sufficient data volume to support high temporal resolution.
<table>
<thead>
<tr>
<th>Network</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>3exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>condmat</td>
<td>0.637</td>
<td>0.670</td>
<td>0.671</td>
<td>-</td>
</tr>
<tr>
<td>dblp-collab</td>
<td>0.640</td>
<td>0.749</td>
<td>0.770</td>
<td>0.779</td>
</tr>
<tr>
<td>hepth-collab</td>
<td>0.598</td>
<td>0.738</td>
<td>0.729</td>
<td>-</td>
</tr>
<tr>
<td>huddle</td>
<td>0.881</td>
<td>0.933</td>
<td>0.929</td>
<td>0.936</td>
</tr>
</tbody>
</table>

(a) VCP\textsuperscript{3,r}

<table>
<thead>
<tr>
<th>Network</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>3exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>condmat</td>
<td>0.582</td>
<td>0.646</td>
<td>0.653</td>
<td>-</td>
</tr>
<tr>
<td>dblp-collab</td>
<td>0.695</td>
<td>0.789</td>
<td>0.807</td>
<td>0.817</td>
</tr>
<tr>
<td>hepth-collab</td>
<td>0.622</td>
<td>0.732</td>
<td>0.720</td>
<td>-</td>
</tr>
<tr>
<td>huddle</td>
<td>0.888</td>
<td>0.926</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) VCP\textsuperscript{4,r}

95
### TABLE 3.9

AUPR PERFORMANCE FOR LONGITUDINAL VCP PREDICTIONS

<table>
<thead>
<tr>
<th>Network</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>3exp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>condmat</td>
<td>0.011917</td>
<td>0.010643</td>
<td>0.010352</td>
</tr>
<tr>
<td></td>
<td>dblp-collab</td>
<td>0.007152</td>
<td>0.009161</td>
<td>0.009931</td>
</tr>
<tr>
<td></td>
<td>hepth-collab</td>
<td>0.007197</td>
<td>0.010280</td>
<td>0.011131</td>
</tr>
<tr>
<td></td>
<td>huddle</td>
<td>0.039394</td>
<td>0.065830</td>
<td>0.067993</td>
</tr>
</tbody>
</table>

(a) VCP$^{3,r}$

<table>
<thead>
<tr>
<th>Network</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>3exp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>condmat</td>
<td>0.008588</td>
<td>0.010103</td>
<td>0.009433</td>
</tr>
<tr>
<td></td>
<td>dblp-collab</td>
<td>0.009410</td>
<td>0.012389</td>
<td>0.014008</td>
</tr>
<tr>
<td></td>
<td>hepth-collab</td>
<td>0.007157</td>
<td>0.009375</td>
<td>0.009024</td>
</tr>
<tr>
<td></td>
<td>huddle</td>
<td>0.046803</td>
<td>0.071811</td>
<td>-</td>
</tr>
</tbody>
</table>

(b) VCP$^{4,r}$

96
### 3.5.3.3 Data Volume Reduction

The principal problem with this method is the volume of data generated by the multi-relational VCP method, especially VCP$^{4,3}$. Training classification models with ideal parameters is expensive due to the large number of features. We take a couple approaches to address this problem. First, we identify VCP elements for which there is no representative membership in the training set, because a feature of constant value provides no discriminative value. This method has no drawbacks since it sacrifices no information, so we were dismayed to find that it did not sufficiently reduce feature volumes to allow fast classification.

The dblp-collab network has potential links that populate 12,436 VCP$^{4,2}$ elements at least once with nearly 7 million instances, or 88,744 instances after undersampling the negative class to 3:1 representation. The huddle network has potential links that populate 16,635 VCP$^{4,2}$ elements at least once with nearly 11 million instances, or 258,536 instances after undersampling the negative class to 3:1 representation. Training the model for the dblp-collab data set with VCP$^{4,3}$ required 80 GB of in WEKA and just over 10 days with the 3:1 undersampling. Training was out of reach for huddle with the VCP$^{4,3}$ data set.
The first problem that we must overcome in this feature selection is one of computability rather than model efficacy. Table 3.10 shows the growth in the number of VCP elements with 3 and 4 vertices. If we assume 8-bytes of storage per feature value in 10 million training instances, a reasonable amount for neighbor-of-neighbor predictions in a moderately sized social network, then the in-memory data footprint for VCP\(^{4,3}\) is 5.2 TB. Clearly we cannot simply load this data set into memory with WEKA or any other standard tool and run a feature analysis. Processing the C4.5 training data file one column at a time would require reading more than 100 PB of I/O. Reading the file once and using output duplication and process substitution would require thousands of simultaneous active processes and completely overwhelm the process scheduler.

The solution lies itself in the curse of dimensionality. We observe that with increasing values of \(n\) and \(r\), the number and discrimination of VCP elements increases while the number of distinct non-zero values decreases within each element. Figure 3.18 shows the scaling of distinct values across VCP\(^{4,2}\) and VCP\(^{4,3}\). In all cases, though there are more than 60 times as many features with four relations, the sum of the cardinalities of the sets of distinct values across the features is less than doubled. We use this fact to implement a single-pass ordered map representation of each feature in the data set that associates values with their frequency of occurrence. The representation is highly compact: the huddle data set requiring more than 80 GB of memory in matrix representation requires only 1.4 GB in this compressed representation. The ordered map in effect implements a fast, non-contiguous counting sort that allows for nearly instantaneous computation of feature selection methods such as information gain.

Due to the high volume of both instances and data, memory-hungry feature
Figure 3.18. Scaling trends in distinct values of VCP elements.
selection methods are not feasible. We choose a very simple and fast feature ranking method, information gain, that is aligned well with the underlying model construction technique. It has light memory requirements and does not consider feature subsets. The disadvantage is that it may discard features that only add information when coupled with other features. In Tables 3.11 and 3.12, we provide the results of selecting the 1000 features with the highest information gain from the data sets built with three snapshots to determine whether they are competitive with classification using complete training sets.

The tables show promising results for this “Offline” method with the exception of the condmat data set. Models built on only the 1000 best features are within 2% AUROC and AUPR of the data sets built from complete training sets. More importantly, the model construction time for the largest data set for which training with all features is even tractable, dblp-collab, is reduced by a factor of more than 100 from over 10 days to under 2 hours. The memory requirements in training likewise decrease from over 80 GB to under 2 GB. Clearly this method is effective at reducing data volumes, memory requirements, and training times, but as proposed, it unfortunately still requires the entire data set to exist somewhere. Therefore it does not limit the cost in disk capacity for the enormous storage of features from the VCP vector that go unused. Constructing the full feature sets also requires that the VCP algorithm needlessly allocate and deallocate vast quantities of memory for elements that result in output that will be ignored.

Computation of VCP\(n,r\) is nearly equivalent for all values of \(r\) when \(n\) is held constant, because queries for edge existence simply become queries for edge value. The only additional costs associated with increasing the number of relations is the cost required to allocate and deallocate temporary storage for the vector and
<table>
<thead>
<tr>
<th>Network</th>
<th>3</th>
<th>3exp</th>
<th>3 Offline</th>
<th>3exp Offline</th>
<th>3 Online</th>
<th>3exp Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>condmat</td>
<td>0.653</td>
<td>-</td>
<td>0.616</td>
<td>-</td>
<td>0.611</td>
<td>-</td>
</tr>
<tr>
<td>dblp-collab</td>
<td>0.807</td>
<td>0.817</td>
<td>0.807</td>
<td>0.821</td>
<td>0.807</td>
<td>0.821</td>
</tr>
<tr>
<td>hepth-collab</td>
<td>0.720</td>
<td>-</td>
<td>0.725</td>
<td>-</td>
<td>0.717</td>
<td>-</td>
</tr>
<tr>
<td>huddle</td>
<td>0.929</td>
<td>0.941</td>
<td>0.930</td>
<td>0.941</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first two columns are baselines without feature selection. Header values of “3” indicate division of data into 3 even periods while “3exp” indicates division of data into periods that increase in size with decreasing recency. Header values of “Offline” indicate using all of the data to select the 1000 features with the highest information gain. Header values of “Online” indicate using a preselected subset of data to make the same selection. Gray cells are too expensive to compute, and dashed cells do not have sufficient data to allow the distinction between “3” and “3exp”.

TABLE 3.11

COMPARISON OF VCP⁴r AUROC PERFORMANCE BEFORE AND AFTER FEATURE SELECTION
<table>
<thead>
<tr>
<th>Network</th>
<th>3</th>
<th>3exp</th>
<th>3 Offline</th>
<th>3exp Offline</th>
<th>3 Online</th>
<th>3exp Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>condmat</td>
<td>0.009433</td>
<td>-</td>
<td>0.008580</td>
<td>-</td>
<td>0.008342</td>
<td>-</td>
</tr>
<tr>
<td>dblp-collab</td>
<td>0.014008</td>
<td>0.015229</td>
<td>0.013963</td>
<td>0.015619</td>
<td>0.014059</td>
<td>0.015495</td>
</tr>
<tr>
<td>hepth-collab</td>
<td>0.009024</td>
<td>-</td>
<td>0.009335</td>
<td>-</td>
<td>0.008619</td>
<td>-</td>
</tr>
<tr>
<td>huddle</td>
<td>0.073406</td>
<td>0.092726</td>
<td>0.074259</td>
<td>0.093402</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first two columns are baselines without feature selection. Header values of “3” indicate division of data into 3 even periods while “3exp” indicates division of data into periods that increase in size with decreasing recency. Header values of “Offline” indicate using all of the data to select the 1000 features with the highest information gain. Header values of “Online” indicate using a preselected subset of data to make the same selection. Gray cells are too expensive to compute, and dashed cells do not have sufficient data to allow the distinction between “3” and “3exp.”
the cost to print the additional output. Although even these costs will become significant with high enough values of $n$ or $r$, the practical problem with using the VCP for many relations instead lies in the volume of temporary storage necessary.

We address this by adapting the offline feature selection technique previously described to be online by performing its selection prior to generating the complete data set. This has the advantage of allowing for more vertices or relations than otherwise would be possible. We use the same general approach for feature selection as for the offline version already described, except that we select a pool of instances from training for the feature selection algorithm to consider first and then filter the training set and the testing set to include only those instances. Since instances for feature selection come from training, we can provide the positive class instances with an equal number of negative class instances to the prediction algorithm first. First, we use the predictor to compute the full VCP vector for the selected instances, which requires little computational time because those instances are much fewer in number. Information gain or any other feasible feature selection output is computed on these instances, and this allows for the creation of a mapping between the original VCP element address and the new address in the limited, feature-selected output. Instances from the original, balanced set are pared to include only the selected features, and VCP vector generation for subsequent instances need not allocate more memory than is necessary for the selected number of features. The mapping from original element address to address in feature selection is a low-cost constant time operation. The results of this method are displayed in Tables 3.11 and 3.12 as “Online”.

103
3.5.4 Discussion

Longitudinal data describing social and other networks is an increasingly available commodity. Though much of the work in link prediction literature very appropriately uses temporal information to form snapshots for training and testing, only in one case has research focused on incorporating this information into the prediction methodology. Our work is the first attempt of which we are aware at encoding temporal information in a multi-relational graph representation to perform data-driven predictions with machine learning algorithms. Our results demonstrate that the addition of temporal information in model construction can create models that outperform comparable baseline counterparts that do not take advantage of such information. The results on all four network data sets also strongly suggest that increased emphasis on recent events is an important consideration in temporally aware models.

In addition to demonstrating the usefulness of incorporating temporal information in models, our results show the performance potential of the VCP approach in multi-relational contexts. The AUROC and AUPR values that we obtained exceed those published in [67] with the same networks, and those values themselves offer exemplary performance. Nonetheless, it is possible to extend the basic approach of encoding temporal data in a multi-relational graph representation with a variety of other sophisticated link modeling techniques capable of accepting multi-relational data. The multi-relational path counting approach of [106] is an example of another method that could be extended to include temporal data. By searching for paths constructed of statistically indicative sequences of links from varying snapshots, it could also take advantage of multi-relationally encoded temporal information.
We have also demonstrated that the high dimensionality introduced by VCPs in multi-relational networks can be reduced by feature selection without losing the performance gains of the extra relations, temporal or otherwise. We surmise that this places 3-vertex VCPs up to VCP$^{3,7}$ and 4-vertex VCPs up to VCP$^{4,4}$ within reach for practical and timely data analysis. At the same time, our results suggest that the biggest benefit from temporal information comes from initially introducing it. The cost of increased granularity must naturally come with diminishing returns up to some asymptotic bound corresponding to the amount of information contained in the raw data.
Link prediction is a challenging domain within which to evaluate and interpret performance because of its extreme class imbalance. There is no standard method of evaluation. This makes it difficult to compare statements made in one paper to statements made in another without obtaining or implementing both methods and comparing them against each other under the same conditions. We examine some of the many evaluation methods proposed, occasionally highlighting weaknesses, and propose a standard method informed by class imbalance research within the data mining community. Sound evaluation matters in any domain, and link prediction includes all the issues of proper evaluation in binary classification, but it also presents several unique challenges.

First, we consider the issue of directionality, for which there is no analog in other classification tasks. In undirected networks, the same method may predict two significantly different results for a link between $v_a$ and $v_b$ depending on the order of input vertices, but there can only be one final judgment of whether the link will form. Next we consider the issue of saturation. Networks are dynamic, but if we have not observed sufficient information about the network to understand its basic structure and link formation mechanism, it may be foolhardy to commit to a particular predictor based on early results. To demonstrate to others sufficient context for comparative predictor performance results, we must provide param-
eters regarding network observation and convergence. Finally, we examine the extreme imbalance of link prediction to illustrate why many common approaches to test set sampling, a ubiquitous practice in this domain, lead to problems.

All of these issues stand in the way of producing fair, comparable results across published methods. Perhaps even more importantly, they interfere with views of performance that indicate what we can really expect of our prediction methods in deployment scenarios. Some of the analysis and conclusions may seem obvious, but the published literature is rife with the issues we highlight making it difficult to understand results. We seek to provide a reference for important issues to consider and a set of coherent standards as recommendations to those performing link prediction research.

4.1 Past Methods

Liben-Nowell and Kleinberg employ an original sophisticated validation strategy designed to ignore the influence of new nodes on prediction evaluation [64]. They develop a set of core nodes, nodes with a sufficient number of existing links in both the training interval and the testing interval. This sufficient number is a parameter to the evaluation. Then, they intersect the set of links that arise in the testing interval with the set of potential links within the core nodes. Finally, they intersect this set, which has size $n$, with the set of top $n$ predictions from the link predictors. The final step is to perform the same intersection with randomly ordered predictions. The output score is the ratio of the result for the predictor under evaluation to that of the random predictor.

In [100], data is non-longitudinal. 10% of linked pairs were randomly held out along with an artificially equal number of unlinked pairs, and results were evalu-
ated in terms of accuracy averaged over 10 trials. The authors of [24] randomly held out a sample of edges from the original networks. [84] uses longitudinal data and creates two equal-sized intervals, one for generating features and the other for generating labels, and uses recall as its performance measure according to Figure 2 in that paper, though the term accuracy is used. They claim to limit their predictions to those nodes that exist in the training period in a manner consistent with [64], but they do not specify the threshold degree. It is not clear whether the same comparison to a random predictor is involved, nor is it clear what threshold is placed on predictions, though the threshold may be constructed to be consistent with Libel-Nowell. In [87], the authors employ a common approach from time series modeling for supervised learning in which training is performed on two intervals with features from the first and labels from the second, and the two original intervals are subsequently converted to a single interval in which testing features are constructed. A following third interval is used for testing labels. This is the general approach we will advocate. They employ precision in a parameterized set of top-ranked prediction instances as their metric. Although they do not make it clear in the paper, the authors of [4] undersample the data set and then perform cross validation on it. They report their results in terms of accuracy, precision, recall, F-measure, and squared error, but the extraordinarily high values that they achieve, virtually impossible to achieve in a domain as extremely imbalanced as link prediction, are due to the artificial modification of the testing data prior to the construction of the cross validation folds. In [111] the authors contribute only a sample of the negative instances to their test set.

We note that evaluation in the link prediction domain seems rife with errors. First, link prediction should ideally be tested longitudinally. Sampling “test
edges” from the occasional static snapshot of a network may be necessary to get an idea of performance on that type of network when additional data is unavailable. In general, it should be avoided when the goal is to compare prediction methods. Sampling edges removes information from the original “real” network in unpredictable ways \[105\], and the removed information has the potential to affect prediction methods differently. Further, testing based on a random sample of edges implicitly assumes that a random process is responsible for link formation. In virtually all real-world networks, there is reason to expect that this assumption is entirely errant.

Second, it is not uncommon for link prediction papers to report results based on a modified test set \[4, 100, 111\]. As researchers familiar with high skew are aware, modifying the data distribution on which testing is performed generates uninterpretable results. The distribution of the resulting testing data no longer presents the same challenges as the real-world distribution, and performance measures in testing no longer reflect the real capabilities and limitations of the model.

4.2 Data and Experimental Setup

We report all results on condmat. As a reminder, this data set is constructed from a sequence of collaboration events in the condensed matter physics community. Each collaboration of \(k\) individuals forms an undirected \(k\)-clique with weights in inverse linear proportion to \(k\). The point of this work is not to illustrate the superiority of one method of link prediction over another but instead to show that the described effects and arguments have real impacts on performance and evaluation. Only one data set is necessary, because if the effects pertain in at least one network, then they may exist in others and must be considered.
We make our points selectively using predictors from three different modeling approaches. The preferential attachment predictor [86] uses degree product and represents predictors based on node statistics. The Adamic/Adar predictor [3] represents the family of common neighbors predictors. The PropFlow predictor [68] represents the family of predictors based on paths and random walks.

4.2.1 Recommendation Versus Query

There are two fundamentally different ways to generate test sets in link prediction. The first is to create a set of potential links by examining the predictor network and selecting all pairs for which no edge exists. Positives are those among the set that subsequently appear in the testing network, and negatives are all others. The second is to use the testing network to generate the set of potential links. Positives are those that exist in the testing network but not in the training network, and negatives are those that could exist in the testing network but do not. The difference lies in whether or not the prediction method is faced with or penalized for links that involve nodes that do not appear in the predictor network.

The choice we should make depends on how the problem is posed. In the recommendation problem, we are faced with returning predictions in order of confidence, so new nodes in the testing network are irrelevant. Although we could predict that an existing node will connect to an unobserved node, we cannot possibly predict what node the unobserved node will be. In the query problem, we are faced with answering a request about specific node pairs, so the ability to handle new nodes is an important aspect of performance. Predictors might offer a constant response to all queries regarding unfamiliar nodes. Some prediction methods support natural extensions for unknown nodes. For instance, preferential
attachment can be adapted to assume a degree of 1. In supervised classification, features for unknown nodes become missing values and the algorithm must support such values. We address the ranking problem.

4.2.2 Evaluation Measures

Evaluation measures fall into one of two categories: fixed-threshold measures and performance curves. All fixed-threshold measures require that some estimate of a reasonable threshold be available. Fixed-threshold measures may rely on different types of thresholds: based on prediction score, based on percentage of instances, or based on number of instances. In link prediction specifically, many link prediction methods do not produce scores with any reasonable correspondence to likelihood. It is rarely sensible to say two vertices with a degree product of 10,000 are 10 times as likely to form a new link as two others with a degree product of 1000.

It is appropriate to use top-$k$ percentages only when $k$ is provided by the domain. When using an absolute number of instances $k$, if the data does not admit a trivially simple class boundary, we set predictors up for failure by presenting them with class ratios of millions to one and taking only some few $k$. Further, if $k$ is small and the number of positives is small, the measure is unstable.

Therefore, while accuracy [4, 47, 59, 108], precision [4, 87, 108, 111], recall [4, 47, 108], and top-$k$ equivalents [47, 87, 111] are used commonly in link prediction literature, we caution trusting any results that come only in terms of fixed thresholds [4, 47, 108]. Sometimes results are expressed in terms of improvement factors or ratios [64] of fixed threshold measures by dividing the performance numbers of competing methods by the performance number of some established
baseline. The disadvantage of this approach is that information about absolute predictive capability is lost, and it is easy to use improvement factors to report stunning numbers that actually indicate very little about performance. A 1000x factor is not so impressive when the original precision is $10^{-7}$. The minimal number of positives may even make the random model so unstable that random models can easily achieve large factors of improvement over themselves.

In the absence of fixed thresholds, curves that sweep through the entire space of trade-offs between correct and incorrect predictions such as the receiver operating characteristic (ROC) curve and derived curves like cost curves [33] and precision-recall curves [30] provide alternatives. They are especially popular when the class distribution is highly imbalanced and hence are commonly used in link prediction evaluation [39, 68, 111]. Some curves also admit scalar measures, such as the ROC area (AUC) or PR area (AUPR).

In typical binary classification tasks, class ratios are approximately balanced. We expect the probability of randomly selecting a positive instance to be approximately equal to the probability of randomly selecting a negative instance. As a result, we can calculate expectations for baseline classifier performance. For instance, the expected accuracy ($\frac{TP + TN}{TP + FP + TN + FN}$), precision ($\frac{TP}{TP + FP}$), recall ($\frac{TP}{TP + FN}$), AUC, and AUPR of a random classifier, an all-positive classifier, and an all-negative classifier are 0.5. Binary classification problems that exhibit class imbalance do not share this property. The expectation for each of these values diverges for random, all-positive, and all-negative classifiers. Accuracy is problematic because its value for all-negative predictors converges to unity even as correct classification of rare positive instances represents high relative interest. Since classification is an exercise in optimizing some measure of performance, we
must not select a measure of performance that optimizes toward a useless result. ROC curves offer a baseline random performance of 0.5 and penalize all-positive and all-negative predictors. Precision and PR curves offer baseline performance calibrated in proportion to the percentage positive. Both are useful for producing reasonable class boundaries with imbalanced class distributions.

4.3 New Ideas

Our goal is to propose a standardization of link prediction evaluation so that, when practicable, link prediction literature is published with results that are both meaningful and comparable. In the following subsections, we will describe the evaluation strategy and metrics that we endorse along with an explanation of why they are appropriate.

4.3.1 Directionality

In directed networks a prediction is uniquely specified by an ordered pair of vertices with order implied by edge directionality. In undirected networks no ordering is implied, so there are potentially two different prediction outputs for
each pair of edges. Some predictors, such as those based on node properties or common neighbors, produce the same prediction output irrespective of ordering. Others, such as those based on walks, implicitly depend on notions of source and target even if with undirected edges.

Contemplate Figure 4.1 with the goal of predicting a link between \(v_a\) and \(v_b\). Consider the percentage of two-hop paths starting from \(v_a\) that reach \(v_b\) versus the percentage of two-hop paths starting from \(v_b\) that reach \(v_a\). All two-hop paths originating at \(v_b\) reach \(v_a\) whereas only a third of the two-hop paths originating at \(v_a\) reach \(v_b\). Topological prediction outputs may diverge whenever \(v_a\) and \(v_b\) are in different automorphism orbits within a shared connected component.

Any function that maps two values to one will resolve the issue, but it is important for reproducibility not to forget or neglect this question when describing results. When a method respects notions of source and target, we must specify how we resolve this issue. It is not acceptable to produce a ranked list of scores that includes predictions with each node alternately serving as source and target, as it is done for the undirected data set in [68]. To emphasize this empirically, we computed ROC areas for two methods using the PropFlow predictor. The first method includes a prediction in the output for both underlying orderings, and the resulting area is 0.610. The second method computes the arithmetic mean of the predictions from the two underlying orderings to produce one final prediction for the rankings, and the resulting area is 0.625.

4.3.2 Saturation

Some networks may always be observable through some form of topological exploration, such as the Internet, the worldwide web, and electricity grids. Many
are observable only through events that indicate the presence of links. In the latter, one must collect events to construct an approximation of the underlying structure. In these circumstances, it is important to collect a sufficient number of events to accurately describe the unobservable network of agents that is actually generating the events. If an insufficient number of events is collected, then any statistics computed on the network will be inaccurate and their predictive power will be greatly diminished relative to the convergent information. After a sufficient number of events is collected, topological statistics converge and the network is said to be saturated. In the case of link prediction, we need such a period from which to gather the measures for either unsupervised prediction or features. We also need a successive period, not necessarily saturated, to either evaluate our unsupervised methods or provide training labels for the classifier. Finally, if we want to evaluate the classifier, we require a third period, again not necessarily saturated.

It is also important how the observational intervals are defined. If these mea-
sures have not stabilized, then any results will be reporting the efficacy of the features in these terms instead of in terms of the actual performance potential of the feature on the saturated network. Prediction methods may be hampered by incomplete saturation to varying degrees. For instance, path-based predictors may be much more affected than degree-based predictors since the densification may have more complex effects than a simple increase in degree. This difference in information may thus cause serious but variable inaccuracies in predictor estimations. Perhaps even more significantly, in unsaturated networks, a large portion of positive labels from subsequent periods may be artifacts. Such “new” edges may actually result from latent edges rather than newly formed relationships in that period. For this reason, we contend that reporting the amount to which topological statistics have converged is an important part of good literature on link prediction. We show these results in Figure 2.2 for all networks for which we present predictor performance data.

In a typical supervised learning task, we are given a unified set of data with each instance of the form $(\vec{x}, y)$. To convert the link prediction task into this format, we have to select two values $\tau_x$ and $\tau_y$. These values correspond to the lengths of two adjacent periods over which we want to record events to construct networks. From the first network, $G_x = (V_x, E_x)$, constructed from $t_0$ to $t_0 + \tau_x$, we extract topological measures, and potentially node attributes, that serve as features for each pair of nodes $(v_i, v_j)$. From the second network, $G_y = (V_y, E_y)$, constructed from $t_{\tau_x + 1}$ to $t_{\tau_x + \tau_y}$, we examine $(v_i, v_j)$ to discover whether $e_{i,j}$ exists and determine the class label. This yields a data set in the standard $(\vec{x}, y)$ format with $|V_x|^2 - |E_x|$ instances.

The two parameters $\tau_x$ and $\tau_y$ have important but predictable influences on the
subsequent success of the resulting models. We can expect that increasing $\tau_x$ will increase the quality of topological measures as the network reaches saturation. This is the point at which $\tau_x$ is large enough that the observed events form a topology that closely reflects the underlying static network. As $\tau_x$ approaches this point, the topological measures converge to their actual unobservable static network values, thus allowing improved individual predictive capacity. We can expect that increasing $\tau_y$ will increase the number of positives. We investigate $\tau_x$ on the phone network in Figure 4.3.

Increasing $\tau_x$ has the expected result. The strength of the predictors increases greatly from $\tau_x = 2$ weeks to $\tau_x = 5$ weeks and again from $\tau_x = 5$ weeks to $\tau_x = 8$ weeks. Since we observe an increase in the predictive power of unsupervised methods, we can expect increases in supervised classification performance too. In effect, the features more closely reflect actual relationships underlying observable events, so models are more closely related to reality.

This plot also reveals something interesting that again relates to the degree of saturation. The AUC performance of the predictors increases to varying degrees
as $\tau_x$ increases and the network more closely approaches saturation. Preferential attachment is virtually unaffected from 5 weeks of observation to 8 weeks likely owing to the stability of the extremely localized degree statistics in relation to more sophisticated measures. Katz, which performs poorly, receives the greatest boost from saturation, and the curve suggests that further increases of the observational period might allow Katz AUC performance to converge with neighbor-based predictors. The neighbor-based predictors always maintain the same ranking and similar relative performance with increasing saturation, and they improve monotonically.

Interestingly, every predictor in Figure 4.3 increases except for PropFlow. PropFlow performance actually decreases with a larger observational period. This accounts for the striking difference in results between [68] and results herein. In [68], the calls network had $\tau_x = 5$ whereas here the predictors use $\tau_x = 8$. It is not clear why this drop in AUC performance occurs, but one possibility is that network densification increases the strength of highly localized neighbor-based predictors while introducing noise into the weights and paths used by PropFlow. In this case, it is surprising that Katz does not also suffer a drop in AUC performance. Whatever the cause, this should poignantly demonstrate how crucial the saturation of a network is in terms of both the rankings and relative performance of link predictors. Different methods react differently to varying observational periods, so reporting the observational period and providing some indication about saturation is an important part of reporting results.
Figure 4.4. Effect of test set negative sampling on ROC area. The vertical line indicates class balance.

4.3.3 Imbalance Challenges in Evaluation

In link prediction, test set sampling is a ubiquitous practice [4, 59, 64, 68, 84, 98, 100, 111], because sparse networks usually include only a tiny fraction of the \( \binom{|V|}{2} \) potential links leaving predictors to rank essentially the entire remaining set. For even moderately sized networks, this places unreasonable demands on processing resources and storage. As a result, there are many sampling methods for link prediction testing sets, but some methods are fairer than others. One common method selects a subset of edges randomly from the original complete set [4, 100, 111]. Another selects only the edges that span a particular geodesic distance [68, 98]. Yet another selects edges so that the sub-distribution composed by a particular geodesic distance is approximately balanced [111]. Finally there are infinitely many potential methods to select edges that present sufficient information along one or more dimensions [64, 84], for instance selecting only the edges for which each member vertex has some sufficient degree.

Proper test set construction is critical, and randomly removing and subse-
quently predicting edges should be a last resort. Removing and predicting edges removes information from the original network in unpredictable ways, and the removed information has the potential to affect prediction methods differently. Randomly sampling edges for testing when using a supervised approach [4] prepares classifiers based on a random growth process instead of either a real growth mechanisms in longitudinal data or the mechanisms that discriminate observed links from unobserved links in non-longitudinal data. When working with threshold-based measures, any sampling method that removes negative class instances above the decision threshold score unfairly and often unpredictably increases most information retrieval measures. Precision is inflated by the removal of false positives. In top-$k$ measures, recall is inflated by the opportunity for additional positives to appear above the threshold after the negatives are removed. Clearly we cannot report meaningful results with these measures when performing any type of sampling on the test set, but researchers have inadvertently reported such results [4, 111]. The question is whether it is fair to sample the test set when evaluating with ROC curves or other threshold curves based on points in ROC space.

4.3.3.1 ROC Robustness to Class Imbalance

ROC curves and AUC are unaffected by changes in class distribution alone. This allows random sampling of negatives from the test set as long as feature distributions are not significantly changed. Researchers and users do not want to waste the effort necessary to generate lists of output scores only to examine a small percentage. For evaluation purposes we instead randomly select without replacement edges that do not exist in the test network until we achieve the required number. This produces an unbiased result equivalent to random sampling.
from a complete set of output scores.

We demonstrate the results empirically in Figure 4.4. The ROC area remains stable down to 1% negative sampling, below which it is affected. Notably, the dashed vertical line shows the ROC area for sampling that produces a balanced test set, a common practice in evaluation. Further sampling causes greater deviations, increasing the chances of apparent statistical significance at reasonable confidence. This instability does not manifest itself uniformly. In this example, preferential attachment exhibits greater susceptibility to the effect.

This result calls into question the soundness of the practice of undersampling link prediction test sets to balance. If the original goal is to reduce data volume for more manageable testing, then clearly sampling is problematic. A reduction factor of 2 still results in $O(|V|^2)$ test instances and the same order of magnitude. A reduction factor of 100 or 1000, which may still provide huge data sets, potentially reduces the stability of results to a degree that may exceed the difference in performance across methods.

### 4.3.3.2 The Real Testing Distribution

Issues of stability aside, random negative undersampling from the complete test set need not affect AUC; however, we question whether the results it produces are meaningful. Figure 4.5 compares overall ROC performance to ROC performance achievable in sub-problems created by dividing the task by geodesic distance.

The apparent achievable performance by preferential attachment is 13.4% higher in the complete set of potential edges than in data sets restricted by distance. The importance of geodesic distance in determining link formation correlates distance highly with any successful prediction method. High-distance regions
contain very few positives and effectively append a set of trivially recognizable negatives to the end, thus increasing the probability of a randomly selected positive appearing above a randomly selected negative. The effect is exaggerated for PropFlow and other methods inherently tied to distance. The ROC curve for the amalgamated data approximates concatenation of distance-restricted ordered outputs, which places the distances with higher imbalance ratios at the end, where they inflate the overall area. Figure 4.5 shows the effect for the PropFlow prediction method on the right. It exhibits 36.2% higher AUC for the overall score ordering than for the highest of the individual orderings.

This result has important practical implications. PropFlow appears to have a higher ROC than preferential attachment, but the only distance at which it outperforms preferential attachment is the 2-hop distance. Preferential attachment is superior for the other distances. ROC space hides these details from view. The performance indicated by overall ROC is not meaningful with respect to deploy-
ment expectations as it conflates performance across neighborhoods within a bias toward rankings that inherently reflect distance.

Consider the data distribution of the link prediction problem used herein: 148.2 million negatives and 29,898 positives. The ratio of negatives to positives is 4,955 to 1. There are 1196 positives and 214,616 negatives in the 2-hop neighborhood. To achieve balance with random edge sampling, expectation is for 43.3 2-hop negatives to remain. The 2-hop neighborhood contains 30% of all positives, so it presents the highest baseline precision. That border is the most important to capture well in classification, because improvements in 2-hop discrimination are worth much more than improvements at higher distances. The real data distribution on which we report performance when we perform random sampling is the relatively easy boundary described by disparate 2-hop positives and high-distance negatives. Figure 4.6 substantiates this by illustrating the minimal effect of selectively filtering all negatives from low-distance neighborhoods. Performance in the 2-hop neighborhood is most significant because of the favorable class ratio and because improvements in defining this critical boundary offer the greatest rewards. The figure shows we can entirely remove all 2-hop negatives with only a 0.2% effect on AUC. We have to remove all 2-hop negatives, all 3-hop negatives, and even all 4-hop negatives before the alteration becomes conspicuous.

Since data distributions do not affect ROC curves, we can extend this observation even when no sampling is involved: considering the entire set of potential links in ROC space evaluates prediction performance of low-distance positives versus high-distance negatives. We need to describe performance within the distribution of 2-hop positives and 2-hop negatives and select predictors that optimize this boundary.
Figure 4.6. The effect of removing negative instances from increasingly distant potential links. The horizontal line represents the baseline ROC on the unsampled test set.

Of the many sampling methods introduced, completely random removal of edges is the only way to preserve the feature distribution in testing. Though we just showed this distribution means little with respect to optimal deployment, it is at least unbiased. One recently employed alternative takes another approach: aggressively subsampling negatives from over-represented distances while keeping the same number of low-distance instances in the distribution. The IJCNN link prediction competition undersampled the testing set by sampling low-distance edges from the testing network with artificially high probability. The resulting distance distribution appears in Figure 4.7 alongside the natural distribution for the condmat network.

While this selective sampling approach might seem to better highlight some notion of average boundary performance across neighborhoods, it instead creates testing feature distributions that would never exist in deployment. In deployment it is impossible to balance positives and negatives within each distance, because
doing so requires knowledge of the very object of the prediction task. Even if
not maintaining a balanced class distribution within neighborhoods, any selective
manipulation of the feature distribution in the test set affects AUC. It is even
possible to predictably manipulate the test set to achieve arbitrary performance
results. This was not a problem in the IJCNN challenge because all the players
were playing the same game, but it would be a mistake to believe that the results
are in general indicative of real achievable performance or even of a proper ranking
of models.

4.3.3.3 Limitations of ROC Curves

ROC curves and AUC make sense for typical data imbalance scenarios because
they optimize for the performance of interest and because the appearance of the
curve provides a reasonable visual indicator of expected performance. It is typical
to hear statements by researchers examining ROC curves such as: “Wow! 0.99
AUROC. This is a trivially easy classification problem.” That statement is true in
most scenarios because data set sizes are relatively small ($10^3$ to $10^6$) and because
imbalance ratios are relatively modest (2 to 20). Corresponding precisions are near 1. Link prediction is an entirely different domain. For complete link prediction in sparse networks, when every potential new edge is classified, the imbalance ratio is lower bounded by the number of vertices in the network [68]. ROC curves and AUC are deceptive. In a network with millions of vertices, even with an exceptional AUC of 0.99 one could easily suffer small fractions as a maximal precision. The same researchers would usually consider the outcome unacceptable when performance is put in these terms. In most domains, examining several million false positives to find each true positive is the classification problem. Even putting aside more concrete theoretical criticisms of ROC curves and AUC [41], in link prediction tasks they fail to honestly convey the difficulty of the problem and reasonable performance expectations for deployment.

4.3.3.4 Advantages of Precision-Recall Curves

Precision-recall (PR) curves provide a more discriminative view of classification performance in extremely imbalanced contexts [30] such as link prediction. Like ROC curves, PR curves are threshold curves. Each point corresponds to a different score threshold with a different precision and recall value. In PR curves, the x-axis is recall and the y-axis is precision. We will now revisit a problematic scenario that arose with AUC and demonstrate that AUPR displays a less deceptive view of what is actually present in the performance data.

Note in Figure 4.8 that the AUPR is higher for the 2-hop distance than it is for the complete data set. In the underlying curves, this is exhibited as much higher precisions throughout but especially for low to moderate values of recall. Performance by distance exhibits expected monotonic decline due to increasing
Figure 4.8. The AUPR performance of the preferential attachment predictor over each neighborhood. The horizontal line shows overall AUPR performance.

baseline difficulty. Compare this to Figure 4.5 where the AUC for all potential links was much higher than for any neighborhood individually, and the apparent performance was highest in the 7-hop distance data set.

4.3.4 Strategy

We propose a strategy typical in the classification of time series. This is achievable in both event-based and snapshot network link prediction tasks. For longitudinal data that composes event-based networks, construct multiple intervals. For the sake of demonstration, assume 10 equal-width periods. Within these periods, we construct the intervals that support the evaluation. For unsupervised link prediction, the first interval includes periods 1 through 9. After constructing a network from the events in this interval, each method generates the set of scores that it will actually use for testing from that network. The testing interval is constructed from period 10, and evaluation is performed by comparing the output
scores to the links that actually form in the network composed from that interval. In supervised link prediction, the first interval includes only periods 1 through 8. The resulting network is used only for the training set features, not for actual testing. Another interval formed from period 9 produces the labels. At this point, the supervised classification model is trained. Subsequently, we construct a new interval from periods 1 through 9, precisely the same as for unsupervised classification, and generate features for a testing set. The labels for the testing set come from period 10 just as with unsupervised prediction.

This strategy accomplishes several things. First, it allows for a direct comparison between the performance of unsupervised and supervised link prediction methods. The population on which the two methods are tested is identical. Second, it does not differentially confound predictors by randomly removing information that may handicap them in different ways. Instead, links in the testing data are necessarily distinct from the information available in the training intervals. Third, supervised classifiers can benefit honestly from any additional information that appears in period 9 prior to generating their actual predictions for period 10.

Most importantly, when evaluating, all of the original potential links from the testing interval should be provided to the classification model. The potential links should not be undersampled or subsampled in any way. Further, if we desire to perform cross-validation, 10 folds for instance, we must separately undersample each 90% training fold and leave the testing fold unmodified.

When neither longitudinal data nor multiple vertex-correlated snapshots is available, then the same procedure degenerates gracefully to the standard random sampling approach. To run the procedure, we can sample without replacement 10% of the edges from the entire set and construct 10 equal-sized groups. We follow
the same process with these groups as with the actual events, so this serves as a sort of benchmark for the visualizations. It obviously lacks the natural independence that true successive snapshots or event periods provide, but it still benefits from the first and third advantages above.

4.4 Conclusions

To select good link predictors, we must know how to evaluate our techniques. Beyond this, we must be sure that readers do not come away from papers with the question of how the method actually performs. It is more difficult to specify and explain link prediction evaluation strategies than with standard classification wherein it is sufficient to fully specify a data set, one of a few evaluation methods, and a given performance measure. In link prediction, there are many potential parameters often with many undesirable options. There is no question that the issues raised herein theoretically can lead to questionable or misleading results. Hopefully the empirical demonstrations convince the reader that they do lead to questionable or misleading results. We propose the following guidelines:

1. Use PR curves and AUPR as an evaluation measure. Use ROC curves and AUC as optional accompaniment. Avoid fixed thresholds.

2. Decompose the link prediction task into sub-problems by geodesic distance. Report results accordingly.

3. Do not undersample negatives from test sets. Test sets corresponding to low geodesic distances are manageable without sampling.

4. If negative subsampling is undertaken for some reason, it must be based on a purely random sample of edges missing from the test network. It must not modify original feature distributions. Any sampling must be clearly reported.
5. In undirected networks, state if the method is invariant to designations of source and target. If not, state how the final output is produced.

6. Use the same testing set instances regardless of the method used for training.

7. In longitudinal data, demonstrate saturation or lack of saturation in some way, such as with a growth plot of the number of vertices and edges. Include statements about how snapshots were formed from raw data.

8. In longitudinal data, the final test set on which evaluation is performed should receive labels from a subsequent, unobserved snapshot of the data stream generating the network. Avoid testing on randomly removed edges.

9. Consider whether the task is solving the recommendation problem or the query problem and construct test sets accordingly.
The final major problem in network analysis and link prediction that we tackle is that of computational speed and scaling. We address this issue in two senses. The first is the speed at which outputs necessary for link prediction can be computed as dictated by the properties of the problem and the choices of the prediction framework. The second is the speed at which the link prediction framework may reasonably be deployed, from start to finish, on raw source data. Predictions are useless no matter how good they are if they take months or years to obtain, and an important component of this work is ensuring that they compute in a reasonable time for those interested in deploying link prediction for either research or practical goals.

5.1 Fast Computations in Networks

The first speed problem we seek to solve relates to computational speed. This does not only involve computational speed as it relates directly to generating prediction scores for a given link prediction algorithm. It also relates to quickly determining reliable statistics about a network that may hint at what link prediction approaches and parameters may be most efficacious. Distributions of vertex properties such as betweenness centrality distributions may be useful as broad
global descriptors of the network itself or as features of vertices that may them-
selves be useful in the prediction task. Given the size of the networks often under
consideration, many such properties are difficult to compute in a reasonable time,
especially if they relate to paths or walks within the network. There are many fast,
well-written graph libraries, and using such a library is an important component
of fast computation overall, but many computations can exceed limits of feasi-
ble elapsed time even with very fast code. We address this with a system called
DisNet, which is designed to handle computation in large networks by exploiting
natural, coarse-grained parallelism [66].

5.1.1 Network Growth

Reflecting on literature from the last decade, one unsurprising trend in network
science is clear. The size of networks under study in research and application con-
texts has exploded. Whereas sociological networks of interest in the past mostly
consist of those produced by extensive direct observation or surveys, naturally lim-
itng the scope to hundreds or thousands, the availability of large electronically
produced data sets has now expanded this scope to millions or even billions. The
Internet and World Wide Web, in a relatively fledgling state in the early 1990s,
have blossomed into networks of billions or tens of billions of nodes and edges.
Another related observation is that the increased interest in network science has
led to its applications in more standard areas of machine learning such as collab-
orative filtering [47], which may be applied to large medical data sets or vast data
repositories such as the Netflix movie ratings data base. Methods implemented in
terms of spectral graph theory, despite their power and elegance, require $O(|V|^2)$
space and thus cannot compete with the compact $O(|V| + |E|)$ adjacency list
representation necessary to maintain these networks in main memory.

5.1.2 Existing Methods

Work in fast graph computation resides at the intersection of work in distributed computing and in graph algorithms, so a wide variety of research has at least some bearing. We focus here on reasonable existing approaches for accomplishing rapid computation in large networks. In [74], the authors offer a much more complete survey of the challenges and options available. One solution is the application of approximation algorithms or specifically targeted parallel algorithms. For instance betweenness centrality is of such interest that researchers have developed a number of algorithms since Brandes’ exact algorithm [13], including an approximation algorithm [7], a range-limited algorithm [35], and a parallel algorithm [75]. While such solutions are impressive and clever, each such approximation or parallel algorithm requires its own impressive cleverness. That is to say that the approach of devising these algorithms lacks generality in providing a solution to the explosion in the size of network analysis tasks. Further, these algorithms are often conceptually complex, may be difficult to implement, may still require substantial serial post-processing time, may provide unacceptable bounds on approximation accuracy, or may admit no error bounds at all.

Another approach is to grant the unmanageable computational complexity and devise better ways to attack the problem with greater computational resources. Barring significant breakthroughs in serial computational speed, we cannot hope for a single processor core to keep up with the size of the network data we wish to analyze. Instead, we must apply resources from computational grids and clouds to attack the problem. One solution involves distributed memory and message-
passing, for example using MPI. The Parallel Boost Graph Library (BGL) [40] is designed primarily for user extension in achieving MPI computation on distributed graphs, but it supplies and supports parallel algorithms on traditional graphs. Writing parallel implementations in this manner is tricky, however, and to such a degree that the implementation and study of single-source shortest paths in the Parallel BGL was sufficiently novel to warrant publication [34]. In short, while researchers can use existing parallel implementations in these tools, and even this requires knowledge of MPI, new implementations require additional knowledge beyond vanilla programming that many lack. Furthermore, message passing with distributed graph representations to exploit fine-grained parallelism is inefficient when problems present sufficient coarse-grained parallelism.

The authors of XWS [26] provide a nice framework with which users can more easily manage concurrency in fine-grained parallelism. Because it allows for fine-grained parallelism, the best it can do in terms of simplicity is provide a set of abstractions and primitives. It does not remove the burden of understanding the parallelism from its users. The simple alternative to such fine-grained parallel computation is coarse-grained distributed computation. MapReduce [31] provides an excellent framework for distributed computation when coarse-grained components of the computation are independent and divisible. For graph computation specifically, [25] describes a number of graph analysis techniques in terms of their MapReduce transformations. In recognition of some of the limitations and efficiency problems of MapReduce for many graph problems [76], the authors of [19] extend the framework with a data propagation primitive. In general, while we do not disagree that MapReduce is an excellent paradigm for computation in graphs, [25] acknowledges that MapReduce transformations often require a significant re-
thinking about the computation in question.

Pregel \[76\] has a vertex-centric approach and its own processing and combination primitives. Pregel provides for the modification of the input network and employs message passing between computational workers. This grants it a great flexibility and a broad range of applicability, but this again comes with an additional burden of code development and understanding on the user. Writing algorithms for Pregel requires writing code in terms of Pregel primitives and messaging operations. Pregel uses message passing with aggregation as part of the means of achieving its flexibility and must therefore occasionally wait for messages to continue processing. Ideally, coarse-grained processing would be bound only by the speed of local processors and memories. If any transmission of results is necessary, such communications should all be performed while other independent computations are running.

One way to achieve this sort of distributed network analysis is by using the Makeflow \[115\] and Work Queue \[109\] computational abstractions. In the context of computation in networks, Makeflow allows for the specification of jobs through explicit vertex groupings. Work Queue provides a master-worker framework that allows for robust distributed computation through worker job assignment with automatic failover handling. The problem with using these systems for graph computation is that they incorporate no knowledge of the specifics of the graph problem. The real unit of independent computation is the vertex, but 5 million jobs each processing one vertex makes for inefficient organization. The natural solution is to specify ranges of vertices for each job, but this requires brutish user intervention. Moreover, each job requires workers to read the network before accomplishing anything meaningful, and workers are single-threaded unless coded
otherwise by the user, so they cannot take advantage of a shared immutable representation of the network. Combining the results also presents difficulties. Makeflow might be used to specify a dependency on all worker vertex results or dependencies on subsets of worker vertex results. A dependency on all workers requires a long wait as local resources are mustered to merge potentially hundreds of thousands of results stored in temporary files, which may require terabytes of disk space. Specifying groups of dependencies requires yet more time-consuming finagling and may work poorly when dependent computations fail due to worker machine failures.

5.1.3 The Vertex as a Fundamental Unit

Many network algorithms are based on running a similar procedure for each vertex in the network. This includes many implementations of all-pairs shortest paths, eccentricity computations, radius and diameter computations, radiality, clustering coefficient, graph census and motif membership, centrality, random walk measures, and link prediction algorithms. The algorithms behind these underlying computations visit a vertex and compute properties relative to that vertex that either are or may be independent of computations that previously or subsequently pertain in other vertex computations. This naturally suggests a Map-Reduce architecture, but such an architecture that incorporates specific knowledge of the network is particularly useful. This list of vertex-independent problems is by no means exhaustive, but the list of applications to which the problems apply is much larger still.
5.1.4 DisNet

There are several fundamental differences between DisNet and the aforementioned existing systems. DisNet represents the recognition that many interesting problems in network analysis involve computations that are inherently vertex-centrically independent. DisNet comes with two maxims. First, naturally parallel graph computation is most quickly run when it is unencumbered by synchronization or knowledge of unrelated computations. Second, graph algorithms are most quickly developed when the developer can focus on the complexities of the serial algorithm instead of the complexities of parallelism. Many researchers in biology, sociology, and other fields have minimal programming expertise. They can write serial graph processing code but cannot be expected to have the knowledge to write MPI or multi-threaded code or otherwise manage aspects of parallelism. DisNet presents many advantages: it is extremely easy to use and deploy, robust, fast, and can exactly solve a large set of problems of interest in the analysis of large graphs. If you can write serial code describing a graph algorithm, you can use DisNet. It is even possible to debug DisNet code easily using standard debugging tools and techniques.

Some may be concerned about using localized in-memory representations exceeding the size of main memory. In reality, well-designed in-memory representations with adjacency lists are only problematic for extremely large networks. In such networks even distributed computational resources fail to return results in a timely manner, so space concerns are moot. An uncompressed space-efficient immutable adjacency list representation of a graph of $|V|$ vertices and $|E|$ edges theoretically requires only $|V| \cdot \lceil \log_2 |V| \rceil + |E| \cdot \lceil \log_2 |V| \rceil$ bits of space, because only $\log_2 |V|$ bits are necessary to address $|V|$ elements. To put this in a more
concrete perspective, a large social network of 5 million nodes and 20 million edges requires less than 72 MB of space. Networks with 6 billion nodes and 36 billion edges, a possible network representation of all living humans, would require only 27.9 GB. Practical demands are somewhat higher, but not much. Meanwhile, many research universities have computing clusters with hundreds or thousands of available computational nodes, and services such as Amazon EC2 provide inexpensive computing time to those without direct access to such clusters.

DisNet represents a new avenue for approaching computation in large networks. With recognition of the natural parallelism in most network computation through a vertex-centric approach, a framework can achieve high efficiency and allow for easy development. We illustrate how computationally complex measures like graph diameter, closeness centrality, and betweenness centrality can be calculated in a matter of hours on a representative of one of the largest social networks under active study, results that would have required years of computation with serial algorithms. Researchers can achieve the same complex computations in a reasonable time given widely available grid resources. Most importantly, the system is easy to use and understand. DisNet abstracts away all details of parallelism so that users only need to consider the fundamentals of the problem they are trying to solve in natural, untransformed terms and can ignore virtually all aspects of how the computation is achieved.

We recognize that some networks under study contain $10^9$ vertices or more. So long as there is a reasonable pool of workers with sufficient memory, DisNet is not limited to networks of any particular size. Nonetheless, for these extremely large networks, duplicate local representations of the network in general, and DisNet in particular, may not be suitable. We contend that the selection of interesting
algorithms, so many of which are \( O(|V|^2) \) or \( O(|V|^3) \), that can reasonably be applied to such networks is currently severely constrained under any system even with world-class computational resources. In short, DisNet is not an attempt to allow the computation of anything conceivable on any conceivable network. It is an attempt to make harnessing many heterogeneous computational resources easy and efficient in the pursuit of computing many interesting computationally complex algorithms on most networks under study.

5.1.4.1 DisNet User Interface

Design and implementation decisions are geared toward allowing users to complete the development phase as quickly and easily as possible so that they can finish the real work of computing the results they desire. From start to finish, typical users of DisNet must simply specify three things: the data type into which results are computed, how to compute results for a single vertex, and how to combine results from two vertices in the selected data type. Specifying the data type requires only selecting from a list of preprocessor macro options. Single-vertex result computation and data combination require writing code or calling graph library functions.

In the current implementation, typical users should modify one file, `user.cpp`, to make these three modifications. This file does not contain any DisNet architectural code or graph library code. Instead, its sole purpose is to serve as the one location where users write the small code segments necessary to specify the computations they want DisNet to perform.

This implementation of the DisNet framework employs C/C++ for reasons of memory efficiency and execution speed. The framework will scale to million-
node networks even with commodity resources and billion-node networks with available cluster resources. Users unfamiliar with C/C++ need not fret as actual coding requirements are minimal. Only a basic knowledge of C/C++ is required to achieve computation using the provided graph library, and the user need not understand any details of the DisNet implementation itself. Despite its current roots in C/C++, the ideas that form the DisNet architecture are entirely separate from this particular implementation. DisNet may be implemented in or employ modules from many programming languages including Perl, Python, Java, and R.

We will briefly describe a few artifacts in the code segments below for reasons of clarity. First, the object variable network is available globally in user.cpp. It contains the single, immutable local representation of the network, which is shared among all worker computational threads. The network object supports several self-analytical functions and simple functions to support arbitrary computation over the topology, such as vertexCount(), which indicates the number of vertices in the network. vertex_t is a type definition for vertex identifiers. Likewise, neighbor_set_t is a type definition for the container class that holds adjacent vertex_t neighbors for each vertex.

Users may select whatever data type they like to contain their data. The specification involves modifying one line in user.cpp to select from the provided data types. Existing selectable data types include primitives, strings, and several STL containers including vector, map, and set. Adding additional data types is simple, requiring the specification of only a few lines of code in the file macros.h and potentially a basic ASCII serialization routine.

The initialization of these data types is important because the result of data type initialization serves as the initial value passed into the combination function
with the first results. In the current implementation, all data types including primitives will be initialized by their default constructor except for random-access data structures, which contain $|V|$ objects each initialized by their default constructor. For example, a `double` is initialized to 0.0, a `map<unsigned int,bool>` is initialized to a map with no existing key-value pairs, and a `vector<string>` is initialized to $|V|$ empty strings. These defaults are reasonable because they serve as identities, either additive identities or set union identities or otherwise. Consider, for instance, betweenness, which involves attributing partial sums to each vertex in the network. In this case, the initialization of a vector of values to the additive identity makes perfect sense. In the rare case when the default initialization behavior for an existing type is undesirable, users may easily modify it by changing one line of code in `macros.h`.

Often, many different data types will get the job done. For some problems particular data structures are required. For others selecting a particular data structure may reduce performance or increase network or disk demands. Consider the case of centrality computations. Computing closeness centrality for a single vertex results in a single number representing the average distance of that vertex from all others. This number could be placed in a `vector<vertex_t,double>`, but then each combination requires $|V|$ summations and every transmission to the master must send the entire vector. If a map is selected, then the combination step is merely adding an entry to the map, and transmission only involves as many vertices as have been computed. On the other hand, the betweenness centrality computation for each vertex produces the contribution of that vertex in shortest paths to other vertices. The computation requires storage and fast access for all vertices, and all vertices must be transmitted for accumulation of partial sums. In this case,
TABLE 5.1

OPTIMAL DATA STRUCTURE AND REQUIRED SLOC FOR (A)

\[ \text{process_vertex} \] AND (B) \[ \text{combine_data} \] ROUTINES

<table>
<thead>
<tr>
<th>Problem</th>
<th>Data Structure</th>
<th>SLOC(A)</th>
<th>SLOC(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Betweenness</td>
<td>vector&lt;double&gt;</td>
<td>28</td>
<td>3</td>
</tr>
<tr>
<td>Closeness</td>
<td>map&lt;vertex_t,double&gt;</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>Eccentricities</td>
<td>map&lt;vertex_t,unsigned int&gt;</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>Graph Diameter</td>
<td>unsigned int</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>Link Prediction - Katz</td>
<td>string</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

a vector<double> is the most efficient choice. Finally, data types may contain multiple information elements such as a map<vertex_t,pair<int,double>> so that problems with overlapping solutions like eccentricity and closeness can benefit from the same computation.

Vertex processing is the first of two entirely isolated locations where users must supply code. The function \text{process_vertex} in \text{user.h} is constructed so that users can forget about every aspect of the framework except how to compute results for a single vertex. User code can either employ the API in the network library to achieve arbitrary unanticipated analysis, or it can call library procedures. For example, the included library already implements closeness centrality so computing closeness for all vertices requires writing only a one-line call. In the case when users want to experiment with new analytical techniques or implement new functionality, most of the coding requirements for the framework will rest in
the vertex processing task.

Result combination is the second of two entirely isolated locations where users must supply code. The function `combine_data` in `user.h` is constructed so that users can forget about every aspect of the framework except how sets of results from two vertices should be combined. This snippet of code will usually be extremely minimal. It might involve merging maps, summing two vectors, concatenating two strings, or computing the maximum of two numbers. These tasks are usually straightforward, involving at most a loop expressed in three lines of code. The combination function does not have any theoretical constraints. It may be idempotent, associative, or commutative depending on what the user wants to accomplish. If the combination function involves generating the union of one set with another, all three properties pertain. If the combination function involves the concatenation of two strings, none of the three pertain.

To illustrate just how little effort is necessary, and to provide an entirely concrete example, we include a complete user specification for a working DisNet deployment in Listing 5.1. To compute the diameter of a graph, we need only to select the largest among all vertex eccentricities. Maintaining the diameter we have discovered so far requires a single integer, so we specify the data type as a single integer. For each vertex, we must perform a breadth-first search, to determine the maximum distance of any vertex from a source vertex. To combine results, we choose the largest eccentricity.

Table 5.1 provides summary details of the development requirements for a few other research problems. We use physical source lines of code (SLOC) as a measurement of the difficulty of the development task. The SLOC determination is based on a standard count of semicolons.
```c
#define DATA_TYPE DATA_TYPE_UNSIGNED_INT

void process_vertex( void * dataPtr , Network const & network , vertex_t vertex ) {
    // automatic cast and declaration of 'data' variable from dataPtr
    vector<bool> found( network.vertexCount() );
    vector<vertex_t> v1;
    found.at( vertex ) = true;
    v1.push_back( vertex );
    for( data = 0; !v1.empty(); ++data ) {
        vector<vertex> v2;
        for( vector<vertex_t>::const_iterator vIt = v1.begin(); vIt != v1.end(); ++vIt ) {
            neighbor_set_t const & n = network.outList.at( *vIt );
            for( neighbor_set_t::const_iterator nIt = n.begin(); nIt != n.end(); ++nIt ){
                if( !found.at( *nIt ) ) {
                    found.at( *nIt ) = true;
                    v2.push_back( *nIt );
                }
            }
        }
        v1.swap(v2);
    }
}

void combine_data( Network const & network , void * dataPtr , void const * vertexDataPtr ) {
    // automatic case and declaration of 'data' and 'vertexData' variables
    data = data >= vertexData ? data : vertexData;
}

Figure 5.1. All required DisNet user code for graph diameter.
```
5.1.4.2 Architectural Design

The framework employs the master-worker paradigm to accomplish computation. The master coordinates work among the workers and maintains the consistency of the results. Workers accept vertices as basic independent units of the computational task and run the same user-specified routine for each. Workers communicate only with the master, never with each other. Figure 5.2 is an illustration of these basic components, their construction, and their interactions. Each master communication thread and each worker share the same data as their exemplars on the left.

The only requirement for a machine to serve as a DisNet master is that it have an accessible port on which to listen for contact from new worker machines. Workers may be instantiated on local or remote machines directly operated by the user, within computing grids such as Condor [70] or the Sun Grid Engine (SGE), or even on cloud resources such as Amazon EC2. DisNet is not inherently married to any particular underlying grid or cloud management system. Any machine or virtual machine that can send and receive data using TCP/IP may serve as a worker. Any grid or cloud management system that allows user-specified programs and network communication can host pools of workers. In practice, we have simultaneously employed locally-operated servers, Condor, SGE, and Globus. DisNet includes batch submission scripts for all of these.

To be useful and effective, the framework must scale to provide service for at least thousands of workers, reliably handle worker failures, and efficiently make use of disk and network resources. Because systems may provide multiple cores to a worker, the worker abstraction should handle these resources optimally. Some high-throughput systems like Condor employ a wide variety of architectures and
Figure 5.2. An architectural overview of DisNet. Data structures with synchronized access are marked with *.
operating systems with heterogeneous resources, so the framework must be con-
structed in a way that is portable across operating systems and architectures both
in terms of how it is constructed and how it transmits information. Given that
computation may require years of CPU time, the framework must consider com-
putations to be precious and must be able to recover from almost any type of error
or failure in a consistent state and with minimal loss of data. The data involved
may place arbitrarily severe demands on the disk and network, but the machines
involved may be arbitrarily prone to failure. To allow for appropriate trade-offs
in this environment, the framework must allow for simple tuning. Perhaps most
importantly, the framework must allow researchers to compute results on large
networks quickly and with minimal effort. We can encapsulate these goals tersely:
portability, efficiency, scalability, reliability, customizable tuning, and ease of use.
We have discussed the final point, ease of use, in Section 5.1.4.1.

DisNet achieves portability through the use of Bourne shell scripts and re-
mote compilation. When `master.sh` is invoked, it combines and compresses all
worker source code and creates a compressed representation of the network. When
`worker.sh` is invoked, it first requests worker source code from the master, which
it then compiles. Remote compilation is performed once at each physical worker
machine and requires only a couple seconds. Next, the worker requests the network
from the master unless the network file is specified to be available locally. Trans-
mission of compressed million-node social networks requires less than 3 seconds
on a 100 Mbps network. After these preliminary operations have been successfully
completed, the worker Bourne script invokes the C++ worker binary. The setup
time is negligible compared to the time required to compute the difficult results
for which DisNet is designed.
DisNet workers load the network into memory once when they are started and use the same immutable representation for the duration of their lifetime. The immutable nature of the network also allows for a single read-only representation of the network that all worker threads can share. Local workers on multi-core machines can employ any number of those cores for computation. As processors offer increasingly many cores, workers will enjoy increasing speed and lower overall ratios of memory requirements to core count. Transient threads arise to handle each vertex and require only the additional memory necessary for running the computing algorithm. The main worker thread also contains a copy of all data combined so far. When each transient thread completes vertex processing, the result is combined with the existing data under a mutex lock.

Time with the master is a precious resource, so workers accept as a parameter a number of vertex results to combine locally before sending data to the master. As we will demonstrate theoretically and practically below, this combination parameter effectively eliminates the master as a bottleneck without the complexity of having multiple masters in a distributed or hierarchical arrangement. It simultaneously reduces processing demands on the master and reduces the bandwidth required by both the master and the worker. Transient threads spawned by the worker for additional vertex assignments continue running while the worker communicates with the master. The combined vertex data is locked until communication finishes, at which time it is cleared and again becomes available to the transient threads for update.

One of the principal concerns of DisNet is scalability. By isolating workers from each other entirely, the architecture guarantees that the only potential bottleneck is the master. The master must be able to handle many simultaneous
connections, deserialize results, combine results with the combination function, and checkpoint all as quickly as the workers can supply results. Whereas workers are likely only to encounter CPU limitations, the master may encounter limitations due to the CPU, the disk, or the network. CPU and network limitations are both due to frequent and voluminous client communication, which necessitates equally frequent deserialization and combination. By designing workers that combine results as they produce them, most of the burden on these resources can be reduced to allow for virtually arbitrary scalability. Scalability problems are reduced to tradeoffs between speed and waste, which we can leave to users based on their level of confidence in the stability of their systems.

We implemented the master using a thread-per-worker communication paradigm. Although we are aware of theoretical arguments against such a paradigm, in practice we did not observe thread scheduling problems or network saturation. The framework does not currently provide for hierarchical master-worker relationships where workers are themselves masters for a broader array of workers. Such a provision would indeed decrease load on the master, but at the cost of a decrease in CPU and network efficiency. Load on the master is not problematic in practice. Even if it were, the master can itself utilize multiple cores, and result deserialization is often substantially more expensive than data combination in the mutex, reducing the probability of encountering a real bottleneck.

The most efficacious approach to achieving reliability is to acknowledge that workers are inherently unreliable and to treat them accordingly. Workers may experience software or hardware failures for innumerable reasons, most of which are entirely beyond our control. The master ensures the appearance of reliability to the user through failover. The master must be able to detect when workers fail
so that it can reassign the relevant vertices. It accomplishes this using keepalive probes. When a worker fails to respond to a number of successive keepalive probes, the master considers the worker dead, closes the connection, and moves all outstanding assigned vertices from the \texttt{RUNNING} state to the \texttt{FAILED} state.

We take the approach that the system should successfully complete as much computation as possible as quickly as possible in the event that errors occur. Because some worker failures may actually result from a problem processing a particular vertex, vertices in the \texttt{FAILED} state are reassigned only when there are no longer \texttt{REMAINING} vertices to assign to waiting workers. This implicitly assumes that the remainder of results are still valuable and avoids spinning on the assignment of problematic vertices that consume worker time only to probably or certainly fail. After all vertices have been moved from the \texttt{REMAINING} state through assignment to workers, \texttt{FAILED} vertices are reassigned repeatedly until all vertices are finished or the user terminates the master. At no time are resources idle, because \texttt{FAILED} vertices occupy free resources as soon as there are no longer \texttt{REMAINING} vertices to assign. Alternatively, \texttt{FAILED} vertices might be retried immediately with a threshold number of attempts before reporting the failure to the user.

In the case of master machine failure, recovery is accomplished via checkpointing. Checkpoints are created by writing a single temporary file that contains the vertex status information and the computed results. After writing a temporary file, the master atomically renames the file to create the latest checkpoint and then updates the in-memory vertex status information, moving vertices from the \texttt{DONE} state to the \texttt{CHECKPOINTED} state. When all vertices reach the \texttt{CHECKPOINTED} state, the master indicates successful completion and terminates. Until then, the
master sends deferral messages to unoccupied workers in case straggling computations fail. The master creates a checkpoint in one of three circumstances: (1) every time a specified number of vertices has been successfully processed, (2) when all vertices have been successfully processed, or (3) upon detecting Ctrl+C. When the master is restarted after failure, it loads the vertex status list and result data structure from the latest existing checkpoint and listens for connections from new workers.

Tuning is achievable with a data combination parameter, $c$, which is accepted by master and worker alike. On the master, this parameter designates the number of vertex results, not worker transmissions, to accept and combine before initiating a checkpoint. Decreasing the value of this parameter results in more frequent checkpoints and less wasted worker time if the master fails; increasing it lowers the disk throughput requirements on the master. On workers, the combination parameter designates the number of vertices to finish and combine before sending data to the master. In this case, a lower parameter value decreases wasted worker time when the worker fails; higher values reduce the computational and network bandwidth demands on the master. The parameter $c$ can be set differently on the master, each worker, or on each batch of workers submitted to a computational grid.

5.1.4.3 Efficiency and Scalability

Analyzing the efficiency of non-parallelized algorithms on real social networks of the size for which DisNet is designed is intractable because they simply take too long to run. DisNet does not reduce the total computational time to complete long-running algorithms on large graphs but simply makes it extremely easy
to distribute this time across many heterogeneous resources. For comparative experiments, we extract the largest strongly-connected component from Erdős-Rényi \((n, M)\) : \(M = 4n\) model random graphs of different sizes to gain an idea of the efficiency and scalability of the distributed system. All experiments are performed with betweenness centrality, closeness centrality, and diameter using code compiled with the \(-O3\) flag to the GNU \texttt{g++} compiler. The master is a dual quad-core 3.0 GHz Xeon machine with 1 Gbps network bandwidth. We report CPU utilization such that 100% utilization means complete occupation of a single core.

Figure 5.3 shows the strong scaling of the system, how it behaves as additional workers are added. We select a random graph with 125,871 vertices and 503,411 edges. To obtain these results, we use only 2.53 GHz core-clock Nehalem workers on the SGE. The master was set to checkpoint only after completion and the
Figure 5.4. Performance and master burden with respect to the worker combination parameter.
worker combination parameter was fixed at 200.

The figure indicates excellent strong scaling, almost perfect scale-free behavior. Doubling the number of workers results in 50-55% of the wall time at any number of workers. From 50 to 100 workers, the time to complete the computation becomes so small that the time to set up worker connections, send source code, and distribute vertices becomes a significant factor. To illustrate that strong scalability continues to pertain with an ever-increasing number of workers, we computed the two more challenging problems on a SCC with 503,758 nodes and 2,014,746 edges. Closeness required 1186 seconds of wall time with 50 workers and 602 seconds with 100 workers, 51% as much time. Betweenness required 6319 seconds of wall time with 50 workers and 3416 seconds with 100 workers, 54% as much time.

We also report burden on the master in terms of CPU time with different values of the combination parameter, \( c \). We use the same random graph with the same parameters as for the strong scaling experiments, but the number of machines is fixed at 100. Figure 5.4 illustrates results. Theoretically, the three problems are representatives of different classes of difficulty in terms of their network bandwidth requirements and deserialization demands on the master. Betweenness centrality scales according to \( O\left(\frac{|V|^2}{c}\right) \), closeness centrality according to \( O(|V|) \), and diameter according to \( O\left(\frac{|V|}{c}\right) \).

To understand the reason for these different classes of scaling, one must consider what information must be sent for each vertex and how that information interacts with or is independent of \( c \). Betweenness centrality requires calculating and summing partial sums that result from one computation initiated at each vertex. Each transmission requires sending \(|V|\) values, and \(|V|\) transmissions are required, but workers can effectively reduce the number of transmissions by a
factor of $c$ by summing together individual vertex results. Closeness centrality requires sending $O(1)$ information for each of the $|V|$ vertices. Worker combination can aggregate this data for fewer transmissions, but ultimately all $|V|$ values must be transmitted. Finally, diameter requires only a single integer. This integer must be selected from the maximum eccentricity of each of the $|V|$ vertices, but worker combination can reduce the amount of data transmitted by selecting the maximum integer from all its results and discarding the rest.

Practical results align well with theory. Betweenness places much greater demands on the master. Nevertheless, we observe that the time required to finish the computation stops decreasing after the combination parameter reaches 25, which indicates that for the selected graph size and 100 workers, the master is no longer a bottleneck. We also note that increasing graph size for betweenness centrality computations increases the size of the data to transmit to the master, but computation for individual vertices also takes longer. The $O \left( \frac{|V|^2}{c} \right)$ requirements above are overall requirements. Per unit time, the number of workers involved is more important in selecting a combination parameter balanced against the reliability of the worker machines in question.

The substantially lighter demands of the closeness and diameter computations are apparent in their lower absolute time requirements, but also in terms of their flatness. Even with workers sending results upon finishing each vertex, the master is no bottleneck, but CPU utilization is slightly higher for low values of $c$. Since closeness centrality data volume is independent of $c$, the smaller, more frequent transactions are causing the higher utilization by virtue of their frequency alone. In employing DisNet to achieve our own results for a large social network, the master was not a bottleneck. Speeds are instead bound to the number of accessible
workers.

Betweenness centrality is among the most stressing problems, because the data structure must always contain information for every vertex. For this reason, it serves as a conservative benchmark for DisNet. Figure 5.5 illustrates results for several problem sizes. In line with our scalability claims, we observe a monotonic decrease in average master CPU utilization with increasing network size. DisNet does not reduce the total computational time required to complete a particular task, so the functional form of the growth in Figure 5.5 is the same regardless of how the computation is achieved. Nonetheless, when the graph size surpasses 500,000 vertices, a week of serial computation is no longer sufficient to reach a solution. Even with only 100 cores, DisNet requires under two hours, and adding more computational power is simple.

Finally, we are interested in the most grandiose performance we can muster.
with DisNet and the computational resources available to us. We compute the betweenness of a real-world social network with 4,773,246 vertices and 29,393,714 edges. The run required 25 hours. The master interacted with 2368 distinct workers and successfully handled 442 worker failures, most due to Condor evictions. The most highly multi-threaded worker ran 28 threads on a 32-core machine, achieved nearly constant 2800% utilization, and required only 10 GB of memory. Local workers were run with a combination parameter of 1000, SGE workers with 500, and Condor workers with 250. The master was set to checkpoint every 100,000 vertices. With logging overhead, peak master CPU utilization was 609%, average master CPU utilization was 307%, and total master CPU time was 76.1 hours.

5.2 Work Flow Management and Job-Level Parallelism

The ability to execute a single in-network task quickly is important, but it by no means ameliorates the only obstacle in the way of easy practical applications of link prediction and other network analysis techniques. Link prediction especially involves a complex procedure consisting of dozens, hundreds, or thousands of steps. Each step itself offers many parameters and may take a long time to run. Many of the tasks are entirely independent, but there is a complex dependency hierarchy between them. An ideal solution would automatically distribute the tasks to available local computing resources, incorporate knowledge of any work that has already been completed in the case of failures, and perform the minimum amount of work necessary for the desired tasks. We introduce the software package, LPmade, which incorporates all of these features in a user-friendly framework that incorporates a highly efficient network library, optimized evalua-
tion tools, and automatic plotting capabilities. The ultimate goal is to minimize the possibility of human error in the computation of link prediction and network analysis tasks while simultaneously minimizing human effort and optimizing the utilization of local machine resources.

Link prediction is of great use in domains ranging from biology to corporate recruiting, but it is a difficult problem for which to develop models because of extreme class imbalance, the longitudinal nature of the data, the difficulties inherent in effective evaluation, and other issues raised by [68]. Further, even for standard prediction algorithms, researchers must often write new code or cobble together existing code fragments. The work flow to achieve predictions and fair evaluation is time-consuming, challenging, and error-prone. LPmade represents not only the first library to focus on link prediction specifically, incorporating general and extensible forms of the predictors introduced by [64]. It also streamlines and parameterizes the complex link prediction work flow so that researchers can start with source data and achieve predictions in minimal time.

There is no shortage of graph libraries: the Boost Graph Library, SNAP, igraph, JGraphT, GraphCrunch, GOBLIN, and many others. Some offer extreme generality, some offer extreme efficiency, some offer modeling utilities, and some have a dizzying array of algorithms. LPmade is not just yet another graph library. Its software components are, by necessity, designed for high performance, and it offers a wide array of graph analysis algorithms, but it is first and foremost an extensive toolkit for performing link prediction to achieve both research and application goals. LPmade provides an organized collection of link prediction algorithms in a build framework that is accessible to researchers across many disciplines.
5.2.1 The Software Package

The purpose of the software is to provide a workbench on which others may conduct link prediction research and application. For link prediction tasks in many large networks even a restricted set of predictions may involve millions, billions, or even trillions of lines of output. Each unsupervised link prediction method, the supervised classification framework from [68], and all the evaluation tools are optimized for just such quantities of data. Nonetheless, the entire process of starting from raw source data and ending with predictions, evaluations, and plots involves an extensive series of steps that may each take a long time. The software includes a carefully constructed dependency tracking system that minimizes overhead and simplifies the management of correct procedures. The software package is designed in line with the UNIX philosophy of combining a series of specialized tools to accomplish broader goals. Both the build system and the link prediction library are modular and extensible. Researchers can incorporate their own link prediction methods into the library and the automation framework just by writing a C++ class and changing a make variable.

5.2.2 Network Library

The LPmade network library is written entirely in scalable, high-performance C/C++ that minimizes memory consumption with a compact adjacency list format based on a vector-of-vectors to represent edges and a translation vector that supports binary search to translate external vertex names to internal identifiers. The library includes clearly written yet optimized versions of the most common asymptotically optimal network analysis algorithms for sampling, modeling, finding connected components, computing centrality measures, and calculating many
Figure 5.6. A simplified depiction of some of the build paths in the automation script. Only the first transition is user-defined. Each step involves multiple invocations of many programs to properly assemble data and perform fair evaluation.

useful statistics.

LPmade specializes in link prediction by including commonly used unsupervised link prediction methods: Adamic/Adar, common neighbors, Jaccard’s coefficient, Katz, preferential attachment, PropFlow, rooted PageRank, SimRank, and weighted rooted PageRank. The library also has some simpler methods useful in producing feature vectors for supervised learners: clustering coefficient, geodesic distance, degree, PageRank, volume or gregariousness, mutuality, path count, and shortest path count. These methods may be selectively incorporated as features into the supervised framework in [68].

Several graph libraries such as the Boost Graph Library are brilliantly designed for maximum generality and flexibility with template parameters and complex inheritance models. One minor drawback to such libraries is that the code is complex to read and modify. The code base for this library takes a narrower approach by offering fewer mechanisms for generality, but as a result it has a much shallower learning curve.
5.2.3 GNU make Script and Supporting Tools

Although it can be used and extended as such, LPmade is not just a library of C++ code for network analysis and link prediction. It is additionally an extensive set of scripts designed for sophisticated automation and dependency resolution. These scripts are all included in a set of 2 co-dependent Makefiles: task-specific and common. Each new raw data set requires its own task-specific Makefile, which generally requires less than 20 lines of user code. This Makefile is where users specify the manner in which raw source data is converted to the initial data stream required by subsequent steps in the pipeline. It is also where rules from the common Makefile can be overridden for task-specific reasons. The common Makefile, called Makefile.common, includes all the general rules that apply to any network analysis or link prediction task once the task-specific Makefile is written to enable proper handling of raw input. The common Makefile script is designed with advanced template features that allow make to modify original Makefile rules in accordance with user requirements. Logical tasks are aggressively provided with their own rules so that the multi-core features of GNU make are of optimal benefit.

In general, users need not be familiar with writing Makefiles. The important options for the behavior of the automatic build system are presented at the top of the common Makefile. For instance, to predict within the 2nd and 3rd degree neighborhoods, set NEIGHBORHOOD := 2 3.

Figure 5.6 illustrates some build paths, and the sample calls below demonstrate several targets with their corresponding actions:

- **make -j 28 sm** - using 28 cores, build data streams from source, generate networks, run predictors, and perform evaluations
- **make -j 8 stats** - using 8 cores, compute several network statistics on the
complete network represented by the entire data set

- **make classify** - construct data sets then use parameters specified in Makefile to train, test, and evaluate

- **make -j 6 growth** - using 6 cores, generate growth information and plots to describe network saturation

Parallelism in these cases is all coarse-grained. Each rule in the Makefile script with no outstanding prerequisites is handled by a separate process to make use of additional cores.

For many large networks, link prediction and supporting analysis yields very large output files. When this prolific output is further combined into data sets, both the I/O capacity and bandwidth requirements may be problematic. To combat this, most steps in the work flow create, accept, and output gzip-compressed results. Especially on multi-core systems, this results in a hefty decrease in I/O capacity and bandwidth requirements with a minimal impact on performance. In most cases, the output from gunzip is produced faster than the consuming process can accept it. Where necessary, named pipes are used to ameliorate potentially large temporary storage requirements.

5.2.4 WEKA Modifications

WEKA has several limitations that make even its command-line mode problematic for operation on large data sets, specifically large testing sets. First, instead of treating test instances as a stream of data and processing them only an instance at a time, it builds a large data set by concatenating successive prediction output. Aside from the excessive memory consumption of this strategy, Java strings cannot exceed $2^{31} - 1$ characters in length, so WEKA terminates with an exception. We remove all unnecessary evaluation calculations and references
to in-memory data and instead print prediction information directly to standard output where it is consumed by high-performance C99 evaluation code built and distributed with LPmade. Next, we add support for operation on compressed C4.5 data files. C4.5 is much more convenient for scripted processing of large data sets. While the same functionality might be accomplished with process substitution or named pipes, the WEKA modification offers a more robust solution.

5.2.5 Documentation and Requirements

LPmade comes with a PDF user manual that describes all aspects of the software, most notably describing the setup process, how to use or extend the raw network library, and how to leverage the existing make script to complete many complex steps with short commands. The network library includes a testing architecture for testing and verification of individual binaries.

The C++ library is written in platform-independent C++ code using only STL extensions. The library may thus be built on any architecture and any operating system that provides a C++ compiler. An included set of high-speed evaluation tools is written in C99 and builds on any system with such a compiler. The bundled distribution of WEKA is cross-platform but requires version 1.5 or higher of the JRE. The automated build system requires GNU make. The common Makefile additionally employs a set of many standard tools like cut, paste, sed, awk, perl, sort, gzip, and bundled gnuplot 4.4.3.

5.3 Unification of DisNet and LPmade

DisNet attacks the problem of individual long-running computations in networks by taking advantage of the vertex-independent parallel slack of many in-
teresting graph problems. LPmade attacks the problem of complex work flows with many disparate tasks by combining optimized libraries with a sophisticated build system. Nonetheless, individual tasks within the LPmade build system may themselves be long running tasks with which DisNet could assist. One area of potential future work is to examine the possibility of a general combination of these two tools such that DisNet masters are automatically deployed by the LPmade framework to work on computationally intensive vertex-independent sub-problems. Since many link prediction methods, especially those based on path computations, require substantial effort at each source vertex, LPmade and DisNet would provide an excellent pairing to further boost the overall ease and speed of the link prediction work flow.
Thus far, we have explored the challenges in performing effective link prediction. Along the way, many observations naturally emerged about the nature of the link prediction task from the viewpoint of computational effectiveness. Now we turn our focus to one particular network heavily studied in the Interdisciplinary Center for Network Science and Applications at the University of Notre Dame: a large cellular phone call record network. This network and its related variants and derivatives have appeared in numerous publications including [8, 36, 38, 43, 68, 71, 90, 95, 112] to name only a few. This type of data is attractive foremost because it offers potential low-noise sources for frequent actual communication events between individuals, and these events can be used to construct large high-confidence social networks based on communication. It presents other opportunities too, such as offering a rare source of fine-grained temporal information about the communication events and potential links to demographic information. It even offers rough geospatial coordinates corresponding to event times [90, 112]. For our case study, we will limit considerations to demographic information, longitudinal information, and the topology described by the communication events themselves.

[43] and [95] exhibit goals somewhat similar to this case study in that they seek explanatory models for link persistence. [95] specifically follow a post-hoc analysis
paradigm in which the authors train an interpretable decision tree model on the data and examine the structure of the resulting tree to reach conclusions. We will be performing the same type of analysis, but we will operate on several fundamental types of data: topological, demographic, and temporal. Since the strength of some of the features in one type of data may overwhelm patterns in the others, we will first present the details of each type of data in isolation. Finally, we will combine all of the data types to observe the form of the model that results. The goal is to understand link dynamics better in terms of individual and combined perspectives of the data. This also aids in understanding the relationships that exist between perspectives.

Unlike in Chapter 3, where the goal was to derive a general framework for effectively accurate link prediction, the goal here is to develop an understandable model. Neural networks, support vector machines, decision forests, and other sophisticated techniques are difficult to interpret [32, 110]. Decision trees, on the other hand, are often easy to interpret, but link prediction often offers such a plethora of data for induction that the complexity of a typical C4.5 tree is quite high. We opt to use WEKA’s REPTree, similar to a standard C4.5 decision tree except that it offers reduced-error pruning and the option to limit the depth of the tree, both of which can increase model comprehensibility [93]. We first balance the classes in the training data and then construct trees with a maximum depth of 5. Although this depth limits the performance of the classification model, it generates a model that is both more interpretable and better targets fundamental driving forces in link formation.

The visualizations in the following sections obey several semantic conventions. Tree nodes are represented by either rectangles or circles connected by lines. Rect-
angles represent leaves and circles represent non-leaves. Rectangles corresponding to positive classification leaves are colored blue, and rectangles corresponding to negative classification leaves are colored black. The splitting feature in non-leaves is shown inside the circle representing the node. Branches to the left or right of the split indicate low or high values of the splitting feature respectively. Inside rectangular leaves, the number of correctly classified training instances is shown in green and the number of incorrectly classified training instances is shown in red.

6.1 Demographic Patterns

We first examine node attributes and demographics in isolation. Table 6.1 shows the features that are available to us, but some of the feature descriptions in the table are deliberately vague to protect sensitive information. The provided list of features naturally excludes agent identification features. Several of the features, such as the disconnection date, relate to customers choosing to terminate service, information that leads to solid conclusions about the possibility of the users forming links in the future. Some of these features may constitute leakage of information [51], but determining the severity of the leakage would require understanding the exact details of user account management as it affects our data. Cellular phone providers are aware of when their customers terminate service, and thus such leakage is a legitimate part of a deployable model. If, however, the awareness of service termination as it appears in our data predates the actionable awareness of service termination on the part of the provider with respect to future calls, leakage has occurred. Either way, service termination is a potentially powerful feature that probably resides outside the scope of a purely sociological
model of link formation. Customer usage in the current month is also a potential source of leakage if it contains information for the entire month.

These node attributes fall roughly into two categories: demographics and usage. Attributes such as location, age, and gender are obviously demographic in nature, but other attributes such as account_type, payment, and cost may hint less clearly at economic status. We bundle all of these attributes into a demographic model. In addition, we assist the learning algorithm by including several distance features that provide an indication of the geographical distance, relative economic situations, or distance in age of the source and target. Table 6.2 shows the precise set involved in training the demographic model and provides a key for Figure 6.1.

The model depicted in Figure 6.1 achieves an AUROC of 0.618 and an AUPR of 0.005260. The overarching observation to draw from the figure is that demographic distances, whether geographical location, age, or our surrogates for economic differences, are related to link formation mechanisms. All but one leaf on the low side of any proximity measure in the tree are designated as positive. Though we do not have information about race, our findings support the importance of geographic proximity at least for communication, if not for friendship [78]. Indeed, the geographic distance between individuals is not only the top split in the model, but it features prominently throughout the tree, and many fewer positive instances reside on the side of the tree indicating lower proximity. The addition of the difference function features to the model is apparently critical, since the splits that do not involve distance produce neither particularly large nor particularly pure leaves.
## TABLE 6.1

### NODE ATTRIBUTES IN THE CASE STUDY

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Description</th>
<th>Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>account_type</td>
<td>tier and purpose of service</td>
<td>nominal: 8 distinct values</td>
</tr>
<tr>
<td>ported_to</td>
<td>company after termination</td>
<td>nominal: 21 distinct values</td>
</tr>
<tr>
<td>usage_1</td>
<td>customer usage 7 months ago</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>usage_2</td>
<td>customer usage 6 months ago</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>usage_3</td>
<td>customer usage 5 months ago</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>usage_4</td>
<td>customer usage 4 months ago</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>usage_5</td>
<td>customer usage 3 months ago</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>usage_6</td>
<td>customer usage 2 months ago</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>usage_7</td>
<td>customer usage 1 month ago</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>usage_8</td>
<td>customer usage this month</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>start</td>
<td>when customer started service</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>end</td>
<td>if/when customer ended service</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>payment</td>
<td>postpaid or prepaid</td>
<td>nominal: 2 distinct values</td>
</tr>
<tr>
<td>cost</td>
<td>contract cost</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>location</td>
<td>customer postal code</td>
<td>numeric: non-continuous</td>
</tr>
<tr>
<td>age</td>
<td>customer age</td>
<td>continuous</td>
</tr>
<tr>
<td>gender</td>
<td>gender or unknown</td>
<td>nominal: 4 distinct values</td>
</tr>
<tr>
<td>rated_months</td>
<td>tracked service months</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>last_rated</td>
<td>last tracked service month</td>
<td>numeric: continuous</td>
</tr>
<tr>
<td>exit_code</td>
<td>why customer ended service</td>
<td>nominal: 5 distinct values</td>
</tr>
</tbody>
</table>
### TABLE 6.2

DEMOGRAPHIC NODE ATTRIBUTES

<table>
<thead>
<tr>
<th>Feature ID</th>
<th>Feature Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v_s$ account_type</td>
</tr>
<tr>
<td>2</td>
<td>$v_s$ payment</td>
</tr>
<tr>
<td>3</td>
<td>$v_s$ cost</td>
</tr>
<tr>
<td>4</td>
<td>$v_s$ location</td>
</tr>
<tr>
<td>5</td>
<td>$v_s$ age</td>
</tr>
<tr>
<td>6</td>
<td>$v_s$ gender</td>
</tr>
<tr>
<td>7</td>
<td>$v_t$ account_type</td>
</tr>
<tr>
<td>8</td>
<td>$v_t$ payment</td>
</tr>
<tr>
<td>9</td>
<td>$v_t$ cost</td>
</tr>
<tr>
<td>10</td>
<td>$v_t$ location</td>
</tr>
<tr>
<td>11</td>
<td>$v_t$ age</td>
</tr>
<tr>
<td>12</td>
<td>$v_t$ gender</td>
</tr>
<tr>
<td>13</td>
<td>$</td>
</tr>
<tr>
<td>14</td>
<td>$</td>
</tr>
<tr>
<td>15</td>
<td>$</td>
</tr>
</tbody>
</table>
Figure 6.1. Visualization of a demographic node attribute link formation model.

Attributes such as the provider to which the user switches, the usage characteristics, and when the customer started and ended service all reflect the nature of the relationship between the customer and the provider. We consider all of these node attributes usage features and train a separate model to determine what role these feature may play in link formation. Table 6.3 shows the set of features that we use to train the demographic model and provides a key for Figure 6.2.

Figure 6.2 achieves a performance of 0.530 AUROC and 0.003909 AUPR. This performance is particularly low, and it suggests that these features may not be aligned well with the mechanisms driving link formation in the network. The only surprise in this result is that usage information should correspond roughly with total call duration and text message volume in terms of actual bandwidth on the network. Though this differs from our own standard determination of usage volume by number of calls or number of text messages, it is interesting that these time-resolved alternative values for volume lead to such poor results. Nevertheless, we provide the results for visualization. Some of the splits are sufficiently pure relative to the performance on the testing set as to suggest that the model may
### Table 6.3

**Usage Node Attributes**

<table>
<thead>
<tr>
<th>Feature ID</th>
<th>Feature Name</th>
<th>Feature ID</th>
<th>Feature Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>vs ported_to</td>
<td>15</td>
<td>vt ported_to</td>
</tr>
<tr>
<td>2</td>
<td>vs usage_1</td>
<td>16</td>
<td>vt usage_1</td>
</tr>
<tr>
<td>3</td>
<td>vs usage_2</td>
<td>17</td>
<td>vt usage_2</td>
</tr>
<tr>
<td>4</td>
<td>vs usage_3</td>
<td>18</td>
<td>vt usage_3</td>
</tr>
<tr>
<td>5</td>
<td>vs usage_4</td>
<td>19</td>
<td>vt usage_4</td>
</tr>
<tr>
<td>6</td>
<td>vs usage_5</td>
<td>20</td>
<td>vt usage_5</td>
</tr>
<tr>
<td>7</td>
<td>vs usage_6</td>
<td>21</td>
<td>vt usage_6</td>
</tr>
<tr>
<td>8</td>
<td>vs usage_7</td>
<td>22</td>
<td>vt usage_7</td>
</tr>
<tr>
<td>9</td>
<td>vs usage_8</td>
<td>23</td>
<td>vt usage_8</td>
</tr>
<tr>
<td>10</td>
<td>vs start</td>
<td>24</td>
<td>vt start</td>
</tr>
<tr>
<td>11</td>
<td>vs end</td>
<td>25</td>
<td>vt end</td>
</tr>
<tr>
<td>12</td>
<td>vs rated_months</td>
<td>26</td>
<td>vt rated_months</td>
</tr>
<tr>
<td>13</td>
<td>vs last_rated</td>
<td>27</td>
<td>vt last_rated</td>
</tr>
<tr>
<td>14</td>
<td>vs exit_code</td>
<td>28</td>
<td>vt exit_code</td>
</tr>
</tbody>
</table>
be too closely fitting the training data despite the reduced-error pruning. In any case, we see no intuitive significance in this model.

6.2 Common Topological Models

Figure 6.3 shows a visualization of the model built from the standard topological link predictors along with the PropFlow predictor. It obtains an AUROC of 0.739 and an AUPR of 0.009660. Consistent with our prior presentation of results, the principal split in the model is based on Adamic/Adar. A high score is suggestive, but weakly so, of link formation. When the Adamic/Adar score is not sufficiently high to merit positive classification, Katz is used but only as a surrogate for reachability following directed edges. Where reachability exists, link formation is much more likely. Where there is no reachability, PropFlow makes the final determination. Notably, although it is not in the top splits, most of the instances fall through to the third split, so PropFlow is providing most of the discriminative power in this model, at least in terms of model error on the
balanced training set. In this case, the higher the PropFlow score, the more likely the model deems link formation to be.

Although it is easier to understand the meaning of individual nodes in this tree model, the model as a whole is somewhat difficult to understand. For instance, it is not clear precisely what the relationships and thus the interdependencies are between Adamic/Adar, Katz, and PropFlow. We now turn to VCPs to perform a structural model analysis in the cellular phone network.

6.3 Structural Patterns in Topology

Figure 6.4 shows the model that results from considering directed VCP$^{3,1}$, illustrated in Figure 3.7. It obtains an AUROC of 0.704 and an AUPR of 0.024197. The first split is determined by the presence of a reciprocal link. Those potential links where the reciprocal link already exists are much more likely to form. The next indicator is VCP$^{3,1}(15)$, which corresponds to a shared neighbor where the
source and target both link to the neighbor with a bidirectional link. This mutual relationship provides a good splitting option. When no such neighbor is shared, VCP$^{3,1}(11)$ indicates a less significant shared neighbor in which the target has only an out-link. This provides evidence that not only the sharing of neighbors but reciprocity in links to shared neighbors is a significant driver of link formation. Finally, VCP$^{3,1}(10)$, which corresponds to a neighbor to which both the source and target have out-links is strongly indicative of link formation when it occurs at least once. VCP$^{3,1}(13)$, which is similar to a bidirectional link between the shared neighbor and the target, is also strongly indicative if it is present.

Figure 6.5 shows the undirected VCP$^{4,1}$ model, which is illustrated in Figure 3.8. In the model, elements with addresses between 20 and 39 correspond in numerical order to those in the Figure 3.8 except with the presence of $e_{t,s}$. The model obtains an AUROC of 0.634 and an AUPR of 0.019225. Most notably in this model, we see that VCP$^{4,1}(24)$ and VCP$^{4,1}(4)$ are the first and second splits respectively. Potential links embedded in many of the VCP$^{4,1}(24)$ elements terminate in a positive leaf. Since we already know that Adamic/Adar is a strong
model for this data set, this is unsurprising. VCP\textsuperscript{4,1}(24) prevalence signifies rarely shared common neighbors, and element 24 specifically indicates a reciprocal link already exists. This is an exceptionally good split, as indicated by the numbers in the box. VCP\textsuperscript{4,1}(4) is the same structural embedding without the reciprocal link, and we observe that potential links frequently embedded in this structure also tend to form. For those potential links that do not fit the previous categories, embedding in VCP\textsuperscript{4,1}(14), which is a similar structure with decreased rarity of the common neighbor, also indicates likely link formation. In the absence of rare common neighbors, frequent common neighbors is still a good predictor in this network. Those potential links not within these categories are next separated by VCP\textsuperscript{4,1}(15), in which the source and target of the potential link have unshared but connected neighbors. If the potential link does not exhibit at least this amount of connectivity, it is unlikely to form in this network. If it does, the final discrimination in this depth-limited tree is based on VCP\textsuperscript{4,1}(17), an isolated source with a highly connected target. This indicates that despite our demonstration of the weakness of general preferential attachment in the cellular phone network, heterophilic attachment to eminent individuals may actually be contraindicated in the presence of few shared neighbors.

As Section 3.4 in Chapter 3 illustrates, these structural memberships can be strong predictors in the link prediction task. They can also provide decent explanatory power in an interpretable model. In longitudinal data, one might also wonder about how sequences of events might correspond to growth or link formation mechanisms.
6.4 The Significance of Event Times

We divide the cellular phone data set into two periods and assign edges relations based on period membership. The first period represents the first 4 weeks of data, and the second period represents the following 3 weeks of data. The training labels come from week 8. We construct a model using VCPs with two relations defined by these two periods in training. The presence of structures that contain both relations are indicators that a particular sequence of communications may be a significant indicator of link formation. Figure 6.7 shows the results. Figure 6.6 provides a key for mapping the temporal multi-relational element numbers to their corresponding link structures.
Figure 6.6. VCP\textsuperscript{3,2} key for the time-resolved multi-relational model. Elements 16 through 63 are identical to their modulo 16 counterparts except for the presence of \( e_{t,s} \) in one or more of the time periods.
Figure 6.7. Visualization of a temporal link formation model based on undirected VCP$^{3,2}$. 
The top split is the calling volume of the source in the first period alone. If that volume is low, the second split is on the calling volume of the source. If it is low, new links are likely to form. If source call volume in the second period alone is high, then the call volume of the target in isolated periods becomes an important determination. Element 15, which indicates high call volume to a shared neighbor in both periods, unsurprisingly suggests upcoming links between the source and target whenever it appears in the tree.

6.5 Grand Unified View

Finally, we combine all the data sets to see how different types of features interact. Many pairs of features have low correlations, so in cases where both features are useful, they offer descriptive diversity to the learning algorithm. Figure 6.8 provides the results. Since there are multiple data sets here, each split uses two numbers. The first number indicates the source data set: 1) demographic, 2) topological models, 3) time-resolved VCP structures, 4) directed 3-vertex VCP structures, and 5) undirected 4-vertex VCP structures. The second number indicates the feature in the data set. So 4,2 represents the second element in the a directed 3-vertex VCP, and 1-5 represents the age of the source from Table 6.2 above. We do not include the usage data features because of the apparently limited use of those features.

This model achieves a test set AUROC of 0.781 and a test set AUPR of 0.024838. It mostly employs as features elements from VCP$^{4,1}$, using only one element from the directed VCP$^{3,1}$ and one demographic feature. Despite the limited use of other sources of information, the unified model manages significantly higher performance on the test set than the next best contenders. It has 5.7%
higher AUROC on the test set than the HPLP-based reduced-error pruning tree. It has 2.7% higher AUPR on the test set than the VCP\textsuperscript{3,1}-based reduced-error pruning tree. Even with this simple depth-limited tree induction algorithm, the unified model clearly does better on the test data than its similarly limited component models.

The model first splits on whether potential links are embedded within lots of sparse subgraphs. Potential links that are not embedded in sparse subgraphs are classified as positive. Potential links that are embedded in sparse structures are next dependent on whether the source and target share many neighbors. In nearly all cases, a large number of shared neighbors leads to a positive classification. The positive classification likelihood is reinforced when the source and target are also geospatially proximate.

As a final note, it would also be fascinating to include geospatial proximity
indicators as another potentially orthogonal data source. The physical proximity of individuals through time has already been demonstrated to offer additional predictive power \[112\], and the way that the proximity of people influences or is influenced by their demographics, topological embedding, and link evolution is largely an open question. Unfortunately, we were unable to obtain this data in sufficiently raw form for analysis.
CHAPTER 7

CONCLUSION

The primary goal of this dissertation is to encapsulate and promote solutions for what we view as the three principal components of effective link prediction: the algorithms and methods necessary to achieve good predictions, the evaluation methods necessary to meaningfully measure and compare the quality of predictions, and techniques targeted at handling difficult computational and scaling issues. Through this continuing work, researchers and domain experts can achieve better results more quickly and with a more reliable understanding of their meaning.

We offer several significant new methods. PropFlow, though it is similar in spirit to existing path-based measures, is both faster to compute and more effective in most of the circumstances in which we tested it. The vertex collocation profile method offers an entirely new approach to link analysis and link prediction, and we provide a fast, non-trivial algorithm for computing members of isomorphism classes of four vertices. General classification frameworks have been proposed for and applied to link prediction, such as with the study by [4], but this work offers the first in-depth coverage of the challenges involved in supervised classification for link prediction.

Coming out of the coverage is an extremely effective classification scheme involving sampling and multiple ensemble techniques that allow for reasonable per-
formance on large link prediction tasks. By applying this classification scheme either to a mixture of lower and higher order features or to a vector of vertex collocation profiles, it is possible to achieve performance benefits of large factors. The novelty of the problem lies in its imbalance and the fascinating network-theoretic properties that intersect to allow or deny the possibility of strong predictors.

We additionally demonstrate, for the first time, the power of including temporal information encoded multi-relationally. Using information about when past links were observed rather than merely which links were observed increases performance over baseline models that do not use this information. The potential of vertex collocation profiles and other multi-relational prediction methods in incorporating time remains a large open area for exploration in link prediction research.

There are now so many creative approaches to performing link prediction that understanding their relative performance is becoming quite problematic. There are many reasons for this, some of which are related to the inaccessibility of research code implementing published methods, the lack of time on the part of researchers, the lack of space within publications, and the lack of a sane, standard evaluation method. Nonetheless, if we really hope to understand what methods are best for purposes of either application or extension, we must remedy this situation. We take a major step toward this remedy with our study of evaluation in the context of link prediction. We identify common mistakes and provide clear, concrete recommendations to offer a meaningful and fair evaluation method. This also paves the way for a disciplined comparative study of the many families of link prediction methods. For instance, we believe it would be interesting to study the comparative performance of standard supervised classification algorithms and the matrix alignment method [100].
Computational scalability and accessibility is a major problem with link prediction. Data sets are often extraordinarily large, reaching billions of instances, and the longitudinal nature of most link prediction tasks complicates proper deployment and evaluation. We tackled the scalability and computability issues with DisNet. The core observation underlying this framework is that most network analysis, particularly analysis related to link prediction, is naturally parallel. DisNet makes it easy for users to perform analysis on large networks by leveraging computational grids and clouds for them in a general and extensible manner.

LPmade tackles the accessibility problem by encoding the network analysis and link prediction tasks as a set of rules and dependencies. This provides a very general data interface that users can leverage to quickly explore their data. Because of the dependency tracking features and inherent multi-core parallelism of this approach, LPmade also reduces redundant computation and allows for additional exploitation of parallel computation opportunities. Due to our work, it is now simple to harness multi-core computational server or supercomputer resources for traversing the many disparate tasks necessary for fair link prediction and evaluation. Further, we have already put heretofore unfathomable vertex-independent computations in large networks within easy reach of novice programmers in fields such as sociology, biology, and physics. These two systems pave the way for an even more powerful synergy wherein LPmade incorporates DisNet and other systems such as MapReduce and Hadoop [103] for its long-running computations.

Considered together, this work represents many new achievements in link prediction literature with a holistic perspective that offers link prediction as a more usable and approachable tool in arbitrary domains. Advances will continue in the areas of algorithm development, evaluation methodology, and computational
efficiency, and it is doubtless from the fervent research that the future of link prediction is bright.
BIBLIOGRAPHY


107. Y. Sun, J. Han, C. C. Aggarwal, and N. v. Chawla. When will it happen? relationship prediction in heterogeneous information networks. In *Proceedings of the 2012 ACM International Conference on Web Search and Data Mining (WSDM)*, February 2012.


