FOO:
Exploiting BAR to Speed Up Software

Harry Q. Bovik

CMU-CS-04-xxx

May 2004

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Abe B. Carson, Chair
Darlene E. Franklin
Greg Harvey
Ira Lowenstein, U.C. Berkeley

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by the National Aeronautics and Space Administration (NASA) under grant nos. NAG1-2345 and NAG6-7890, and by generous donations from the Intel Corporation, IBM Corporation, and the NSF.

Keywords: FOO, BAR, Computers, Science

September 7, 2008
DRAFT

For graduation. How I love thee. Hopefully someday I will attain you.

Abstract

This thesis introduces the notion of *FOO*. It is good. I will show how to make software if you have FOO.

Acknowledgments

I love Jello. This stuff has fed me from the day I proposed until the day I defended. Jello is my friend, my sustenance, my very being.

Oh yeah. My advisor is cool too.

In addition, **Catherine Copetas** and **Sharon Burks** should be mentioned in a **large** font in everyone's Acknowledgements section, since they said so. (It is a required part of the thesis formatting guidelines.)

Contents

1 Introduction **1**

1.1 Contextual Search and Disambiguation 3

1.2 Learning 4

1.3 Case Studies 5

1.4 Scalability 6

2 Background **7**

2.1 Entity-Relation Networks 7

2.1.1 Spreading Activation 7

2.2 PageRank algorithm and its Variants 9

2.3 Lazy Graph Walks and Diffusion Kernels 10

2.4 Statistical Relation Learning 10

3 Framework **11**

3.1 Definitions and Notation 11

3.2 Graph walks and Query Language 13

3.2.1 Graph Walk Variants 15

3.2.2 Graph Walk Properties 16

3.3 Applicability 17

3.3.1 Structured and Semi-structured data as a Graph 18

3.3.2 Motivations 20

3.4 Related Work . . . 22
3.5 Discussion . . . 26

4 Learning **27**
4.1 Learning Settings . . . 28
4.2 Edge Weight Tuning: Error BackPropagation . . . 29
4.3 Re-ranking . . . 30
4.4 Path-Constrained Graph Walks . . . 33
4.5 Method Comparison . . . 36
4.6 Related Work . . . 38

5 Case Study: Personal Information Management (PIM) **43**
5.1 Email and Meetings Graph Representation . . . 44
5.2 PIM Tasks as Queries . . . 46
5.3 Experimental Corpora . . . 50
5.4 Experiments and Results . . . 51
5.4.1 Person Name Disambiguation . . . 51
5.4.2 Threading . . . 57
5.4.3 Meeting Attendees Prediction . . . 60
5.4.4 Alias Finding . . . 63
5.4.5 Predicting person-activity future involvement . . . 66
5.4.6 Message foldering and tracking . . . 69
5.5 Related Work . . . 73
5.6 Summary . . . 77

6 Applications of Parsed Text **79**
6.1 Parsed Text as a Graph . . . 80
6.2 Text Processing Tasks as Queries . . . 81
6.3 Experimental Corpora . . . 82
6.4 Exeperiments and Results . . . 83

6.4.1 Coordinate Term Extraction 85
6.4.2 General Word Similarity 89
6.5 Related Work 92
6.6 Summary 93

7 Design and scalability considerations **95**
7.1 Graph walk parameters 95
7.1.1 Walk Length 95
7.1.2 Stay Probability 98
7.1.3 Graph walk variants 99
7.2 Learning 101
7.2.1 Local vs. Global Learning 102
7.2.2 Combining Learning Methods 106
7.2.3 PCW thresholding 106
7.3 Scalability 108
7.3.1 Graph walks 108
7.3.2 Learning 114
7.4 Related Work 114

A I like Pie **119**

Bibliography **121**

List of Figures

3.1 A simple example of the considered graph scheme 12

3.2 Example graphs (left) and their corresponding graph schemas (right). . . . 19

4.1 An example path-tree. 34

4.2 Pseudo-code for path-constrained graph walk 35

5.1 A joint graph representation of email and meetings data 44

5.2 Person name disambiguation test results: Recall at the top 10 ranks, for baseline and plain graph walk, where the query includes a term only (denoted as G:U(T)), or term and file (denoted as G:U(T+F)) (left); and for all methods using contextual queries (T+F) (right). 54

5.3 Meeting attendee prediction results: Precision-recall curve 62

5.4 Person to email-address mapping: Precision-recall curve 65

6.1 A joint graph of dependency structures 81

6.2 Test results: Precision at the top 100 ranks, for the city name extraction task (top) and person name extraction task (bottom). 87

7.1 Precision-recall curves varying the walk length k for city name extraction (top) and person name extraction (bottom). The left graphs show the full curves, and the right graphs focus on the top of the lists (down to recall 0.2). These results were all generated using the MUC corpus. 96

7.2 aa. 100

7.3 Precison-recall performance for city name extraction from the MUC corpus for path constrained walks with varying thresholds, and graph walks with uniform weights. 108

7.4 The cumulative number of nodes visited at each step of the graph walk, for the city name extraction and person name extraction datasets, for increasingly larger corpora. 112

7.5 The cumulative number of nodes visited at each step of the graph walk (logarithm scale), for city and person name extraction and the MUC+AP corpus, using path-constrained walks with varying thresholds (left). The ratio between the number of nodes visited by the path-constrained walks relative to the unconstrained graph walks is presented in the right graphs. 113

List of Tables

5.1 Email and meetings node and relation types. (Inverse edge types are denoted by a superscript.) . . . 45
5.2 Query realizations of the considered tasks . . . 47
5.3 Person disambiguation corpora and dataset details. . . . 52
5.4 Example person name type distribution per dataset. . . . 53
5.5 Person Name Disambiguation Results . . . 55
5.6 Threading corpora and dataset details. . . . 57
5.7 Threading Results: MAP . . . 58
5.8 Meeting attendee prediction corpus and dataset details. . . . 61
5.9 Meeting attendees finding results . . . 62
5.10 Alias finding corpus and dataset details. . . . 63
5.11 Alias Finding Results . . . 64
5.12 Activity-person prediction corpora and dataset details. . . . 67
5.13 Person-activity prediction results: Recall at 20 . . . 67
5.14 Message foldering and tracking: corpora and dataset details. . . . 69
5.15 Message foldering results: MAP . . . 71
5.16 Message tracking results: MAP . . . 72

6.1 Corpus statistics . . . 83
6.2 Word synonym pairs: train and test examples . . . 90
6.3 Message tracking results: MAP . . . 91

7.1 Results of applying graph walks using uniform edge weights, varying the graph walk length parameter k ($\gamma = 0.5$). 97

7.2 Results of applying graph walks using uniform edge weights, varying the graph walk length parameter k ($\gamma = 0.5$):MAP. 98

7.3 Results of applying graph walks using uniform edge weights, varying the graph walk length parameter k ($\gamma = 0.5$):MAP. 99

7.4 Performance comparison (MAP) of graph walks with random weights (G:R), weight tuning (G:L), reranking using edge sequence features (Rrk$_{G:R}$) and the combination of weight tuning and reranking (Rrk$_{G:L}$). Reranking using the full set of features is denoted as Rrk^{+}. 102

7.5 Performance comparison (MAP) of graph walks with uniform weights (G:U), path constrained walk (PCW), reranking using edge sequence features (Rrk$_{G:U}$) and the combination of path constrained walks and reranking (Rrk$_{G:L}$). Reranking using the full set of features is denoted as Rrk^{+}. . 103

7.6 A comparison of path constrained walks performance, for different thresholds (MAP). 107

7.7 Average query processing time per dataset and different walk length k. . . 110

7.8 Average query processing time for the named entity extraction tasks, using graph walks of 6 steps and corresponding path constrained graph walks with varying thresholds. 111

Chapter 1

Introduction

Many tasks of text processing and information retrieval (IR) can be performed by clever application of textual similarity metrics: in addition to the canonical problem of *ad hoc* retrieval, which is often formulated as the task of finding documents "similar to" a query, textual similarity plays a prominent role in the literature for diverse tasks such as text categorization [104], data integration [24], summarization [87] and document segmentation [51].

In modern settings, however, documents are usually not isolated objects: instead, they are frequently connected to other objects, via hyperlinks, meta-data or relational structure. A few natural examples are XML documents [47], the Semantic Web [5] or email, where an email message is connected via header information to other emails and also to the recipient's social network [73].

The famous algorithms of PageRank [79] and HITS [61] were innovative in considering structural hyperlinks as a measure of document similarity, or document relatedness. In their view, the Web is a network of entities (documents), connected by directed edges (the physical hyperlinks). In particular, the PageRank model allows randomness in a surfer's behavior, such that every document is reachable along the course of a search, either via following a link or by 'jumping' to another page (*teleporting*). An inifinte random walk in this model then leads to a steady state, where probability distribution over nodes gives a measure of document centrality.

As PageRank and its variants study measures of centrality in a network, by definition these algorithms discard the initial distribution of the graph walk. An alternative line of research in this respect, which also emerged from the methodology of IR, is *spreading activation*. As in the case of linked wegpages, spreading activation is applied on entity-

relation networks. The edges linking the entities may be highly diverse – modeling a semantic taxonomy, for example. The mechanism of spreading activation includes assigning activation levels to nodes. Initially, the nodes associated with a given query are activated. Activation then propagates to adjacent nodes, where the output to the query includes those nodes that are active after a predefined number of propagation steps. The goal of the spreading activation framework as it emerged in IR was to enrich a query with related concepts. Unfortunately, in order to control activation flow effectively, activation propagation required careful manual design, where activation thresholds, contraints over paths and other contraints on activation flow had to be pre-set.

The focus of this thesis proposal is a a framework of *finite lazy graph walks* over entity-relation structures. Nodes in the underlying graph are typed, and the directed edges are labelled with the relevant relations. Similarly to PageRank, we suggest performing a random graph walk over this entity-relations network. However, as in spreading activation, we are interested in defining an extended measure of similarity between the objects in the network. In a *lazy* graph walk, there is a fixed probability of halting the walk at each step. This halting probability means that short walks will be more probable, and thus reduces the probability of reaching nodes distant from the starting points of the walk. Rather than modeling "centrality" of nodes, a lazy graph walk can be viewed as propagating "similarity" from a start node through edges in the graph—incidentally accumulating evidence of similarity over multiple connecting paths. The similarity metric can be viewed as a *tool for performing search* across the nodes in the graph.

While graph walks extract nodes in the network that are similar by virtue of their connectivity to the start nodes, the notion of similarity is often task dependent. Assigning generic weights to every edge type in the graph can be used to control the probability flow in the graph. In this thesis proposal we study two approaches to learning to better rank graph nodes for a given task: tuning the graph edge weights, and re-ranking the list of nodes output by the graph walk, using features that described global properties of the paths traversed to reach these nodes.

In previous work [100] lazy walks over graphs have been used for estimating word dependency distributions: in this case, the graph was one constructed especially for this task, and the edges in the graph represented different flavors of word-to-word similarity. Other papers have also used lazy walks over graphs for query expansion [103, 30].

In this thesis proposal, we suggest to study this framework and its properties, as well as develop supporting algorithms. In particular, we are interested in the following directions.

1.1 Contextual Search and Disambiguation

Compared with ad-hoc traditional IR, the suggested framework has the advantage of representing various objects types, whereas common IR indexing methods, like the TF-IDF vector model, are strictly textual. Thus, depending on application design, *implicit* context may be incorporated in the graph walk search. For example, in email data, terms are linked to the message file in which they appear. A search for related items to a particular term mention therefore allows incorporating context by adding the node representing the relevant email message to the initial distribution of the graph walk.

There are several key advantages of using the suggested framework over other approaches for performing contextual search. For exmaple, the lazy graph walks method can be constrasted with vector representation, where distributions are compared (e.g., using dot product) to derive a measure of object similarity. An advantage of our framework is that while such methods consider first-order adjacencies, the graph walk accumulates evidence of similarity also over longer range dependencies (considering similarity of similar objects). Representing long-range dependencies in a vector space would require exponenially larger distributions size, which is not feasible in practice. Thus, keeping data in its raw form, of linked entities, allows using deeper contexts on-the-fly, saving storage and pre-processing time.

Why is contextual search useful? the motivation for using context is two-fold. First, it allows for enriching a set of items with related information, thus potentially increasing the recall of a retrieval system. Another motivation is that context assists in entity disambiguation. We have shown that lazy graph walks are effective in entity disambiguation [73]. The problem investigated was of person name resolution. For example, consider a term that is known to be a personal name mention, like “Andrew”. Such a name mention in free text may refer to multiple persons, like “Andrew Ng” or “Andrew McCallum”. We have shown that our framework is successful in mapping name mentions to the corresponding *person* nodes in a graph representing an email corpus, by virtue of co-occurrence (shared contexts). Moreover, starting the graph walk from both the term and the email file in which this term appeared improved the ranking of the relevant person in the output distribution for ambiguous name mentions over the non-relevant persons candidates.

The formal details of the suggested framework are described in Section **??**. Experimental results for the personal name disambiguation and other contextual search tasks are included in Section **??**. An open question following the empirical evaluation, is how to eliminate side noise (involving irrelevant nodes in the walk). This question is partially answered by learning, that is discussed next.

1.2 Learning

The described graph framework can be used for many types of tasks, and it is unlikely that a single set of edge weights will be optimal. This suggests the goal of learning edge weights for a particular class of queries. Several researchers have suggested schemes for adjusting the set of edge weights using hill-climbing methods [35, 77, 3]. This group of methods can be adjusted from infinite to finite lazy graph walks. We have adapted an error backpropagation gradient descent algorithm [35] for tuning the graph weights in our framework.

A different approach that we have suggested learns to re-order an initial ranking, using features describing the edges in the traversed graph paths [73, 25]. This method parameterizes the graph walk with a set of representative features, and thus loses some information; however, such features allow one to capture certain "global" properties of the graph walk. Such features can describe edge label n-grams, that is, the sequences of edge labels that construct the set of connecting paths to the target node. Other feature we suggested was *source-count*, indicating the number of source nodes which have connecting paths to a candidate node in the output list. This feature models the assumption that output nodes that were reached from multiple source nodes in the graph walk are more relevant than those reached from a smaller subset of the start distribution.

A main difference between the weight tuning and the re-ranking approaches is that adjusting the graph parameters is based on "local" information only, while a re-ranker can use features derived from the full paths. On the other hand, representing the graph walk as discrete features necessarily loses some quantitative information. Given this trade-off, we evaluated the two approaches empirically for a shared set of tasks, using authentic data.

The learning methods are described in detail in Section **??**. Experimental results of applying learning on a set of evaluated tasks, as well as the empricial comparison of weight tuning and re-ranking, are presented in Section **??**.

In the proposed research we are interested in exploring the potential of learning for large graphs and longer graph walk length k. The main challenges are learning the graph edge weights efficeintly under such conditions. For re-ranking, we would like to identify feature sets which are sufficiently compact in describing longer graph walks (the set of n-gram edge sequences, for example, grow exponentially with k), as well as suggest other useful features, in addition to the features evaluated thus far. In order to make re-ranking scalable to large graphs we are also interested in efficient derivation of the feature vectors. A possible venue is to describe a target node in terms of the paths connecting to it, based on partial or approximated information. The proposed work regarding learning in this

framework is further detailed in Section 8.

1.3 Case Studies

The suggested framework is general whereas we believe it is applicable to various domains. We would like to investigate at least two different domains in the proposed thesis.

One domain that we have studied so far is of personal information management (PIM). We have shown that email data, as well as meeting entries from one's personal organizer, can be represented as a joint graph, including entities like *persons*, *dates*, *email-addresses* etc. We find that unlike other methods, lazy graph walks over this network naturally integrates relevant textual and non-textual objects – that is, it combines text, recipients information and a timeline. Further, we have shown that many useful email- and meeting-related applications can be phrased as search queries in the suggested framework. In particular, we evaluated the tasks of *person name disambiguation*, *threading*, *finding meeting attendees* and *finding email aliases*. At least for some of these tasks, labelled data can be acquired automatically, facilitating the optimization of system performance via learning. We have evaluated these tasks individually, comparing them to viable baselines. The conducted experiments, using multiple corpora (including the public Enron corpus) proved our framework to be superior to other methods. Beyond the "hard" measures considered, we also believe that our approach of presenting ranked results is adequate for the purpose of personal information management by a human user as a user who is reasonably familiar with the data can quickly validate the relevance of the returned items in the list, scanning it top-down. It would be the user's decision then at which rank in the list to stop, depending on his personal preference given the precision-recall trade-off.

Section **??** provides an overview of personal information representation as a graph, and the related tasks. The empirical evaluation of these tasks is given in Section **??**. Additional tasks that we would like to apply in this framework in the domain of personal information management are detailed under proposed work (Section **??**).

A second domain that we would like to apply our framework to is text processing. More specifically, we intend to represnt word mentions as nodes, and the syntactic structure that binds these words as labelled edges denoting the relevant relations. The syntatic information we consider representing is dependency parse information. In recent years, relatively efficient and accurate depedency parsers have become available. In a dependency parsing paradigm, an individual sentence is represented as a graph, where nodes contain the words, and directed edges denote a set of functional relations between them (e.g., *subject*, or *direct object*). We would like to unify the dependency structures derived

from a large corpus (or corpora) of sentences in a large graph. We believe that performing lazy graph walks in this graph would allow learning an extended textual similarity measure that considers the immediate relations between words as well as long-distance dependencies. Given a sufficiently large graph, an example application is to search for items similar to a word or concept of interest. This task, which has drawn some interest in the research community over the last years, should be useful for automatic or semi-automatic construction of knowledge bases. We hope that learning can be applied to direct and improve the graph walk results per particular tasks in this domain. Preliminary results, where we evaluated the extraction of coordinate terms (specifically, city names) given a seed of relevant items, suggest that learning can be beneficial in these settings. The preliminary results and specification of proposed work of text representation and processing are given in Section **??**.

As graphs representing text corpora are expected to be very large, the applicability of this framework to text processing tasks requires us to address scalability issues, related to both the graph walk implementation (see Section 1.4) and learning (as detailed in Section 1.2).

1.4 Scalability

The lazy graph walk paradigm can be implemented via sparse matrix multiplication. That is, the graph walk is calculated by iteratively multipying a distribution vector with the transition matrix. While the complexity of full matrix multiplication is $O(n^3)$, where n is the matrix dimension, this complexity is reduced for sparse matrices. However, for large n values, and for dense distribution vectors, full computation may become very slow. We are therefore interested in considering alternative methods for calculating the graph walk results. This issue is part of the proposed future research directions, detailed in Section **??**.

Insert introduction here.

Chapter 2

Background

2.1 Entity-Relation Networks

Entity-Relation (ER) networks are graphs, in which the vertices represent entities, and directed edges represent inter-entity relationships. Prototypical ER networks that are of interest to this work are:

Associative networks - this is a generic network, where the set of node types and link types is determined by the data available. An intuitive example of associative networks may be an organizational schemas ('person a *reports* to person b'). Another more general example is relational data, which defines objects of various types as well as the relations (typed or untyped) between these objects.

Semantic networks - in these networks nodes represent *concepts*, and the edges represent semantic relations between them. WordNet, a lexical taxonomy of English, is an example of a semantic network. Important semantic relations in a taxonomy include hyponymy (A is a kind of B), hypernymy (A is a superordinate of B), synonymy (A denotes the same as B) etc. Semantic networks are also a paradigm for world-knowledge representation, used as a platform for logical inference (consider sentences like 'bear *is a* mamal','bear *has* fur') [94].

2.1.1 Spreading Activation

The Spreading Activation (SA) Model is based on supposed mechanisms of human memory operations. Originated from psychological studies [83, 85], it was first introduced in

Computing Science in the area of Artificial Intelligence to provide a processing framework for semantic networks. In this method, "activating" some node in a network leads to iteratively activating adjacent nodes, thus reaching a broad context from an initial distribution.

There are many ways of spreading the activation over a network (a review is available in [31]. In its simple form, SA computes the input signal I_j to node j as a weighted sum of the activation levels of the nodes i connected to j:

$$I_j = \sum_i O_i w_{ij}$$

where O_i is the activation level of node i, and w_{ij} is the weight of the link connecting node i to node j. The weights may be real or binary values. A node's activation level is usually computed as a function of the input signal:

$$O_j = f(I_j)$$

where example functions are the threshold, linear and sigmoid functions. After the node has computed its output value, it fires it to all the nodes connected to it. If the edge weights are binary, this process is often referred to as *marker passing* [40].

The result of the SA process is the activation level of nodes reached at termination time. The interpretation of the level of activation of each node depends on the application, as well as the characteristics of the object being modelled by that node.

Drawbacks of the described general approach is that the activation ends up spreading over all the network. For this reason, and in order to use the information captured in the edge labels, the following heuristic constraints are often implemented:

distance constraints - cease SA once reaching nodes that are far (in terms of links traversed) from the initially activated nodes. It is common to consider only first, second and third order relations.

fan-out constraints - cease SA at nodes with very high downstream connectivity (fan-out).

path constraints - spread activation using preferential paths, reflecting domain specific inference rules. This can be modelled using the edge weights or, if links are labelled, by diverting the activation flow to particular paths while stopping it from following other paths.

activation constraints - It is possible to assign different threshold levels to each node or sets of nodes, considering their meaning in the context of the application. This allows implementing various complex inference rules.

Since its peak in the Eighties, there has been relatively little research activity related to the spreading activation paradigm in the area of information retrieval. A possible reason for that is that designing and adapting the various constraints that optimize activation flow in SA require a substantial manual effort. Furthermore, the underlying graphs had to be often manually crafted as well, for a given domain. We believe that the framework suggested in this thesis proposal addresses some of these shortcomings. First, a probabilistic graph walk inherently applies the principles underlying the abovementioned distance and fan-out contraints. Further, path constraints and the importance (weights) of different link types in the network are learned, rather then set manually. Finally, we construct the graph in an automatic, data driven, fashion.

2.2 PageRank algorithm and its Variants

The famous *PageRank* algorithm [79] represents Web pages as nodes in a graph. If there exists a physical hyperlink from page x to page y, then a corresponding directed edge is added to the graph. Thus, this model can be viewed as a simple associative ER network, where nodes are of uniform type, and there is a single type of edges. The sufer behavior is modeled as follows: given that the surfer is at node (page) i, then with probability d the surfer chooses to move to node j that has an outgoing link from i, and with probability $(1-d)$ the user will "jump" randomly to some page in the network (reset). Given that the user chose to follow a link, or to reset, the probability of node i is distributed uniformly over the relevant set of nodes. That is, the transition matrix is constructed as follows:

$$M = (1-d)[\frac{1}{N}]_{N\times N} + dL$$

where

$$L_{ji} = \begin{cases} \frac{1}{pa(i)} & \text{if there is an edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

, pa(i) is the set of nodes that have an outgoing link to i (its parents), and the total number of nodes (pages) is N.

PageRank dumping factor: This ensures that the MC has a unique stationary distribution (Bremaud, 1999), and in practice also prevents the chain from getting stuck in small loops (Brin and Page, 1998).

Under this model, the transition matrix is ergodic and has a stationary distribution. The *PageRank score* of node j, p_j, is defined as its probability in the stationary state, giving a measure of document centrality, or connectivity, in the network. This score can

be computed by repeating the following recursive formula until convergence:

$$p_j = \frac{(1-d)}{N} + d \sum_{i \in pa(j)} \frac{p_i}{ch(i)}$$

where $ch(p_j)$ is the set of outgoing links from j (its children). This is equivalent to the primary eigenvector of the transition matrix M.

2.3 Lazy Graph Walks and Diffusion Kernels

A *lazy graph walk* is a random walk, where the process remains at the current vertex with some probability.

Using ideas from spectral graph theory, a class of kernels on graphs, *heat diffusion kernels*, has been proposed [63] for handling categorical data. It has been shown that diffusion kernels are the continuous time limit of lazy random walks. The diffusion kernel function can be regarded as a sum over paths from i to j, namely the sum of the probabilities that the lazy walk takes each path. We note that diffusion kernels are applicable to symmetric graphs, as the kernel function must be symmetric.

There are multiple works in the area of machine learning addressing the problem of semi-supervised clustering using methods that directly apply or can be interpreted as random graph walks. Given a graph in which some of the nodes are labeled, the link structure of the graph is exploited to infer the labels of the remaining unlabeled nodes.

Another work has applied a Markov diffusion process in the settings of semi-supervised clustering [96]. In this work, a local metric defines probabilities of transitioning between two nearby points in one time step, and a global distance is constrcuted as the probability of transitioning between two points in t timesteps. Thus, they consider all the paths of length t on this graph. In particular, the authors have considered the settings of an undirected graph, where a node is connected to its k-nearest neighbors and the local transition probabilities are derived from a given distance (or, similarity) score for the relevant nodes.

2.4 Statistical Relation Learning

Chapter 3

Framework

In this section we give the formal definition of the considered underlying graph structure and the graph walk mechanism for deriving a similarity measure between nodes in the graph. In addition, we formulate the interface between a user and the system in the form of a query language as the system's input, and a ranked list as the system's output.

3.1 Definitions and Notation

A graph G consists of a set of nodes, and a set of labeled directed edges. Nodes will be denoted by letters such as x, y, or z, and we will denote an edge from x to y with label ℓ as $x \xrightarrow{\ell} y$. Every node x has a type, denoted $\tau(x)$, and we will assume that there is a fixed set of possible types. We will assume for convenience that there are no edges from a node to itself (this assumption can be easily relaxed). We will assume that edge labels determine the source and target node types: i.e., if $x \xrightarrow{\ell} z$ and $w \xrightarrow{\ell} y$ then $\tau(w) = \tau(x)$ and $\tau(y) = \tau(z)$. However, multiple relations can hold between any particular pair of nodes types: for instance, it could be that $x \xrightarrow{\ell} y$ and $x \xrightarrow{\ell'} y$, where $\ell \neq \ell'$. Note also that edges need not denote functional relations: for a given x and ℓ, there may be many distinct nodes y such that $x \xrightarrow{\ell} y$. Finally, for every edge in the graph there is an edge going in the other direction, denoting an inverse relation. That is, the graph is cyclic and highly connected.

For example, see the graph depicted in Figure 3.1. In the Figure, node types are denoted by the different shapes of a circle, sqaure and hexagon. The edges have different types as well, denoted by different line styles. Suppose that a circle represents a node

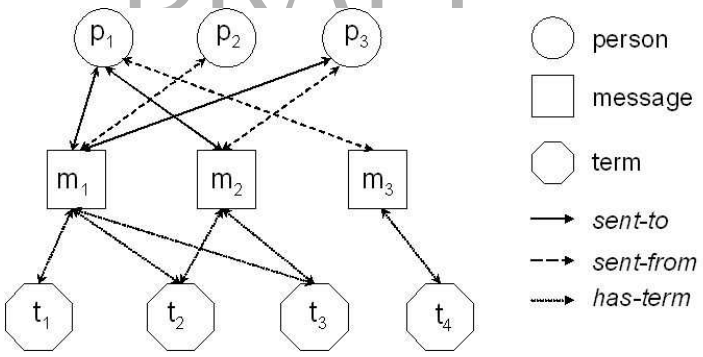


Figure 3.1: A simple example of the considered graph scheme

of type *person*, a square represents an *email message*, and an hexagon stands for a *term*. The dotted edges (e.g., $m_1 \rightarrow t_1$) may then represent a relation of *has-term*, pointing from a message node to the terms it contains. (For simplicity, the edges are marked as bi-directional in the Figure; in practice, however, the inverse relation – e.g., *has-term-inverse* – is represented by a seperate edge in the opposite direction). Similarly, the dashed edges may represent a relation of *sent-from*, directed from an *email message* node to a *person* node, that is recognized to be the sender of that message. As shown in the Figure, there may be multiple types of relations between given types of nodes. For example, the solid edges may represent a *sent-to* relation, drawn from an *email-message* node to a *person* node, that is known to represent its recipient.

Edge weights

Similarity between two nodes in the graph is defined by a graph walk process, where the walk on the graph is controlled by a set of parameters Θ. An edge of type ℓ is assigned an edge weight θ_ℓ. Let L_{xy} denote the set of edge types of the outgoing edges from x to y. The probability of reaching node y from node x over a single time step is defined as:

$$Pr(x \longrightarrow y) = \frac{\sum_{\ell \in L_{xy}} \theta_\ell}{\sum_{y' \in ch(x)} \sum_{\ell' \in L_{xy'}} \theta_{\ell'}}$$

where $ch(x)$ denote the set of immediate children of x (the set of nodes that are reachable from x in one time step). That is, the probability of reaching node y from x is defined as the proportion of total edge weights from x to y out of the total outgoing weight from x.

Continuing the example per the graph in Figure 3.1, the set of edges corresponding to this graph includes six types, namely $L = \{$ *has-term*, *has-term-inverse*, *sent-from*, *sent-from-inverse*, *sent-to*, *sent-to-inverse* $\}$. The set of parameters Θ corresponding to this

graph includes the the weights of these edges. For example, one possible assignment of the parameter values Θ is the following:

$$\begin{aligned}
&\theta_{has-term} = 2 \\
&\theta_{has-term-inverse} = 2 \\
&\theta_{sent-from} = 4 \\
&\theta_{sent-from-inverse} = 3 \\
&\theta_{sent-to} = 5 \\
&\theta_{sent-to-inverse} = 4
\end{aligned}$$

Given this parameter set, the probability of reaching node t_1 from node m_1 in a single step, for example, is calculated according to the given graph topology, as follows:

$$Pr(m_1 \longrightarrow t_1) = \frac{\theta_{has-term}}{3 \times \theta_{has-term} + \theta_{sent-from} + 2 \times \theta_{sent-to}} = \frac{2}{3 \times 2 + 4 + 2 \times 5} = 0.1$$

The graph edge weights Θ can be set uniformly, randomly, manually according to prior beliefs and experimemtation, or using a learning procedure, as will be discussed in the next chapter.

3.2 Graph walks and Query Language

Conceptually, the edge weights above define the probability of moving from a node x to some other node y. One can recursively define $Q(x \xrightarrow{=d} z)$, the probability of moving from x to z in exactly d steps, as follows:

$$\begin{aligned}
&Q(x \xrightarrow{=0} z) = 1 \\
&Q(x \xrightarrow{=d} z) = \textstyle\sum_y \Pr(y|x) \cdot Q(y \xrightarrow{=d-1} z)
\end{aligned}$$

Given a fixed probability γ of stopping the walk at each step, the probability $Q(z|x)$ of stopping at z in an infinitely-long walk from x is defined as

$$Q(z|x) = \gamma \sum_{d=1}^{\infty} (1-\gamma)^d Q(x \xrightarrow{=d} z)$$

This can be approximated by limiting the summation to some maximal value k.

Add? Toutanova: A Markov chain defines a distribution over sequences of states, via a generative process in which the initial state S0 is first sampled from according to p0, and then states St (for t = 1, 2, . . .) are sampled in order according to the transition probabilities.

In general, the walk propagates similarity to a start distribution over nodes through edges in the graph—incidentally accumulating evidence of similarity over multiple connecting paths. Larger values of γ increase the weight given to shorter paths between x and a destination node z. If V_0 gives probability 1 to some node x_0 and probability 0 to all other nodes, then the value given to z in V_T can be interpreted as a similarity measure between x and z.

For example, consider a graph walk starting from node m_1, in the graph described in Figure 3.1. A walk of one time step would reach those nodes that are immediately connected to m_1, namely $\{p_1, p_2, p_3, t_1, t_2, t_3\}$. Continuing the walk for an additional time step would propagate similarity from these nodes to their adjacent neighbors. After two time steps, node m_2, for example, would acquire probability mass due to the following set of connecting paths:

$$m_1 \overset{sent-to}{\longrightarrow} p_1 \overset{sent-to-inverse}{\longrightarrow} m_2$$
$$m_1 \overset{has-term}{\longrightarrow} t_2 \overset{has-term-inverse}{\longrightarrow} m_2$$
$$m_1 \overset{sent-to}{\longrightarrow} t_3 \overset{sent-to-inverse}{\longrightarrow} m_2$$

Node m_3 would acquire probability mass due to two paths:

$$m_1 \overset{sent-to}{\longrightarrow} p_1 \overset{sent-from-inverse}{\longrightarrow} m_3$$
$$m_1 \overset{sent-to}{\longrightarrow} p_3 \overset{sent-to-inverse}{\longrightarrow} m_3$$

and so on. The probability cumulated at each node thus reflects graph topology. The exact probability attributed to each node depends also on edge labels and the parameters Θ, γ and k.

Query Language

In this framework, a *query* is an initial distribution V_q over nodes, plus a desired output type τ_{out}, and the answer is a list of nodes z of type τ_{out}, ranked by their score in the distribution V_T.

Consider our running example. In the described domain, one may be interested to find people that are related to a particular term, for example t_2. In general, a term or a set of

terms may represent a concept of interest, such as a project name, or a specialized key word. This query can be easily formed as $Q = \{V_q = \{t_2\}, \tau_{out} =$'*person*'$\}$.

3.2.1 Graph Walk Variants

Added:

Theorem 1.2. [Jeh and Widom 03, Fogaras 03] Suppose that a number L is chosen at random with probability PrL = i = c(1 - c)i for i = 0, 1, 2, ... Consider a random walk starting from some page u and taking L steps. Then, for the vth coordinate PPV(u, v) of vector PPV(u), PPV(u, v) = Prthe random walk ends at page v.

In addition to the graph walk model described above, there are other graph walk variants. We discuss several variants in this section.

A graph walk paradigm defined as random walk with restarts is defined as follows:

$$V_{d+1} = (1 - \gamma)MV_d + \gamma V_0$$

where M is the transition matrix and V_q is denoted as V_0 for consistency. **It can be shown that our phrasing is equal to RWR. Show it here?**

Another variant of a graph walk is *lazy graph walks*. A *lazy graph walk* is a random walk, where the process remains at the current vertex with some probability. Here, a stay probability is applied.

$$\mathbf{M}_{xy} = \begin{cases} (1-\gamma)Pr(y|x) & \text{if } x \neq y \\ \gamma & \text{if } x = y \end{cases}$$

Another distinction is the probability mass distribution schema (normalized or not). First, select the edge to be traveresed. Then, distribute the edge weight uniformly among all nodes that are connected to the node over that edge type. (the BMC paper. also, ObjectRank).

$$Pr(y|x) = \sum_{\ell} Pr(x \xrightarrow{\ell} y|\ell) \cdot Pr(\ell|T(x)$$

In this thesis we take the approach that ...

3.2.2 Graph Walk Properties

The graph walk paradigm has several inherent preferences in ranking the graph nodes. We next discuss these graph walk properties. While generally desired, these preferences may obviously be ideal or sub-optimal, depending on settings as well as user preferences.

- The framework applies an exponential decay to the transmitted probability mass with path length (due to the parameter γ). This implies that nodes in the graph that are connected to a query node over *shorter* connecting paths are considered in general more relevant.

  For example, the email-message m_3 will be considered less similar to the terms $V_q = \{t_1, t_2\}$ compared with m_1 or m_2, since it is connected to the query nodes via paths of length 3, whereas the other two messages are associated to the terms with a direct *has-term-inverse* relation.

  Implementation-wise, the exponential decay factor leads to fast convergence of node rankings. Therefore, small values of walk length k should provide a good approximation of node scores. (This will be further discussed and evaluated in Chapter 7.)

- Evidence of similarity is accumulated at each node over multiple connecting paths. That is, a node that is linked to the query distribution over a large number of paths will be considered in general more relevant than nodes connected over fewer paths.

  For example, in the graph schema described in Figure 3.1, the email-message m_1 will be considered more similar to the phrase $V_q = \{t_1, t_2\}$ compared to m_2, since there are two paths connecting the initial distribution to m_1:

  $$t_1 \overset{has-term-inverse}{\longrightarrow} m_1$$
  $$t_2 \overset{has-term-inverse}{\longrightarrow} m_1$$

  whereas there is a single paths leading to m_2:

  $$t_2 \overset{has-term-inverse}{\longrightarrow} m_2.$$

- The framework includes an inherent Inverse Document Frequecy (IDF) weighting scheme. Suppose we restrict ourselves to only two types, *terms* and *files* and allow only *has-term* edges, as is the case in traditional IR settings. Now consider an initial query distribution, which is uniform over the two terms "the aardvark". A one-step

graph walk will result in a distribution V_1, which includes file nodes. The common term "the" will spread its probability mass into small fractions over many file nodes, while the unusual term "aardvark" will spread its weight over only a few files.

In our toy example, the probability mass attributed to m_1 over a single time step due to path $t_1 \overset{has-term-inverse}{\longrightarrow} m_1$, starting from $V_q = \{t_1, t_2\}$ will be doubled compared with the probability mass transmitted by the path $t_2 \overset{has-term-inverse}{\longrightarrow} m_1$. The reason being that as shown in Figure 3.1, t_2 is mapped to two message nodes whereas t_1 is linked to a single message node. (It is assumed here that equal initial probability was assigned to t_1 and t_2.) Hence, node connectivity has a similar effect as the use of an IDF weighting scheme.

- As a consequence of probability cumulation at graph nodes throughout the graph walk, the resultant similarity measure is inclined towards nodes that are highly connected (central) in the graph.

  For instance, suppose that given the query $V_q = \{t_1, t_2\}$, we are interested in retrieving relevant entities of type τ_{out} ='*person*'. A graph walk of two steps will assign higher score to nodes p_1 (and p_3) compared with p_2, since p_1 is associated with multiple *message* nodes and thus is credited with probability mass via a larger number of connecting paths, compared with p_1.

- Finally, the edge label weights Θ provide an additional mechanism for affecting the probability flow in the graph. As an extreme example, suppose that edge types $\theta_{sent-to}$ and $\theta_{sent-from}$ are assigned a weight of zero in Figure 3.1. In that case, the graph will degenerate into a bipartite *message-term* schema. In the more general case, however, edge weights reflect relation importance and allow (limited) control over node preferences.

3.3 Applicability

This chapter first discusses the graph schemas that are considered in this thesis, and representation limitations. In the second part of this section, we discuss the various possible motivations (or, counter motivations) for using the framework. While the framework described so far is not novel, a main contribution of this thesis is in extending and fomralizing its applicability as a general purpose tool. This chapter is therefore described in special detail.

3.3.1 Structured and Semi-structured data as a Graph

Many real-world data and problems can be modeled as a graph in the specified framework. The description of the web as a network, where nodes denote wepages and directed links represent the hyperlinks between them [88, 79] can be considered as a private simple case of the framework here. This example of a *homogenous* graph, where there is a single type of nodes and a single type of edges, is shown in Figure 3.2(a). Similarly, scientific papers (or other publications) can be represented as a graph, where relationships between papers are inferred from their cross-citations [92], and so on.

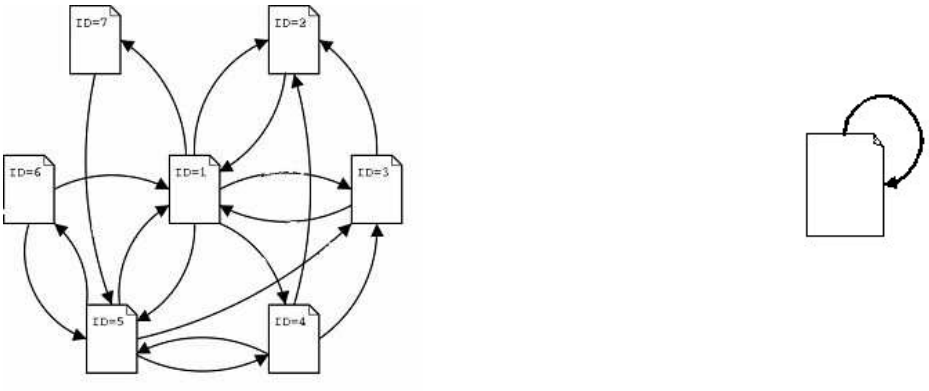
Another generic type of graph that can readily be represented in this framework are recommender (also known as *collaborative filtering*) systems. In this domain, a user's preference for an item constitutes a relationship between the user and the item. This problem is naturally modeled as bi-partite graphs. The underlying graph schema in this case includes two types of nodes and a single type of edge, as demonstrated in Figure 3.2(b).

In general, however, the framework is adequate also for domains that demonstrate richer sructure, where the relation schema includes multiple types of nodes (objects) and various relations between them. Figure 3.2(c) gives a representation of email data (which extends our toy example from Figure 3.1. In this representation, node types include *message*, *person*, *email-address*, *term* and *date*. The graph edges correspond to various inter-entity relations, such as *sent-from*, connecting a message to its sender; *alias*, linking a *person* node to its *email-address*, etc. Most of the relations can be readily parsed from an email header. The email content is treated as a bag-of-words, where each unique word is represented as a *term* node, and is linked to the *messages* which contain it. Other similar examples include citation represenation, where related entities like *author*, *venue* and *year* are represented in a relational structure. An ontology such as WordNet [41] forms a network whichs include a single type of entity (words), but multiple types of edges, including *hypernym*, *hyponym* relations and so on.

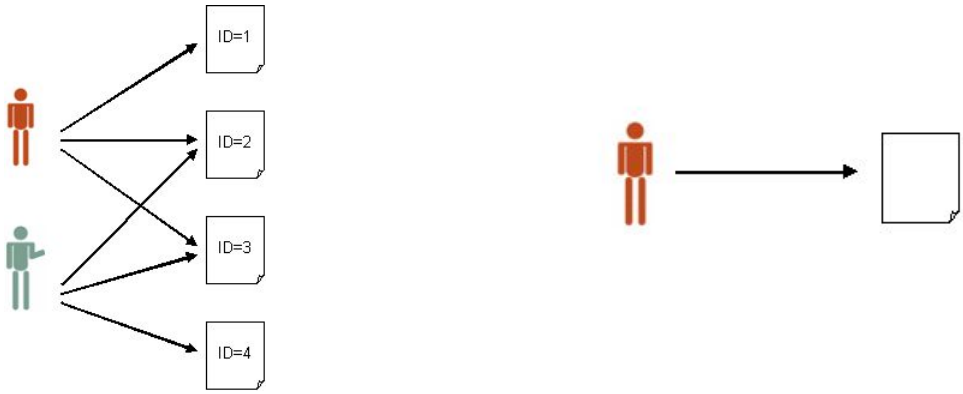
Representation Limitations

Graph schema. We notice that for each of the above examples, described in Figure 3.2, there is a clear graph schema. That is, closed sets of entity types and the possible inter-relations between them are well defined. The graph edge weights Θ correspond to the given finite set of edges. Domains where the full graph schema is not known (such as in an evolving domain), may be applicable by adjusting the model. For example, default edge weights may be defined for unknown edges. This case, however, is out of the scope of this thesis. In addition, domains where the graph schema is very large, in proportion to the observed data, pose a challenge for setting the edge weights parameters Θ – whether these are set randomly, manually, or using learning. The semantic Web may be a relevant

a. WebPages and hyperlinks



b. Recommendation Systems



c. Email

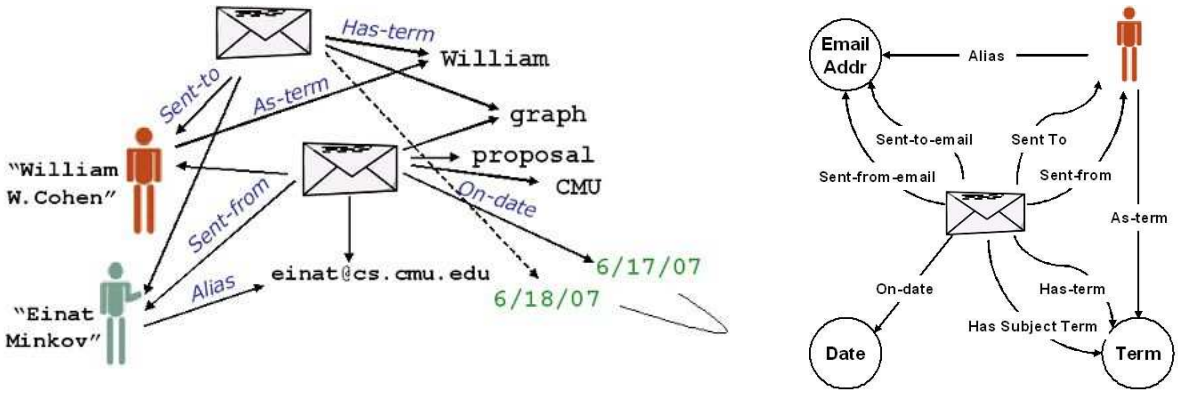


Figure 3.2: Example graphs (left) and their corresponding graph schemas (right).

example, which is of great interest. We believe that at least some of techniques presented in this thesis will be beneficial in such cases. However, this remains a venue for future study.

Entity attributes. Entity-relation schemas, as used in database terminology, may include entity attributes, in addition to the entities and their inter-relations. For example, an entity such as a *employee* may be linked to a node denoting a *company* and demonstrate other relations. In addition, it may have attributes such as *name*, *age*, *address* etc, where each instance of *employee* is associated with relevant attribute values (e.g., *name:* John, *age:* 42, ...). In our schema, attributes values can be modelled as additional nodes, and links added between attribute-entity pairs. It is also straight-forward to define such links to be *non-walkable*, and only use them for a *filtering* operation (similarly to a "where" statement in SQL). Real-value attributes, however, can not be modelled as nodes.

distinguish btw pre-designed graphs and graphs that naturally represnt the data.. Also, directed versus uncdirected graphs.

Real-values. A limitation of the framework is that it does not readily support real-values representaion. It is possible to weight edges by importance, confidence etc. (Rather than distribute the weight θ_l among all outgoing edges of type l uniformly, θ_l can be distributed among the relevant edges according to their individual weights). However, nodes in the representation considered here necessarily correspond to discrete entities. Thus, in case that real-value attributes are to be incorporated, for example, they must be discretized first.

Agarwal KDD 2006: In contrast, from the associative network viewpoint, edge (u, v) connotes similarity, carefully encoded using domain knowledge as a fixed weight w(u, v);

3.3.2 Motivations

Associative retrieval. Graph walks can be viewed as an implementation of notion of transitive similarity. It may therefore be used as a mechanism for expanding or enriching a set of entities with related objects. In the general framework of information retrieval, while a query usually corresponds to a single or a small number of objects, graph walks can be applied in order to *expand* the query nodes with their neighbors. In general, query expansion results in higher recall retrieval systems. In the past, random walks using WordNet [41] links have been applied for query expansion in Information Retrieval [30], and as a smoothing mechanism for the task of prepositional phrase attachement [100]. Somewhat similarly, automatic image captioning has been applied using graph walks, where textual descriptions of images are enriched with terms that are linked to related images [49, 81].

In the email domain, this means that rather than search for items explicitly and directly, it is possible to search by using associations. For example, one can submit a query that includes a person's first name (represented as a *term*) and retrieve related *email-address* nodes. (This problem is referred to as the *alias-finding problem* in Chapter x.)

Structural similarity. A main motivation for representing data as a graph and performing graph walks is to generate a notion of global, structural similarity. In general, structure is expressed by sub-clusters in the graph, such that similarity between points that have multiple common neighbors is reinforced as the graph walk proceeds. For example, in a citation network, which includes co-citation links, applying graph walks starting from a paper of interest should identify a cluster of papers, which are mutually closely related; in a social network, graph walks can reveal sub-communities, and so on. The extent to which similarity get concentrated in internal graph structures is dependent on the walk length, and on the graph walk parameters. It has been shown the length of the walk should be sufficient, but not inifinite (i.e., shorter than mixing time), to find clusters in data [96, 98]. The *Personalized PageRank* paradigm that is our main interest in this thesis keeps high association to the query nodes and decays fast thereafter; however, being a algorithm of the same family, it does reflect global structural similarity to some extent.

Node centrality. In settings such as the classical PageRank, graph walks converge to a stationary state indepednent of the starting point. Evaluating and ranking nodes by their centrality, or *prestige*, is desired for many applications [6, 39]. The framework which is in the focus of thesis, however, establishes an association to the starting point by assigning exponentially lower importance to longer connecting paths. Yet, the graph walk process is somewhat biased towards nodes that have high connectivity in the graph (as discussed in Section 3.2.2). We note that while bias towards central nodes may be a motivation to use this framework for some applications, in other settings – for example, the retrieval of relevant but low-frequency items – such bias may degrade the quality of the results (e.g., [14]).

Relation integration. The described framework is applicable to homogenous graphs. Yet, our main interest in this thesis is in the more general case, where the underlying graphs are heterogenous, i.e., include various types of local inter-entity relationships. In the email domain, for example, email *messages* are inter-connected through shared content (via the *has-term* relations), through social network information (via the relations *sent-from* and *sent-to*), and also through a timeline (*sent-on-date*). In an email-retrieval system this allows to reach email messages that are related by some of these dimensions. Furthermore, the graph walk naturally integrates such multiple evidence into a single score. Additional examples include lexical networks, which include various WordNet-based and other word-to-word relations [100, 30, 55], the heterogenous representation of image-text

and intra-similarity (as described above) and so on. **Add here: for email – can incorporate external information sources, such as organizational charts...**

Modularity. These types of inferences often exploit external knowledge sources (such as a stemmer, or WordNet)...

Context modeling. Representing data as a graph that includes multiple types of entities supports contextual retrieval. In an email application, for example, a user operates within a context which is explicit and readily available. In particular, the message browsed by a user is on display, a partial list of recipients in a composed message may be available, and so on. Together with a user-composed query, such information forms a contextual query, which is useful for favoring a relevant subset of the graph and in particular, for ambiguity resolution. An example task of person name disambiguation in email is formulated and evaluated in Chapter x.

Generality. Finally, a major advantage of represnting data as a network of entities and the relations between them is that many flavors of inter-entity similarity can be induced using graph walks. The underlying graph schema can be applied to carefully engineered graph schemas, constructed using information that is partial or pre-processed for a particular task. However, we argue in this thesis that graph walks can be applied as a *multi-purpose* tool, provided that the graph represents the data "naturally". Obviosuly, some relations observed in data may be of importance for a particular task, but irrlevant for others. A graph that includes a full set of the observed entity and relations can be used for various different tasks in the subject domain. In our study of the email problem, we will show that graph walks yield good performance for a variety of email-related problems, using default parameters. Further, this thesis makes the claim that given a base general representation, learning can be applied to specialize and adapt the graph walk to a particular flavor of node similarity of interest. Learning how to rank graph nodes in this framework is the subject of the next chapter.

Should scalability be discussed here, as a motivation? graph walks are more scalable than statistical relational learning... I guess that this will become more clear after writing the scalability section.

3.4 Related Work

Semi-supervised Learning Using Random Walks

There are multiple works in the area of machine learning addressing the problem of semi-supervised clustering using methods that directly apply or can be interpreted as ran-

dom graph walks. Given a graph in which some of the nodes are labeled, the link structure of the graph is exploited to infer the labels of the remaining unlabeled nodes.

Szummer and Jaakkola [96] have applied Markov diffusion processes in the settings of semi-supervised transductive classification, where labels are known for only a small number of the available data points. In their work, a local similarity metric is used that defines the distance between pairs of adjacent points. The underlying graph is undirected, and a node is connected to its k-nearest neighbors. (local distances are then expontiated and normalized to obtain transition probabilities.) A global similarity measure between arbitrary two points i and k is defined as the *diffusion probability* – the probability of transitioning from i to k in t time steps. They apply a *lazy graph walk* model, i.e. self-transitions back to each point are also included. The association of unlabelled points to the different labels is defined as the expectation over the known points label associations and the corresponding diffusion probabilities to the labelled points.

Using the matrix exponentiation idea, a class of kernels on graphs, *diffusion kernels*, has been proposed [63] for handling discrete structures, where the kernel captures both the local and global structure of the graph. Diffusion kernels can be regarded as a generalization of Gaussian kernels to graphs. It has been shown that diffusion kernels are the continuous time limit of *lazy random walks*. The diffusion kernel function is interpreted as a sum over paths from point i to point k, namely the sum of the probabilities that the lazy walk takes each path. Diffusion kernels are applicable to undirected graphs, as the kernel function must be symmetric.

Another approach to semi-supervised learning is based on a random field model defined on a weighted graph over the unlabeled and labeled data, where the weights are given in terms of a similarity function between instances [108]. In this framework, the known label assignments are fixed, and harmonic energy minimization is applied over a continuous state space to label the other instances. In terms of a random walk, it is interpreted as a mutli-step walk from an unlabelled node i, until a labelled node is hit. The labeled data is viewed as absorbing boundary for the random walk. The solution is an equilibrium state, expressed in terms of a hitting time. The resulting classification algorithms can be viewed as a form of nearest neighbor approach, where the nearest labeled examples are computed in terms of a random walk on the graph. **they also give an electrical network interpretation.** In order to improve classification accuracy, a mechanism for integrating class priors is provided, as well as integration of external classifiers' predictions.

A regularization framework that forces the classification function to change slowly on densely linked subgraphs has also recently been suggested for *directed* graphs [106].

Pan et-al [81] have applied the algorithm of Personalized PageRank – which they refer

to as *Random Walk with Restarts (RWR)* – to graphs that are undirected and unweighted, but include multiple types of nodes and multiple types of inter-object similarities. In particular, they study the problem of automatic image captioning. The graph constructed includes nodes representing *images*, as well as *terms* included in available captions and image *regions*' graphical representation. In the graph, nodes are either connected due to structural links (image to its graphical regions, and image to its caption terms is applicable), or due to high domain (graphical) similarity. RWR are applied in a cross-modal form: that is, key words are ranked for given images. Terms are also ranked by their relevancy to other given terms, etc.

Random Walks for Ranking

The framework described in this chapter is closely related to the algorithm of PageRank [79] and its variants. As PageRank scores are independent, by definition, of the initial distribution of the walk, researchers have suggested ways to bias the random surfer model to preserve an association between rankings and a query or user preferences. The *Intelligent Surfer* model [84] suggests that the surfer only follows links to pages whose content has been deemed relevant to the query. Similiarly, the distribution of PageRank's "jump" operation is skewed to include only pages relevant to the query. A similar approach has been suggested for search personalization [48]. In this model, named *Personalized PageRank*, the surfer is biased to reset his or her search uniformly over pages pre-categorized as relevant for a given topic. (In this model the transition probabilities are the same as in the original PageRank.)

The above works focused on web data. Zhou et-al [107] suggested to rank other types of data, such as images or text, with respect to the intrinsic global manifold structure [6, 7] collectively revealed by a huge amount of data. They suggest that for many real world data types this should be superior to a local method, which rank data simply by pairwise Euclidean distances. In their view, the ranking problem can be viewed as an extreme case of semi-supervised learning, in which only positive labeled points are available. In their framework, the graph is represented as a weighted symmetric **normalized** matrix, constructed using a local similarity metric. Positive scores are assigned to each query nodes. The query points then spread their score to their nearby neighbors via the weighted network. As in *Personalized PageRank* a fixed ratio of the propagated scores is re-assigned to the query nodes, and the process is repeated until a global stable state is reached. The authors show that this variant of random graph walks is similar to Personalized PageRank, where the initial ranking score of each query is weighted with respect to its degree. They show that the suggested graph walks equals to assembling all paths between two points, and weighting them by a decreasing factor.

Relational data.

The idea of representing structured data as a graph is widespread in the data mining community, which is mostly concerned with relational or semi-structured data. Proximity search in databases represented as graphs has been suggested by Goldman et-al [45]. They suggested inter-object search, i.e., ranking objects based on their distance to other objects. In their work, a database is represented as an undirected graph with weighted edges, and proximity is defined based on shortest paths between objects.

SimRank [56] is a similarity measure adapted for directed graphs, where nodes denote objects and edges denote the relations between them. In this model, objects are similar if they are related to similar objects. A base case is that objects are similar to themselves. Unlike PageRank and its variants, the SimRank measure computes scores for pairs of objects (the original graph is converted into a graph where vertices denote node pairs), by comparing their neighbors. overall, for a graph of size n, SimRank includes a set of n^2 similarity equations. An iterative calculation propagates scores one step forward along the direction of the edges, until scores converge. In terms of a random walk, the SimRank score for a pair of objects (a, b) is shown to measure how soon two random surfers are expected to meet at the same node if they started at nodes a and b and randomly walked the graph backwards.

BANKS [14] is a model suggested for keyword-based search in relational databases. In this framework, tuples are modelled as nodes in a graph, and they are connected by links induced by foreign key and other relationships. Inverse links are added to the graph schema. In response to a query, the returned ranked list of answers constitutes of small sub-trees connecting nodes that match the query terms. The authors suggest that such trees should help the user understand how the answer was reached, and allow him or her to further browse the database. The underlying graph includes typed and weighted edges, where weights reflect link importance. In addition, nodes are also assigned scores denoting prestige, that are defined as the node in-degree. Query answers are ranked using notions of proximity and node prestige.

In the *ObjectRank* model [7], random walks with restarts are applied to keyword search in databases modeled as labeled graphs. The underlying graph is a directed graph, where each node is typed and associated with a set of keywords (derived from the attribute values of the represented tuple). The graph edges are typed, and an *authority trasfer schema graph* determines the outgoing weight (autority transfer) per edge type. The assigned weighting schema may be deficient (where the total outgoing weight from a node according to the schema may be less than 1). The assigned authority transfer rates are distributed uniformly among the outgoing edges of the relevant edge type at each node of the graph. Given a query, RWR are applied, and graph nodes are ranked according to stationary state scores. The authors construct a *keyword-specific* ObjectRank score, where the reset op-

eration is limited to those nodes that include the keyword, and a *global* score, defined similarly to PageRank approach. The score of a node with respect to a keyword query is a combination of the global and the keyword-specific ObjectRank scores.

Spreading Activation?

Add somewhere (by ObjectRank):

The relevance between two nodes in a data graph can also be viewed as the resistance between them in the corresponding electrical network, where a resistor is added on each edge. This approach is equivalent to the random walk model (Doyle and Snell).

Statistical Relational Learning

Also add references from: Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation. It has refs abouts similarity measures by Faloutsos and Koren.

Use this somewhere: the semantic web, [5] A reference about XML: [**?**]

In ObjectRank [3], the Intelligent Surfer [20], and XRank [12], the random walk favors nodes containing query keywords in a fixed, arbitrary manner.

3.5 Discussion

Chapter 4

Learning

Given a graph, multiple types of tasks can be addressed as queries in this framework. We consider a *task* as a particular flavor of inter-node similarity in the graph. For instance, consider the domain of email, represented by the graph schema described in Figure 3.2(c). *Alias finding* is an example task in this domain. In this task, given a person's name, a user is interested in retrieving *email-address* nodes that are most associated (that is, belong) with that person. Another different task in this domain is *threading*, where given a particular *message*, a user may be interested in automatically recovering messages that immediately preceded (or followed) the focus message in the same thread. Yet another example is the *person name disambiguation* task, where a person's name is mentioned in an email message, and the goal is to associate the name mention to the relevant *person*. These and other generic tasks in the extended domain of personal information management are presented and evaluated in the next chapter.

While the same underlying graph can be used for various different tasks, it is reasonable that a particular subset of link types be relevant for extraction of semantic relations between nodes that are not directly linked in the graph. The importance of relations may vary and it is unlikely that a single set of parameter values Θ will be best for all tasks. Furthermore, the sequences of edge types (*paths*) that are traversed by the graph walk in reaching a target node also carry semantic meaning characteristic to the type of relationship (similarity type) between the nodes. It is thus important to consider the problem of *learning* how to better rank graph nodes for a given task.

This chapter discusses several approaches for learning to rank nodes in the given settings. The approaches include tuning of the edge weight parameters Θ. Several methods exist that learn the edge weight parameters to optimize graph performance [100, 35, 3]. We adapt an error backpropagation approach [35] to our settings. All of these weight

tuning methods, however, are local in the sense that they decompose the graph walk into discrete time steps. We present the concept of *global learning* to improve graph walks performance. In particular, we apply re-ranking in this framework using features that describe global properties of the graph walk, such as sequences of the edge types traversed. The approach of re-ranking is applied as post-processing, and can be combined with weight tuning. Finally, we also suggest a *path-constrained* graph walks variant. In the latter approach, random walkers 'rememeber' the paths traversed and select the next step probabilistically, using information based on full paths to the goal. That is, in this approach global information about the walk is 'injected' into the graph walk process.

In this chapter we will first define the learning problem and settings. We then give a detailed overview of a hill-climbing method that tunes the graph weights, as well as a re-ranking schema. ...

4.1 Learning Settings

In this thesis, we consider supervised learning settings. By problem definition, we are given a graph G and a set of initial graph edge weight parameters Θ^0. In addition, a set of labeled example queries is provided. For every example i ($1 \leq i \leq N$), an initial graph walk (using Θ^0, and a pre-set walk length k) generates a corresponding output ranked list l_i^0. Henceforth, z_{ij} will denote the output node ranked at rank j in a ranked list l_i, and $p_{z_{ij}}$ will denote the probability assigned to z_{ij} by the graph walk.

In general, example labels in the settings of learning to rank can be given in the form of absolute scores p'_{ij} [101], ordinal values assigned to nodes representing their relative relevancy to the example query [17], or pairwise node preferences sampled from the lists l_i^0 [3]. Throughout this chapter, we consider a binary labelling scheme, where sample nodes annotated as relevant (as well as irrelevant) to the query are provided. Such a labelling schema is particularly adequate for well defined problems, in which a query corresponds to a finite set of "correct answers" and the other nodes are considered irrelevant. In general, learning is aimed at improving the rankings l_i^0, such that nodes known to be relevant are ranked higher then the irrelevant nodes ($j_{rel} < j_{irrel}$) for every node pair in the output ranking l_i; that is, produce lists l_i, in which the relevant nodes occupy the top ranks, to the extent possible. As in the general case of learning, the goal in learning a model is to generalize and improve the rankings of unseen instances. **These instances may correspond to the same underlying graph on which training took place, or a different graph of the same domain, i.e., which adheres to the same graph schema.**

Finally, the example queries should all be instantiations of the query class, or *task*, of

interest. For example, a possible task is the generic query that finds relevant person nodes given a term node (as suggested in the running example of the previous section). Example queries in this case would include different term nodes as a starting point and potentially different relevant person sets.

4.2 Edge Weight Tuning: Error BackPropagation

Several methods have been developed that automatically tune the edge weight parameters, where edge weights are parameterized by their type (see Section 4.6). As a case study of these methods, we follow closely on an error backpropagation gradient descent algorithm [35], applying it to our schema of graph walks.

In this approach, a gradient descent algorithm is developed using the paradigm of error backpropagation in neural networks. The target cost function is defined as follows:

$$E = \frac{1}{N}\sum_{i \in N} e_z = \frac{1}{N}\sum_{i \in N} \frac{1}{2}(p_z - p_z^{Opt})^2$$

where e_z is the error for an (labelled) target node z, defined as the squared difference between the final score assigned to z by the graph walk p_z and some ideal score according to the example's labels, p_z^{Opt}. Specifically, p_z^{Opt} is set to 1 in case that the node z is known to be relevant or 0 otherwise. The error is averaged over a set of example queries of size N.[1] This cost function is to be minimized by gradient descent with respect to every edge weight $\theta_{\ell'}$, as follows:

$$\theta_{\ell'} = \theta_{\ell'} - \eta \frac{\partial E}{\partial \theta_{\ell'}} = \theta_{\ell'} - \eta \frac{1}{N}\sum_{i \in N} \frac{\partial e_i}{\partial \theta_{\ell'}}$$

The derivative of the error is a summation over each of the graph walk's time steps, where the final error is propagated backward, weighted by the relative contribution of every intermediate node to the final node score, as follows:

$$\frac{\partial e}{\partial \theta_{\ell'}} = (p_z - p_z^{opt}) \sum_{t=0}^{T-1} \sum_{y \in U_z(t+1)} P(y, t+1 \rightarrow z, T) \cdot \frac{\partial p_y(t+1)}{\partial \theta_{\ell'}}$$

where $U_z(t+1)$ denotes the set of graph nodes that are in the set of connecting paths leading to z, at time $t+1$; and, given that node y belongs to this set, $P(y, t+1 \rightarrow z, t)$ is

[1] In this case, N more precisely denotes the number of target nodes, where multiple nodes may be considered per query.

the total probability of reaching z at time T starting from y at time $t+1$. The derivative of the node y with respect to an edge weight $\theta_{\ell'}$ is:

$$\frac{\partial p_y(t+1)}{\partial \theta_{\ell'}} = \sum_{x \in pa(y)} p_x(t)\gamma \cdot \frac{\partial \frac{\sum_{\ell \in L_{xy}} \theta_\ell}{\sum_{y' \in ch(x)} \sum_{\ell' \in L_{xy'}} \theta_{\ell'}}}{\partial \theta_{\ell'}}$$

where $pa(y)$ is the set of nodes linked by an outgoing edge to y (its parents). Specifically, denoting as $C(\ell', L_{xy})$ the count of edge type ℓ' in the set of connecting paths L_{xy}, the explicit derivative is:

$$\sum_{x \in pa(y)} p_x(t)\gamma \cdot \frac{C(\ell', L_{xy})O_x - C(\ell', L_{xy'})\theta_\ell}{O_x^2}$$

where we use the abbreviation O_x for the total outgoing weight from node x, i.e. $O_x = \sum_{y' \in ch(x)} \sum_{\ell' \in L_{xy'}} \theta_{\ell'}$

Implementation Considerations

Since the target function is non-convex, it is possible that this gradient descent procedure result in local minima. Common techniques to overcome this pitfall include executing multiple trials, using different initialization parameters (Θ^0, here); simulated annealing, etc. Given the cost function and the gradient, it is possible to apply an optimization package such as LBFGS.

Obviously, the search process involves re-computing the ranked list (by executing the graph walk) for every iteration. The described weight tuning procedure may therefore be time consuming. However, relatively few example nodes were shown to give good performance. And, most importantly, once the set of weights is learned for a given task, it can be readily to new examples, simply by applying the learned edge weight paraemters Θ^* to the graph walk.

4.3 Re-ranking

An alternative approach for improving graph walk performance is learning to *re-order* an initial ranking. A reranking approach has been used in the past for meta-search [27] and also for several natural-language related tasks (e.g., [29, 28]). While typically the ranked list of candidates is generated using local search methods, reranking can incorporate features which represent global phenomena that was not captured in the local model.

Such information is often useful in discriminating between the top ranked candidates. For example, previous work [28] applied a MaxEnt learner to perform named entity tagging; then, reranked high-probability annotations using features describing the entity boundaries predicted.

Following is a short overview of the reranking approach, described in more detail elsewhere [29]. Each output node z_{ij} is represented through m features, which are computed by pre-defined feature functions $f_1, \ldots, f_m$. The *ranking function* for node z_{ij} is linear, defined as:

$$F(z_{ij}, \bar{\alpha}) = \alpha_0 log(p_{z_{ij}}) + \sum_{k=1}^{m} \alpha_k f_k(z_{ij})$$

where $\bar{\alpha}$ is a vector of real-valued parameters. Given a new test example, the output of the model is the output node list, re-ranked by $F(z_{ij}, \bar{\alpha})$.

There is a variety of algorithms suggested that learn the weight vector $\bar{\alpha}$ and are adapted for a ranking framework **examples**. In this thesis, we apply two methods: an average Perceptron algorithm [43], and a boosting algorithm that is adapted for ranking [29].

For completeness, following is a short description of the latter method. To learn the parameter weights $\bar{\alpha}$, the algorithm minimizes an exponential loss function on the training data:

$$ExpLoss(\bar{\alpha}) = \sum_{i} \sum_{j=2}^{l_i} e^{-(F(z_{i1}, \bar{\alpha}) - F(z_{ij}, \bar{\alpha}))}$$

where z_{i1} is, without loss of generality, a correct target node.[2] The weights for the function are learned with a boosting-like method, where in each iteration the feature f_k that has the most impact on the loss function is chosen, and α_k is modified. Provided that the features are binary, closed form formulas exist for calculating the optimal additive parameter updates [89].

Re-ranking Features

For improving graph walk performance, we suggest features that describe the output nodes in terms of the graph walk traversed to reach these nodes. These features are derived from the set of paths leading to every candidate node in the ranked list, including some non-local properties. In particular, we define the following three types of feature templates:

[2] If there are $k > 1$ target nodes in a ranking, ranking can be split into k examples.

- *Edge label n-grams* - features indicating whether a particular sequence of n edge labels $(n < T)$ occurred within the set of paths leading to the output node considered.

- *Top edge label n-grams* - these features are constructed similarly to the previous feature type. However, here the subset of top k paths that had the largest contribution to the final accumulated score of the output node is considered.

- *Source count* - In case that the initial distribution defined by the query included multiple nodes, this feature indicates the number of different source nodes in the set of connecting paths leading to the candidate node. This feature models the assumption that nodes reachable from multiple query source nodes are more relevant to the query.

In addition to this feature set, special additional feature sets that are suitable for different domains can be developed. In particular, features may include the types, or other properties, of the source or target nodes in addition to edge types. Specialized feature schemes that include extended features to the basic templates described were developed in the case studies described in the following chapters.

Maybe add an example of an instance and its description via these features?

Feature computation

A number of features describing the set of paths from the query distribution V_q can be conveniently computed in the process of executing the graph walk. Recall the definition of $Q(z|x)$ (Section 3.2). The same sort of recursive definition can be used to build up a feature vector that describes a ranked item z. First, a vector f of primitive feature functions that describe the individual edges in a graph is defined. (Such a feature function may include additional properties of a path segment, such as the source node type $\tau(x)$ etc.) We can define a weighted vector function F which aggregates the feature primitive functions over a walk that starts at node x and walks to node z in exactly d steps, as follows:

$$\begin{aligned} F(x \xrightarrow{=0} z) &= \mathbf{0} \\ F(x \xrightarrow{=d} z) &= \textstyle\sum_y \left(\Pr(x \xrightarrow{\ell} y) \cdot f(x \xrightarrow{\ell} y) \right) \cdot Q(y \xrightarrow{=d-1} z) \end{aligned}$$

Finally, we can define

$$F(z|x) = \gamma \sum_{d=1}^{d_{max}} (1-\gamma)^d F(x \xrightarrow{=d} z)$$

$F(z|x)$ can be computed throughout the execution of the graph walk, while computing $Q(z|x)$. In our implementation, this approximately doubles the cost of the computation. **memory-wise?**

Alternatively, rather than calculate the feature vectors for every node in the graph, it is possible to compute the feature vectors for the nodes at the top of the retrieved list, by recovering the set of connecting paths to the focus node. Recovering the set of connecting paths requires less memory, but additional computation time (about half the time of the actual walk?). Also, allows to design more sophisticated feature types?

Implementation Considerations

Unlike the weighted tuning approach, re-ranking puts some overhead over the graph walks. Namely, given new instances of queries, feature vectors need to be computed as part of query execution, before the model can be applied to re-rank them. (the cost of applying the model is negligible compared with feature computation.) A main concern is that while edge label *bigrams* correspond to a realtively small space, higher order *n-grams* may translate to a very large feature space. Given a limited number of training examples, this is likely to lead to over-fitting. In case high-order n-grams are incorporated, it is therefore required to apply techniques such as feature selection.

4.4 Path-Constrained Graph Walks

While node reranking allows the incorporation of high-level features that describe the traversed paths, it is desired to utilize such information already in the graph walk process. Assume that preliminary knowledge is available that indicates the probability of reaching a correct target node from the query distribution V_q, following distinct edge type sequences (*paths*) . Rather than have the graph transition probabilities be evaluated locally, based on a fixed set of edge weights Θ, the probability of following an edge of type ℓ from node x can then be evaluated dynamically, given the *history* of the walk up to x. That is, the edge weights Θ, are to depend on the random walker history. Performance-wise, assuming that indeed paths carry additional information compared with individual edges, this should be beneficial as paths that lead mostly to irrelevant nodes are likely to be degraded in the graph walk process. In addition, it is straight-forward to apply a threshold, to prune paths with low estimated probability of reaching a relevant node in the walk. This can yield scalability gains, while keeping performance at a high level.

This section describes a *path-constrained* graph walk variant, which implements these ideas. The algorithm includes two main components. First, it addresses the evaluation

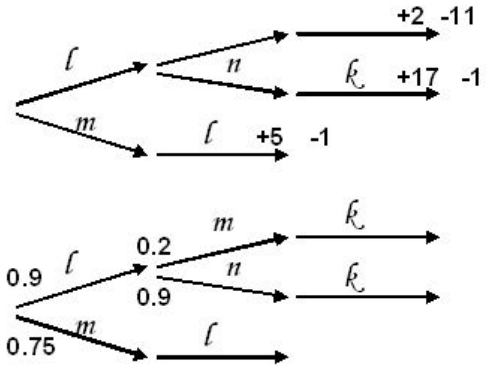


Figure 4.1: An example path-tree.

of dynamic edge weights, given the history of walk, based on training examples. The second important aspect of the algorithm is the adaptation of the graph walk to consider path history throughout the graph walk. This requirement also necessitates efficiency in handling the memory resources.

Our general approach is to model paths observed in a training dataset as a path tree, where path probability of reaching a relevant target node is deduced from the observed walks and node labels. Path probabilities are then propagated backward in the tree, to obtain estimates of the parameters Θ at each vertex of the path tree. While performing the graph walk, we will compactly represent walk histories by associating each node visited with the corresponding vertices of the path tree.

Next we describe in detail the constrcution process of the path tree, and the evaluation of its edge probabilities. We then describe a modified algorithm for the path-constrained graph walk.

Path-Tree Construction

We construct a path-tree T using the training set of N labeled queries. Let a *path* p denote a sequence of edge types up to (the maximal pre-defined) length k. For each training example, we recover all of the connecting paths leading to the top M correct and incorrect nodes.[3] Let C_p^+ be the count of path p within the paths leading to the correct nodes, over all examples N; and similarly, let C_p^- denote its count within paths leading to the negatively labelled nodes in the example set. The full set of paths observed can then represented as a tree. (The conversion to a tree is straight-forward, where identical path prefixes are merged.) The leaves of the tree, which correspond to the distinct paths,

[3]The connecting paths per a single node may include multiple instances of a path, as defined here, as the same edge sequence may be observed for intermediate nodes visited.

Given: graph G, path-tree T, query distribution V_0, number of steps K
Initialize: for each $x_i \in V_0$, assign a pair $< root(T), x_i >$
Repeat for steps $k = 0$ **to** K**:**
For each $< t_i, x_i > \in V_k$:
Let L be the set of outgoing edge labels from x_i, in G.
For each $l_m \in L$:
For each $x_j \in G$ s.t., $x_i \xrightarrow{l_m} x_j$, add $< t_j, x_j >$ to V_{k+1}, where $t_j \in T$, s.t. $t_i \xrightarrow{l_m} t_j$, with probability $Pr(x_i|V_k) \times Pr(l_m|t_i, T)$. (The latter probabilities should be normalized with respect to x_i.)
If t_i is a terminal node in T, emit x_i with probability $Pr(x_i|V_k) \times Pr(t_i|T)$.

Figure 4.2: Pseudo-code for path-constrained graph walk

are assigned a laplace-smoothed probability: $Pr(p) = \frac{C_p^+ + 1}{C_p^+ + C_p^- + 2}$. $Pr(p)$ is a (smoothed) maximum likelihood estimate of the probability of reaching a correct node following path p, based on the observed examples.

An example path tree is given in the top part of Figure 4.4. The path-tree in the figure includes three paths, constructed from the edge types k, l, m, n. The path $\xrightarrow{l}\xrightarrow{m}\xrightarrow{k}$, for instance, is shown to have occurred twice among the paths leading to nodes known to be relevant, and 11 times among paths leading to irrelevant nodes. The probability of reaching a relevant node following this path is therefore estimated at a low $3/15$.

Further, path probabilities are propagated backwards to the tree vertices, applying the MAX operator. Consider again the example in Figure 4.4. The bottom part of the figure gives the probabilities derived at each vertex. For example, after following an edge of type l from the root, the path-tree specifies that the probability of reaching a correct target node if an edge of type m is selected next is 0.2, whereas the respective probability if an edge of type m is followed is estimated at a higher 0.9. The probabilities associated with edge types not included in the path-tree in a particular vertex are considered to be zero.

Say something here about the reasoning behind the MAX operations.

A Path-tracking Graph-walk

Given a path-tree, we apply *path-constrained* graph walks that adhere both to the topology of the graph G, and to the path tree T. Walk histories of each node x visited in the walk are compactly represented as pairs $< t, x >$, where t denotes the relevant vertex in the path tree. This means that if x was reached via K different paths, as defined in this section, it would be represented using the K node pairs. Notice that the number of nodes

visited in the modified graph walk increases relative to an unconstrained walk. On the other hand, possible paths in the graph that are not represented in the path tree are readily pruned. It is also straight-forward to apply a threshold such that paths associated with a lower probability of reaching a relevant node in T are discarded. Such path pruning has a counter effect on the complexity of the walk.

A pseudo-code of the path-constrained graph walk is given in Figure 4.2. As detailed, the path-tree edge probabilities are treated as dynamic edge weights. These weights are normalized at each node (pair) traversed to generate edge probabilities, following the original graph walk paradigm.

Implementation Considerations

Want to have balanced examples...

What to sample from the top of the list ...

What if we don't know whether an example is positive of negative ...

A trade-off using a threshold ...

4.5 Method Comparison

In this chapter we presented three different approaches for learning to rank nodes given labelled examples, in the graph walk framework. We are now interested in inspecting the strengths and weaknesses of these methods, using some key criteria.

Global vs. local features. The graph walk process is strictly Markovian, where the random walker does not "remember" the history of the walk. That is, edge probabilities (or, edge weights) are fixed over time. Similarly, in learning the graph edge weights using the error backpropagation method, the graph walk is decomposed into single time steps, and optimization is performed "locally". This chapter established the notion of "global" learning in graph walks. The node re-ranking approach allows to exploit global properties of the walk, as it can represent information about the full paths traversed to reach a target node. For example, features can describe *edge sequences* traversed over multiple time steps. The path-constrained graph walks method embeds such high level information, considering the paths traversed, already into the graph walk.

Overall, re-ranking is probably the "most global" method, out of the approaches considered. In addition to edge sequences, re-ranking can incorporate features such as the *source count* feature, which corresponds to the collection of paths leading to a node. Similarly, re-ranking features can model the number of paths leading to a node, and other

global properties pertaining to a node connectivity. Finally, re-ranking can also model arbitrary domain-specific features, incorporating additional relevant information sources that are independent of the graph walk.

Applicability. The methods differ in their training requirements, the product of learning, and its application to unseen instances. The error backpropagation weight tuning approach may require long time to train, and careful tuning (in order to avoid local minima). However, this method yields a set of optimal edge weights adapted for a particular task. Given a query which is an instance of the same task, the learned set of edge weights parameters can be readily applied to the graph walk. For the path-constrained graph walk approach, training is a realtively simple procedure, which produces a path-tree. The path-tree is then employed in the modified graph walk, given a new query.

The re-ranking approach, on the other hand, requires that nodes be transformed into a feature vector representation. While learning a reranking function and applying the learned model to other feature vectors are efficient, the procedure of encoding nodes with their feature values adds processing overhead to query execution. This additional processing time is affected by the types of features used, and the fashion in which they are computed (see above). Nevertheless, this may be an important factor in real-world systems.

In addition, as noted before, in training discriminative methods such as re-ranking, a rule of thumb is that the larger the feature space, the larger is the training set required. This generally means that compared with re-ranking, we expect more examples to be required using the re-ranking approach. Otherwise, feature selection methods may be helpful.

Impact. Notice that re-ranking processes the top nodes produced by an initial graph walk. Performance of re-ranking is therefore limited, being dependent on the quality of the initially ranked lists. This also means that re-ranking can be applied in combination with weight tuning, or path-constrained walks, in a sequential fashion.

Weight tuning of course alters the results produced by the graph walk. Recall, however, that the graph walk generated rankings are affected also by other factors, involving the topology of the graph and properties of the graph walk paradigm (Section 3.2.2). For instance, an exponential decay over the transmitted probabilities is applied by the graph walk, diminishing the contribution of long walks. Nodes that are linked to nodes in the query distribution via short connecting paths are therefore likely to be assigned high probability scores and appear among the top ranks of the output node list.

The path-constrained graph walk variant can affect the output rankings to a large extent, as it allows the graph walk to follow only meaningful paths, pruning away other paths. Thus, nodes that are close to the query nodes, but are so related by an unmeaningful relation, for example, will be excluded for the ranked list. In practice, this approach can

alter the output of the graph walk to varying extents, depending on the applied threshold.

To summarize this discussion, we conclude that edge weight tuning is a theoretically justified (?) tool, as it is derived and applied directly to the graph walk. However, it is limited to considering local information of the graph walk. In addition, the graph edge weights affect node rankings together with other factors, and to that extent may have confined effect in some cases. Path-constrained walks consider global information concerning path relevancy, and incorpoate a path pruning mechanism, and can thus have a great impact on the quality of the output ranked list. Finally, re-ranking allows the consideration of features that describe nodes by various global properties of the graph walk. Re-ranking, however, operates on the top graph node rankings retrieved by the graph walk. It also involves a a feature encoding overhead in query proceesing. While node reranking can be used as an alternative to the other methods, it can readily be used as complementary approach, as the techniques can be naturally combined by first adapting the graph walk generated rankings, and then applying the re-ranking model. This hybrid approach has been used successfully in the past on tasks like parsing [29].

The effect of learning and the relative performance of these approaches (and their combination) will be evaluted empirically in the next chapters, where we review test cases for a couple of different domains.

4.6 Related Work

Learning Random Walks

Pagerank and Personalized Pagerank variants find the stationary distribution of a reasonable but arbitrary Markov walk over a network, and do not learn from relevance feedback. Several researchers have suggested to learn the link weights of the transition matrix, such that the authority scores assigned to nodes better reflect user preferences. Chang et-al [22] applied gradient ascent on the elements of the link matrix constructed by the related HITS algorithm [61], altering link weights to more closely align with the principal eigenvector of the documents that match user interests. They begin by running HITS to convergence using the original link matrix. They then derive a gradient of authoritative webpages with respect to the link matrix, and add a fraction of the gradient to each element of the link matrix. This operation not only increases the rank of a given node but also increases the rank of other similar documents. The algorithm, however, produced variable results.

It was later suggested to learn the teleport (reset) vector in the PageRank algorithm, to

affect node rankings [101]. User preferences considered were formed as either absolute node scores (where the initial walk scores were given to the user as reference), or as node pairwise preferences. The authors applied a quadratic programming approach to optimizing the teleport vector, where preferences were modelled as linear constraints.

Agarwal et-al [3] assume a similar setting, where a user has one or more hidden preferred communities that the learning algorithm must discover, and relevance feedback is given as node-pair preferences. Their goad is to tune the transition probabilities of the link matrix. In particular, teleport transitions are modelled as regular transitions using a dummy node, such that tuning of the teleport vector is included within the general framework. They present NetRank, an algorithm that seeks to optimize the transition matrix probabilities, such that the final node probabilities are similar in terms of KL divergence to the results of an initial flow, while trying to satisfy the given pairwise ranking preferences. NetRank does does not provide generalization guarantees, and indeed, does not generalize well in the experiments. In a later work [2], a theoretical justification has been given to this approach. it was shown that minimizing KL divergence between the learned and reference (standard PageRank) flows amounts to searching for a smooth scoring function. That is, it bounds the probability of the expected loss being very different from the empirical loss for the considered loss function.

Agarwal and Chakrabarti [2] draw a connection between learning to rank graph walks, where directed edges denote structural inter-entity relations (as in PageRank), and learning in *associative networks*, where undirected edges denote similarity and are weighted according to similarity strength. In particular, they analyze a Laplacian smoothing approach, applied in the latter settings [4]. The authors argue that in contrast to Laplacian smoothing, which assigns arbitrary scores to nodes, thus inducing all possible permutations, certain node orders in a given graph may be impossible to achieve via Pagerank. That is, the hypothesis space of Pagerank is contained in the hypothesis space of the Laplacian smoothing approach. Preliminary experiments indicate that this increased bias aids generalization. The authors also suggest an enhanced approach to learning in PageRank-like directed graphs, using additive margin and cost/rank-sensitive learning. They show that this approach compares favorably, as a general technique, to Laplacian-based smoothing.

Edge Weight Tuning.

Several methods have been developed that automatically tune edge weight parameters in extended PageRank models, where edge weighted are associated with the relation type that they represent. Earlier works, including XRank and ObjectRank, experimented with assigning different edge weights, but did so manually.

Nie et-al [77] have suggested PopRank, an object-level link analysis model that ranks

the objects within a specific domain, where relationships between objects are heterogenous. They apply a simulated annealing algorithm to explore the search space of all possible edge type weight combinations, with the goal of reducing the difference between partial rankings available from domain experts and the ranking produced by the learned model. In order to make the learning time manageable, they use a subgraph in the learning process, trading optimality for efficiency. The focus subgraph consists of a set of concentric circles with the training objects in the center as the core.

Toutanova et-al [100] have constructed a special graph including diverse word-to-word relationships, relating to WordNet relations, morphology and word features derived from dependency relations. They applied finite graph walks to induce smoothed word probabilities, using these probabilities for the task of predicting prepositional word attachement. In their work, the edge weight parameters of the model were fitted to optimize the conditional log-likelihood of the correct attachment sites for a development set of samples, including quadratic regularization **does this mean that they used only positive example?; also, they do not give details about deriving the gradient. check again**. Optimization was performed using a limited memory quasi-Newton method. The authors have also experimented with tuning separate edge weight parameters for different nodes in the graph, defining equivalence classes of states for which they tied the parameters. For example, parameters were binned based on the observed number of word occurrences. However, they report that the simplest model having a single equivalence class performed on average as well as the more complex models. **They use the same formula as in the BMC Informatics paper, and the main variant here.**

Agarwal et-al [3] have presented a hill-climbing approximation algorithm adapted for partial order preferences. They add given pairwise constraints as a violation penalty to the cost function. The derivative with respect to the weight of each edge type is computed by applying the chain rule, accompanying the regular Pagerank iterations with gradient finding steps. It is shown that scaling up the graph size, the time per iteration scales essentially linearly with the number of graph vertices and edges, and the number of iterations grows slowly with the size of the graph. Overall, the training time is mildly superlinear to the graph scale factor. The authors experiment also with a maxent flow setting, addressing the problem of general transition weight learning, where edge weights are not tied to edge weight parameters (see above). They find that in comparison, the approximate gradient-descent approach estimates relatively few global weights, and can therefore generalize from training to test instances that involve completely different nodes, far away in the graph, with a much smaller number of examples.

Global graph walks

Toutanova: a state is a pair consisting of a word and a single 'bit' taking on a value of

0 or 1. As we will shortly see, the extra memory bit allows our walk to 'remember' if the word in the current state is a head (0) or a dependent (1), and will permit us to build richer models. Other examples of Markov chains that can be thought of as random walks with an extra memory bit include (Lafferty and Zhai, 2001; Ng et al., 2001).

Re-ranking

An alternative approach for improving graph walk performance is learning to re-order an initial ranking. A reranking approach has been used in the past for meta-search [27] and also several natural-language related tasks (e.g., [29, 28]). Recently, it has been suggested to apply reranking to improve on graph walks in entity-relation networks [73, 25].

While node reranking can be used as an alternative to weight manipulation, it can readily be used as complementary approach, as the techniques can be naturally combined by first tuning the model parameters, and then reranking the result using a classifier which exploits non-local features. This hybrid approach has been used successfully in the past on tasks like parsing [29]. We follow this paradigm as well in our experiments.

Spreading Activation

Performing search via finite graph walks is closely related to *spreading activation* over semantic or association networks: there the underlying idea is to propagate "activation" from source nodes via weighted links through the network (e.g., [13, 86]). Spreading activation methods are parameterized by user-provided threshold functions for node activation, limits on node distance, preferences over paths, and other constraints. The framework of lazy graph walks discussed here is similar in character, but is less constrained; rather, it relies on learning to optimize how similarity "spreads" through the graph.

A tought: could be interesting to somehow compare between a rperesentation that includes relation types and the more studied problem of untyped but weighted edges. or, PageRank settings with no relation types... what are the gains?

Chapter 5

Case Study: Personal Information Management (PIM)

add this somewhere:

Results will be presented in terms of Mean Average Precision (MAP). We found that the error function and MAP are well-correlated.

In this chapter, we evaluate personal information management as a case case to the graph walk and learning derived similarity measures. We suggest representing personal information as a graph (extending the toy examples given earlier), and will evaluate a variety of related tasks. Some tasks have been studied before, and some tasks are novel.

There are several motivations for applying our framework to this domain.

Personal information, such as email and meeting entries implicitly represent social network information, textual content and a timeline. Obviously, there is a close relationship between these components of information. For example, persons on a user's contact list may be related by being part of one social "clique", as derived by a simple analysis of header information in an email corpus [54, 53]. Or, they can be related via common key words that appear in the relevant correspondece in the email corpus [69]. Such persons' relatedness is also tied to a time dimension. It is therefore desired to combine multiple relevance measures to utilize the multi-faceted information that is included in personal inforamtion source for related applications and tasks. Using graph walks, these multiple email-related aspects can be integrated.

Another motivation for using graph walks, is that the graph is modular, and can be easily extended to include various entity types. For example, we combine meeting entries

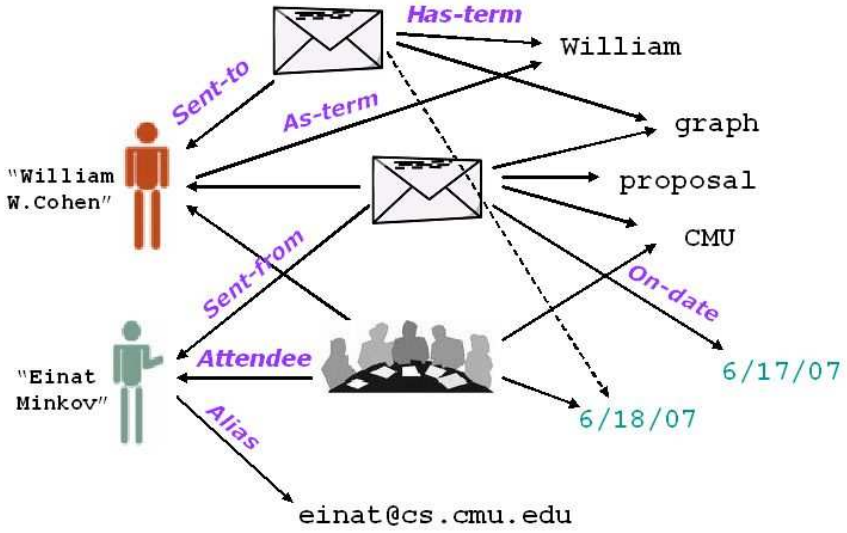


Figure 5.1: A joint graph representation of email and meetings data

in the graph. We also consider a notion of *activity*, represented by an email folder.

As shall be shown, since the graph representation is not reduced to task-specific features, we will use the same underlying graph to perform multiple different tasks. We also evaluate the extent to which learning can then further enhance the graph-walk generated similarity measure per the specific task of interest.

This chapter is organized as follows. We first discuss the representation of personal information as a graph. We then present the set of email tasks evaluated, and their corresponding representation as email queries. Then, experimental results are given for all tasks, where both base graph walks and the learning techniques are included, and performance is compared against relevant baselines. The chapter concludes with a discussion of related works, and a summary.

5.1 Email and Meetings Graph Representation

A graph example, including both email and meeting objects is given in Figure 5.1. The corresponding graph *schema* is detailed in Table 5.1. The graph representation naturally models an email corpus in the sense that it forms a direct layout of the information included within the corpus. The graph entities correspond to objects of types *messages* and *terms*, as

source type	**edge type**	**target type**
message	sent-from	*person*
	sent-from-email	*email-address*
	sent-to	*person*
	sent-to-email	*email-address*
	on-date	*date*
	has-subject-term	*term*
	has-term	*term*
meeting	attendee	*person*
	attendee-email	*email-address*
	on-date	*date*
	has-term	*term*
person	sent-from^{-1}	*message*
	sent-to^{-1}	*message*
	attendee^{-1}	*meeting*
	alias	*email-address*
	as-term	*term*
email-address	sent-to-email^{-1}	*message*
	sent-from-email^{-1}	*message*
	attendee-email^{-1}	*meeting*
	alias^{-1}	*person*
	is-email^{-1}	*term*
term	has-subject-term^{-1}	*message*
	has-term^{-1}	*message*
	is-email	*email-address*
	as-term^{-1}	*person*
date	on-date^{-1}	*message*

Table 5.1: Email and meetings node and relation types. (Inverse edge types are denoted by a superscript.)

well as *email addresses*, *persons* and *dates*. Directed graph edges represent relations like *sent-from*, and *on-date*. As shown, we distinguish between *terms* and *subject terms*, for the *has-term* relation. In the suggested schema, a person node is linked to its constituent token values with an "as-term" edge. Similarly, terms that are identified as an email-address are linked over an "is-email" edge type to the corresponding *email-address* node. In some of the experiments described in this chapter, we have added a "string similarity" edge type, linking email-addresses that demonstrate string similarity higher than a threshold. It is straightforward to add other information available, such as inter-personal organizational hierarchy relations if relevant, and so on.

Given a graph that includes email information, *meeting* objects can be easily incorporated to create a graph representing both email and meeting information. In particular, we assume that a given meeting includes attendees' information (names, or email-addresses), text describing the meeting (e.g.,"Webmaster mtg, 3305 NS") and a date. One can imagine a richer setting where meetings are also linked to richer text, files, web URLs, etc. Evidently, related email and meeting corpora have many entities in common: namely persons and email-addresses, terms and dates. It is therefore straightforward to join the two information sources. In the combined graph, a meeting will have a connecting path via term and date nodes to email files, for example. Relevant emails or other potentially included entities, like papers and presentations, can be therefore identified as background material for a meeting's participants in this framework. Similarly, the social network information embedded in emails may be enhanced with meeting information. That is, if meetings in the graph are linked to known attendees, these links may provide additional knowledge about persons' relationships, complementing the social network derived from email files.

Finally, we also suggest to embed a notion of *activity* in the graph. **ADD HERE!**

5.2 PIM Tasks as Queries

We claim that the suggested framework can be used as an ad-hoc contextual search platform, given email, meetings and other relevant information represented as a graph. The data included in the graph may describe *personal* information, in which case it can be used to serve one personal data search and consolidation needs; or, it may relate to *organizational-level* data, where cross-organizational infomation is available for retrieval and analysis.

The framework is general, and many query and search types are possible. In particular, contextual search allows for searching similar or related items to a set of objects of interest (e.g., "show persons names that are related to person P"); alternatively, it may assist a user in finding a specific item, given loose associative clues (e.g., "show those email-messages

task	V_q	τ_{out}
Person name disambiguation	*term* (name mention) *(+file)*	*person*
Threading	*message*	*message*
Finding meeting attendees	*meeting*	*email-address*
Finding email aliases	*term/s* (person's name)	*email-address*
Message foldering	*message*	*folder*
Message tracking	*folder*	*message*
Activity-person Prediction	*activity/folder*	*email-address*

Table 5.2: Query realizations of the considered tasks

that are related to 'Jenny', around 'March 1' ").

In this section, we will show that many email-related tasks, which have been treated separately in the literature, can be addressed uniformly as queries in the suggested framework. As defined in the previous chapter, a *task* is a *query class*, where independently of the query parameter values, a particular type of similarity or association between objects is sought. For example, in the task of *threading*, a user (human, or an automatic email processing agent) looks for messages that are adjacent to a given message in a thread. Given labeled examples, the available learning techniques can be applied for each task.

Following is a description of the tasks evaluated in our case study. Table 5.2 shows the corresponding query representation for each of these tasks.

Person Name Disambiguation. Consider an email message containing a common name like "Andrew". Ideally an intelligent automated mailer would, like the user, understand which person "Andrew" refers to, and would rapidly perform tasks like retrieving Andrew's preferred email address or home page. Resolving the referent of a person name is also an important complement to the ability to perform named entity extraction for tasks like social network analysis or studies of social interaction in email. However, while the referent of the name is usually unambiguous to the recipient of the email, it can be non-trivial for an automated system to find out which "Andrew" is indicated. Automatically determining that "Andrew" refers to "Andrew Y. Ng" and not "Andrew McCallum" (for instance) is especially difficult when an informal nickname is used, or when the mentioned person does not appear in the email header. This problem can be modeled as the following search query: given a name-mention in an email message m, retrieve a ranked list of *person* nodes. Assuming that the identity of the message which includes the person's name is available, one can phrase a contextual query, which includes both the name mention and the *message* node, adding valuable information for name disambiguation.

Threading. *Threading* is the problem of retrieving other messages in an email thread

given a single message from the thread. Threading is a well known task for email. As has been pointed out [66], users make inconsistent use of the "reply" mechanism, and there are frequent irregularities in the structural information that indicates threads; thus, thread discourse arguably should be captured using an intelligent approach. It has also been suggested [62] that once obtained, thread information can improve message categorization into topical folders.

As threads (and more generally, similar messages) are indicated by multiple types of relations including text, social network information, and timing information, we expect this task to benefit from the graph framework. We formulate threading as follows: given an email file as a query, produce a ranked list of related email files. We consider the immediate parent and child of the given file to be "correct" answers for learning.

Finding Meeting Attendees. Having meetings embedded in the graph, one can leverage the information included in both the email and meeting corpora to assist in meeting management. Specifically, we assume that a given meeting is associated with a text description. One can apply a search query starting from a meeting node, looking for relevant email addresses. A returned ranked list of such addresses can be utilized semi-automatically, assisting the user in the task of identifying relevant recipients to include in the meeting invitation or update notifications.

Finding Email Aliases. Consider the task of automatic assistance in finding a person's email-address. A typical email user oftentimes needs to retrieve email-addresses from his or her address book. In some cases, this is done by searching for an email with the desired information in the header. In the graph walk paradigm, this information can be retrieved by querying a person's name, searching for relevant email-addresses. The user may provide either a person's full name, as a set of terms, or the person's first or last name only. The latter setting may be faster and more convenient for an end user, and can be used also when a user is not certain about the full name.

Message foldering and tracking. Naturally, email, as well as other entities at the work station including meetings, files and directories, correspond to different facets of underlying user *activities*, which evolve over time. we consider the task of foldering email messages to existing user-created folders which denote an activity or a project, and vice versa. While not all folders pertain to a coherent activity (for example, a "sent-items" folder holds an eclectic collection of email messages), folders are often used to tag a collection of messages related by an underlying activity. For example, email folders often relate to distinct projects, or to recurrent activities (e.g., travel). We add *activity* nodes to the graph schema, which correspond to such folders. These nodes are linked to the email *messages* that are tagged with each folder.

Foldering has been studied in the past [90, 9, 52], with the goal of classifying an email message to a single relevant folder. We are interested in a scenario where a user may be interested in associating a message to *multiple* folders relevant. (Multi-tagging is supported, for example, of the popular *gmail* application.) For example, a user may be interested in tagging a message both with the relevant project folder and with a general "recruiting" folder. Unlike previous works, which classified email messages to the relevant folder, we approach this task as a *ranking* problem. Suggesting a ranked list of folders to the user supports multiple choice, where it is desired to have the most relevant folders placed at the top of the list. We evaluate the performance of the graph-walk framework for this task, and compare it to TF-IDF.

In addition to the foldering task, we consider the *inverse* problem, namely *message tracking*, which did not get previous attention. Consider a scenario where a user tags most messages with the relevant folder, but happens to skip some messages. Once this user is interested in retrieving a specific mistakenly untagged message, he will not be able to find it in the relevant folder. The task of folder-message ranking can be useful in such settings, as well as in the general case, where messages related to a particular activity are sought, while they may have been associated to other folders. We phrase this task as a query that specifies a *folder* of interest, where the entities sought are of type *message*.

Predicting person-activity involvement.

We consider a novel and ambitious task, where we seek to predict persons that are likely to get involved in an ongoing project activity, represented by a folder in an email corpus. While finding experts [8, 82] and recommending recipients [20] relies on evidence observed in the past, the prediction of future ivolvement of persons from the enterprise in an ongoing project may depend on the dynamics of the project and other factors that are unknown within the email corpus alone, and possibly hard to predict in general. Nevertheless, it is reasonable that some of the people that will get involved in a project can be predicted based on observed email correspondence. The task of person prediction, or recommendation, for an activity may be valuable to an organization, as it may promote early involvement of relevant individuals in a project. In the experiments conducted we are interested in evaluating the utility of person prediction for an ongoing activity, using authentic data drawn from the Enron corpus. The query, in this case, is phrased as a *folder* representing an ongoing activity, such as a group project, and the entities retrieved are of type *email-address*.

5.3 Experimental Corpora

We experimented with the following corpora.

The **management game** corpus contains email messages collected from a management course conducted at Carnegie Mellon University in 1997 [74]. In this course, MBA students, organized in teams of four to six members, ran simulated companies in different market scenarios. The corpus we used in our experiments includes the emails of all teams over a period of four days.

The **Enron** corpus is a collection of email from the Enron corpus that has been made available to the research community [62]. This corpus can be easily segmented by user: in the experiments, we used the saved email of several different users. To eliminate spam and news postings we removed email files sent from email addresses with suffix ".com" that are not Enron's; widely distributed email files sent from addresses such as "enron.announcement@enron.com"; emails sent to "all.employees@enron.com" etc. We also removed reply lines (quotes) from all messages, for the same reason.

The **Meetings** corpus contains a subset of William Cohen's email and meeting files. The email files were all drawn from a "meetings" folder, over a time span of about six months. In addition, we use all meeting entries (as maintained in a "Palm" calendar) for the same period. The information available for the meeting files is their accompanying descriptive notes as well as the meeting date. The meeting notes typically include one phrase or sentence – usually mentioning relevant person names, project name, meeting locations etc. The list of attendees per meeting was not included in the constructed graph.

Finally, the **Personal** corpus consists of email messages sent and received by the author.

The statistics of corpora size and their graph representations are detailed per experiment below. For all corpora, terms were Porter-stemmed and stop words were removed. The Enron corpora, and the Management game and *Personal* corpora are of moderate size—representative, we hope, of an ordinary user's collection of saved mail. The Meetings corpus is modest in size. In general, we believe that this framework should benefit from larger corpora that may be less sparse in text and have a richer link structure.

The processed Enron-derived corpora used in the experiments are available from the thesis author's home page. Unfortunately, due to privacy issues, the Management game and Meeting corpora can not be distributed.

5.4 Experiments and Results

There are currently no available annotated email corpora for evaluation of email-related queries. Thus, a key property of the evaluated tasks is that a non-subjective correct answer set can be constructed per query. This section describes the experiments conducted per the each of the tasks defined above.

For every task, we evaluate performance using graph walks with *uniform* edge weights Θ, i.e., $\theta_\ell = \theta_{\ell'}, \forall \ell$ (denoted by G:U), and also for graph walks where the edge weights have been tuned (G:L). In order to avoid local minima in learning the graph edge weights using the gradient procedure, we initiated the learning process from five randomly selected set of edge weights, and picked the weights which yielded the final best results on the training sets. Further, in all exepriments we applied reranking on top of the uniform-weighted graph walk results. For every example, the top 50 message nodes have been reranked (denoted as 'Rerank'), and both train and development set examples have been utilized in training the reranking model. Finally, for the path-contrained graph walk variant, path trees were learned using the train and development sets, where the positive and the top ranked negatively labelled nodes were considered. In general, the number of negative examples was limited to the number of positive example available.

In all of the experiments reported in this chapter we applied a stay probability $\gamma = 0.5$.

Statistical significance in comparing performance of the various methods was obtained using a two-sided Wilcoxon test, at significance level of 95%.

For every task, the specific experimental settings and datasets are presented. Results are given for graph walks and the various learning techniques, as well as for relevant baselines. We discuss the results and derive conclusions from each experiments regarding the framework.

5.4.1 Person Name Disambiguation

As described in Section 5.2, in the person name disambiguation task we are given a *term*, which is known to refer to a person's first name. The goal is then to retrieve a ranked list of entities of type τ =*person*, such that the relevant person appears at the top of the list.

Datasets

Unfortunately, building a corpus for evaluating the person name disambiguation task is non-trivial, because (if trivial cases are eliminated) determining a name's referent is often hard for a human other than the intended recipient. We evaluated this task using three

	corpus			dataset		
	files	nodes	edges	train	dev.	test
M.Game	821	6248	60316	20	25	61
Sager	1632	9753	112192	15	12	35
Shapiro	978	13174	169016	15	10	35

Table 5.3: Person disambiguation corpora and dataset details.

labeled datasets, as detailed in Table 5.3.

The Management game corpus has been manually annotated with personal names [74]. Additionally, with the corpus, there is a great deal of information available about the composition of the individual teams, the way the teams interact, and the full names of the team members. Based on this information, we manually labelled 106 cases in which single-token names were mentioned in the the body of a message but did not match any name from the header with the corresponding person node. In addition to names that refer to people that are simply not in the header, the names in this dataset include people that are in the email header, but cannot be matched because they are referred to differently than their formal names. Overall, the types of name mentions identified include:

- *initials*–this is commonly done in the sign-off to an email;
- *nicknames*, including common nicknames (e.g., “Dave” for “David”), uncommon nicknames (e.g., “Kai” for “Keiko”); and, American names that were adopted by persons with foreign-language names (e.g., “Jenny” for “Qing”).
- *other* – other name mentions labelled are regular first names, mentioned in the body of the email message, while not being included in the sender or recipient list.

For Enron, two datasets were generated automatically. The datasets correspond to corpora drawn for two Enron employees: Sager and Shapiro. For these corpora, we collected name mentions which correspond uniquely to names that are in the email “Cc” header line; then, to simulate a non-trivial matching task, we eliminated the collected person name from the email header. We also used a small dictionary of 16 common American nicknames to identify nicknames that mapped uniquely to full person names on the “Cc” header line.

Table 5.4 gives the distribution of name mention types for all datasets. For each dataset, some examples were picked randomly and set aside for training and development purposes (see Table 5.3).

	initials	nicknames	other
M.Game	11.3%	54.7%	34.0%
Sager-E	-	10.2%	89.8%
Shapiro-R	-	15.0%	85.0%

Table 5.4: Example person name type distribution per dataset.

Baseline: string similarity

To our knowledge, at the time this experiment was conducted, there were no previously reported experiments for this task on email data. (There were, however, a concurrent and subsequent works, which are included in the discussion of related research.) As a baseline, we applied a reasonably sophisticated string matching method [26]. Each name mention in question was matched against all of the person names in the corpus. The similarity score between the name term and a person name was calculated as the maximal Jaro similarity score [26] between the term and any single token of the personal name (ranging between 0 to 1). In addition, we incorporated a nickname dictionary,[1] such that if the name term is a known nickname of the person name, the similarity score of that pair is set to 1.

The results are given in Table 5.5, listing MAP and accuracy results. (*Accuracy* is defined as precision at the top rank.) In addition, Figure 5.2 shows the average recall at every rank down to rank 10. As can be seen, the baseline approach is substantially less effective for the Management game dataset. Recall that the Management game corpus includes many nicknames that have no literal resemblance to the person's name – these cases are not handled well by the string similarity approach. For the Enron datasets, string similarity perfoms very well since lexical similarity was used in automatically generating the dataset. In all the corpora, however, there are ambiguous instances, e.g., common names like "Dave" or "Andy". In these cases string matching match the name mentions with multiple people with equal strength. This results in lower recall at the top ranks.

Graph walks

We performed two variants of graph walk, corresponding to different methods of forming the query distribution V_q (Table 5.2). In the first variant, we concentrate all the probability in the query distribution on the name term. In the other graph walk variant, V_q is a uniform distribution including the name term and the relevant message node. In both cases, the length of the graph walks has been set to 2.

Using this *term* graph walk variant, the name term propagates its weight to the messages in which it appears. Then, weight is propagated to person nodes which co-occur with

[1]The same dictionary that was used for dataset generation.

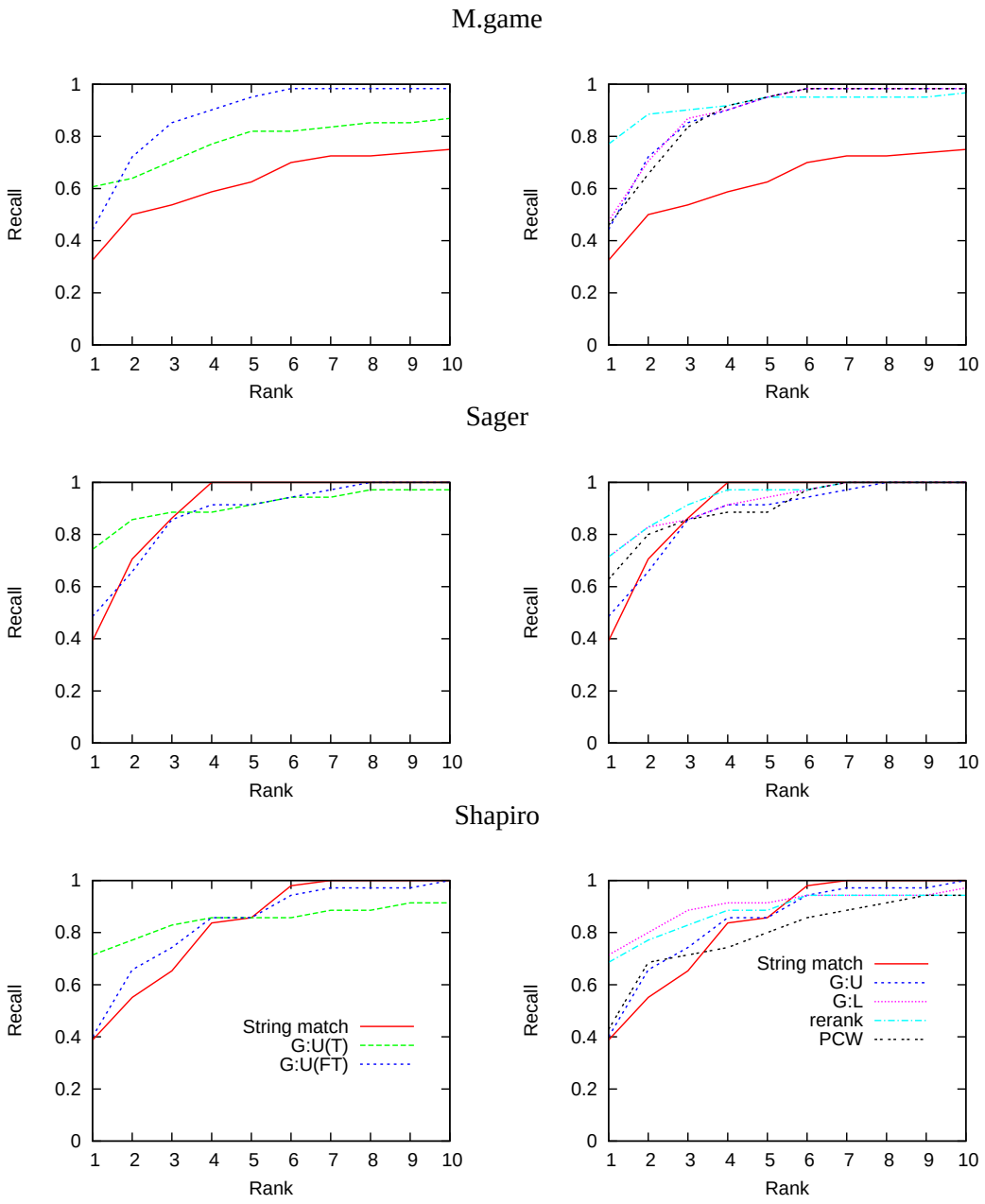


Figure 5.2: Person name disambiguation test results: Recall at the top 10 ranks, for baseline and plain graph walk, where the query includes a term only (denoted as G:U(T)), or term and file (denoted as G:U(T+F)) (left); and for all methods using contextual queries (T+F) (right).

	MAP		Accuracy	
	T	T+F	T	T+F
Cspace				
String sim.	0.49	-	0.33	-
G:U	0.68*	0.65*	0.53	0.44
G:L	0.61*	0.67*	0.46	0.48
G:PCW	0.64*	0.65*	0.53	0.46
G:reranked	**0.75**$^{*+}$	**0.85**$^{*+}$	**0.66**	**0.77**
Sager-E				
String sim.	0.68	-	0.39	-
G:U	0.83*	0.67	0.74	0.49
G:L	0.82*	0.81^{+}	0.74	**0.71**
G:PCW	0.81*	0.76^{+}	0.71	0.63
G:reranked	**0.87***	**0.82**$^{+}$	**0.80**	**0.71**
Shapiro-R				
String sim.	0.61	-	0.39	-
G:U	0.78*	0.61	**0.71**	0.40
G:L	**0.78***	**0.80**$^{*+}$	**0.71**	**0.71**
G:PCW	0.76*	0.62	0.69	0.43
G:reranked	0.76*	0.78^{*+}	0.69	0.69

Table 5.5: Person Name Disambiguation Results

these files. Note that in our graph scheme there is a direct path between terms to person names, so that person nodes may receive weight vis this path as well. The column labeled "T" in Table 5.5 gives the results of the graph walk from the *term* probability vector, and Figure 5.2 (left column, G:U(T)) shows recall at each rank, down to rank 10. As can be seen in the results, the graph walk performance is preferable. For example, the graph walk accuracy is 52.5% for the management game corpus, vs. 32.5% using the string matching approach. More drastic improvements in accuracy are observed for the Enron corpora. In terms of MAP, the graph walks are significantly better than string matching. However, this graph walk variant does not handle ambiguous terms as well as one would like, as the query does not include any information of the *context* in which the name occurred: the top-ranked answer for ambiguous name terms (e.g., "Dave") will always be the same person (where well-connected nodes get ranked higher).

We found that adding the file node to V_q provides useful context for ambiguous instances—e.g., the correct "David" would in general be ranked higher than other persons with this same name. Indeed, as shown in Figure 5.2 (left part, G:U(FT)), this contextual search

yields recall improvements compared to term-only queries, leading to nearly perfect recall at rank 10. On the other hand though, adding the file node results in the attribution of probability score to nodes that link to the file node, but not to the *term* node. In that sense, adding the file node to the query adds noise to the output ranking. This is reflected in the lower MAP and accuracy evaluation scores. This shortcoming

Learning

Weight tuning. We learned the graph edge weights using the error backpropagation method (denoted as G:L). In particular, learning was applied using 5 different initialization of the edge weight Θ, and the learned weights which performed best on the training set were selected. We applied learning to each corpus separately.

Edge weight learning results in a comparable performance to the *term*-query graph walks using uniform weights in all cases. Learning the weights significantly improves performance for the contextual search, for the two Enron corpora. A described earlier, the Enron datasets were created using a simpler automatic procedure. We conjecture that the difference in weight learning performance between the M.game and Enron corpora is due to the difference in name mention distributions (and consequently, to different connectivity patterns).

Reranking. For re-ranking, we applied the *edge bigram* and *source count* features described in Section 4.3. We also formed *string similarity* features, which indicate whether the query term is a nickname of the candidate person name retrieved (using the available small nicknames dictionary); and whether the Jaro similarity score between the term and the person name is above 0.8. This information is similar to that used by the baseline ranking system.

As shown in Table 5.5 and in Figure 5.2, reranking substantially improves performance, especially for the contextual walk. While the base graph walks yielded high recall, but incorporated noise at the top ranks for the contextual queries, the discriminative model learned allowed to rerank the nodes such that noisy nodes were demoted. In particular, high weights were assigned to the string similarity features and the *source count* feature. Both types of information assist in eliminating the "noise" due to the query file node in the contextual search.

Overall, as shown in Table 5.5, reranking gives the best results for two of the tree datasets, including the harder management game dataset. Reranking results are significantly better than the base graph walks with uniform weights in the contextual search settings, for all datasets. The right part of of Figure 5.2 shows reranking results, compared with the other methods, for the contextual search case.

Path constrained walks. Performance of the path contrained graph walk variant was

	corpus			dataset		
	files	nodes	edges	train	dev.	test
M.Game	821	6248	60316	20	25	80
Farmer	2642	14082	203086	22	23	93
Germany	2651	12730	158484	24	21	42

Table 5.6: Threading corpora and dataset details.

comparable to the base graph walks with uniform weights. An exception is the Sager dataset, for which the PCW method significantly improved results for the contextual search settings. We conjecture that rather than a small number of predictive paths, the person disambiguation problem is characterized by a combination of "weak" noisy paths (i.e., paths that lead to both correct and incorrect answers at high rates). Unlike reranking, string similarity or the high-level information considered by the *source-count* feature could not be modeled in the path constrained walk approach.

5.4.2 Threading

As discussed in Section 5.2, in the thread recovery task, we are interested in retrieving *messages* that are adjacent to a subject *message* in a thread (i.e., either a 'parent' or immediately consecutive messages). We consider this task as a proxi to the more general task, of fiding generally related messages.

Datasets

We created three datasets for the evaluation of the threading task, using the Management game and two Enron corpora. (here we use the messages extracted for two other Enron employees, Farmer and Germany). Statistics about the corpora and the constructed datasets are given in Table 5.6. For each relevant message, its parent was identified by using the subject line and time stamp. About 10-20% of the messages have both parent and child messages available, otherwise only one file in the thread is a correct answer.

We used several versions of this data, in which we varied the amount of message information that is available. More specifically, we distinguish between the following information types: the email *header*,including sender, recipients and date; the *body*, i.e., the textual content of an email, excluding any quoted reply lines or attachements from previous messages; *reply lines*, i.e., quoted lines from previous messages; and *the subject*, i.e., the content of the subject line. We compared several combinations of these components, in which information is gradually eliminated. First, we included all of the information

header	√	√	√	√	√	√	√	√
body	√	√	√	-	√	√	√	-
subject	√	√	-	-	√	√	-	-
reply lines	√	-	-	-	√	-	-	-
	MAP				**Accuracy**			
Cspace								
TF-IDF	0.55	0.49	0.37	0.42	0.44	0.34	0.22	0.17
G: U	0.59	0.53	0.36	0.36	0.46	0.35	0.20	0.22
G: L	0.68^{*+}	0.59	0.44^{+}	0.43^{+}	0.59	0.47	0.31	0.37
G: PCW	0.75^{*+}	0.73^{*+}	0.52^{*+}	0.45^{+}	0.67	**0.62**	0.39	**0.41**
G: Reranked	$\mathbf{0.77}^{*+}$	$\mathbf{0.73}^{*+}$	$\mathbf{0.59}^{*+}$	$\mathbf{0.51}^{*+}$	**0.68**	**0.62**	**0.44**	0.34
Germany-C								
TF-IDF	-	0.53	0.36	0.23	-	0.34	0.22	0.07
G: U	-	0.55	0.49	0.44^{*}	-	0.39	0.34	0.27
G: L	-	0.55	0.51^{*}	0.44^{*}	-	0.39	0.37	0.27
G: PCW	-	0.65^{+}	0.53^{*}	0.45^{*}	-	0.46	0.37	0.22
G: Reranked	-	$\mathbf{0.72}^{*+}$	$\mathbf{0.65}^{*+}$	$\mathbf{0.64}^{*+}$	-	**0.56**	**0.51**	**0.51**
Farmer-D								
TF-IDF	-	0.69	0.36	0.32	-	0.55	0.25	0.13
G: U	-	0.65	0.53^{*}	0.50^{*}	-	0.48	0.40	0.41
G: L	-	0.72^{+}	0.57^{*+}	0.50^{*}	-	0.61	0.46	0.41
G: PCW	-	0.76^{+}	0.63^{*+}	0.52^{*+}	-	0.66	0.54	0.45
G: Reranked	-	$\mathbf{0.83}^{*+}$	$\mathbf{0.65}^{*+}$	$\mathbf{0.61}^{*+}$	-	**0.70**	**0.56**	**0.52**

Table 5.7: Threading Results: MAP

available in the graph representation. We then removed reply lines if applicable, and eliminated further subject line information; finally, we removed the content of the messages. Of particular interest is the task which considers header and body information alone (without reply lines and subject lines), since it excludes thread-specific clues, and can therefore be viewed as a proxi for the more general task of finding related messages.

Baseline: TF-IDF

As a baseline approach we applied a vector space model, in which a message is represented as a TF-IDF weighted vector of terms, and inter-message similarity score is defined as the cosine similarity of their vectors. All information, including a message header, was included in the vector representation as terms.

The particular TF-IDF weighting scheme used is the following:

$$w_{i,j} = tf_{ij} \cdot idf_i = tf_{ij} \cdot log_2(\frac{N}{df_i})$$

where N is the total number of files, df_i is the count of messages in which of term i appears, and tf_{ij} is the count of term i mentions in message j.

The results, detailed in terms of MAP and accuracy (Table 5.7), show that this approach performs reasonably well. As one might expect, removing information, in particular the subject and reply lines, degrades performance substantially.

Graph walks

To formulate this as a problem in the graph model, we let V_q assign probability 1 to the *file* node corresponding to the original message, and let $\tau_{out} = file$ (see Table 5.2). Graph walks of length 2 were applied.

The results show that the graph walk using uniform weights (G:U) and the TF-IDF method give comparable performance when identical chunks of text, such as subject lines, are present in both the query message and the "target". However, the graph walk performs significantly better in the case that only header and body text information are available, improving MAP by 91% and 56% for the Germany and Farmer corpora, respectively.

Learning

Weight tuning. Learning the graph edge weights results in (often significantly) improved performance, across corpora, as shown in Table 5.7 (G:L).

Reranking. We applied reranking using *edge bigram* features. Overall, reranking the graph walk output yields the best results of the considered methods. In all cases, the results of graph walk with reranking are significantly better than the TFIDF baseline, and the graph walks with uniform weights. The MAP for the setting in which the least information is available, namely header information only, is higher than 0.5 across corpora with reranking.

Most features that were assigned high weight by the learner were edge type bigrams corresponding to paths such as:

- message $\overset{sent-from}{\longrightarrow}$ person $\overset{sent-to^{-1}}{\longrightarrow}$ message
- message $\overset{has-term/has-subj-term}{\longrightarrow}$ term $\overset{has-term/has-subj-term^{-1}}{\longrightarrow}$ message
- message $\overset{on-date}{\longrightarrow}$ date $\overset{on-date^{-1}}{\longrightarrow}$ message

These paths are indeed characteristic of a thread: e.g., the sender of a message is likely to be a recipient of a reply message, there is high temporal proximity between messages in a thread, and some textual overlap.

Note that while such sequences of relations can be readily identified as important in the graph framework, they cannot be modeled in a flat representation such as the vector space model. Sequential processes exist also for other email-related tasks, e.g., workflows and social interaction [18].

Path constrained walks. Finally, the path constrained graph walk variant give second-best overall results for this problem. We found high correlation between the edge sequences considered significant by the reranking models and the path probabilities of the path trees constructed.

5.4.3 Meeting Attendees Prediction

Given a *meeting* description, which links to textual notes, a date and possibly a partial list of attendees, the task of attendee prediction is to rank the *person*, or *email-address* nodes in the graph by their relevancy to the meeting (Section 5.2). While this is a *recommendation* task, we evaluate it as a prediction task, where links from meeting to attendees have been removed, and we are interested in recovering these links.

Datasets

For evaluation of the meeting attendee prediction task we use the *Meetings* corpus that contains a subset of William Cohen's email and meeting files. The email files were all drawn from a "meetings" folder, over a time span of about six months. In addition, we use all meeting entries (as maintained in a "Palm" calendar) for the same period. The information available for the meeting files is their accompanying descriptive notes as well as the meeting date. The meeting notes typically include one phrase or sentence – usually mentioning relevant person names, project name, meeting locations etc. The list of attendees per each meeting is not available, and is not included in the constructed graph.

The corpus statistics are given in Table 5.8. The first line of the table ('email') gives the number of email messages, and the size of their respective representing graph. The second line of the table ('meetings') refers to meeting entries statistics. The size of the respective graph refers to *additional* graph nodes and edges, given that the email information is already represented in the graph.

The experimental dataset consists of labeled examples of meetings for which the list of the email addresses of relevant attendees is given (manually annotated by the corpus

	corpus			dataset		
	files	nodes	edges	train	dev.	test
email	346	3239	27366	5	0	6
meetings	334	441	2074			

Table 5.8: Meeting attendee prediction corpus and dataset details.

owner). The examples for the time slice of which this corpus was derived are often similar to each other, given that many meetings are periodic. In order to avoid bias towards specific repetitive examples, the constructed dataset includes only 11 examples, manually selected as having distinct attendee lists[2]. The number of relevant meeting attendees varies – for some examples that represent personal or small meetings there are only few relevant email-addresses identified, while for larger project meetings there are dozens of relevant email-address nodes. For all examples, all attendees are considered to be equally relevant. Overall, 195 email-addresses are known in the corpus.

We notice that mapping email-addresses to meetings is not trivial since in many cases, there are multiple email-addresses referring to a single person. Some addresses may be rarely used or obsolete. In the experiments conducted, we consider *all* email-addresses that are associated with the attendees as correct answers. In results reported elsewhere, evaluation procedures where the *first* email-address retrieved per attendee was considered, or all email-addresses considered per user, gave similar results [72].

Overall, the experimental corpus and dataset are modest in size. We believe that this framework should benefit from larger corpora that may be less sparse in text and have a richer link structure. Nevertheless, despite its size, this experimental corpus is an interesting testbed for the suggested application.

Baseline: String similarity

To the best of our knowledge, the suggested task is novel and there are no previous suggested methods in this settings. As a baseline, we use a string matching approach. Since many of the message notes include persons and project names, string matching can utilize the similarity between persons name or public project names and relevant personal or project-related email-addresses. We use the Jaro-Winkler measure [26] to compute string similarity. The similarity score for every email-address is considered as the maximum Jaro-Winkler score of that email-address against any one of the words appearing in a meeting notes. The result of the described procedure is a ranked list of email-addresses,

[2]We also required that the meetings relate to persons that are likely to appear in the email corpus, as opposed to random visitors.

	MAP	Accuracy
Meetings		
String sim.	0.24	0.33
G:U	0.58*	0.67
G:L	0.65*	0.67
G:PCW	**0.68***	**1.00**
G:reranked	0.59*	0.67

Table 5.9: Meeting attendees finding results

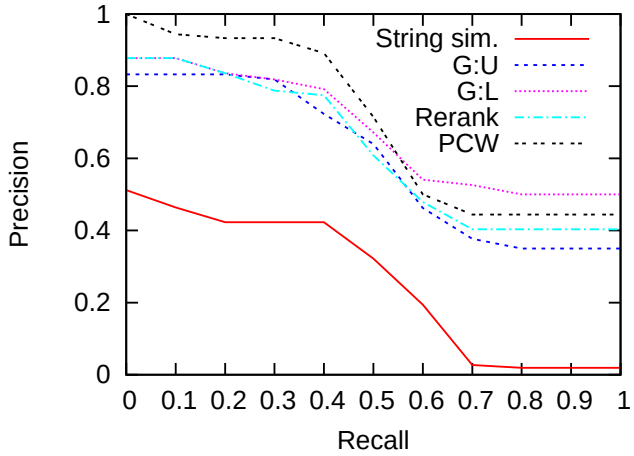


Figure 5.3: Meeting attendee prediction results: Precision-recall curve

given the meeting notes.

The results of applying the string matching approach are given in Table 5.9 in terms of MAP and accuracy, and in Figure 5.3. Since the number of correct answers varies between examples we use an 11-point precision-recall curve averaged over all examples for evaluation. As the given meeting notes often include explicit mentions of persons names, string matching reaches some of the relevant email-addresses. This approach fails, however, in many cases where the meeting text is more general, referring to (formal or informal) project names. In such cases, string matching can not map the given terms to individual persons' email-addresses. In addition, string matching does not allow the retrieval of email-addresses that are not similar to person names mentioned.

Graph walks

We perform a 3-step graph walk. As shown in the results, the graph walk performance

	corpora			dataset		
	files	nodes	edges	train	dev.	test
personal	810	11136	113224	9	8	26
meetings	346	3239	27366	8	-	6

Table 5.10: Alias finding corpus and dataset details.

is significantly preferable to string matching. Unlike string matching, the graph walk can retrieve email-addresses that have no literal ressemblence to a person's name, using co-occurence mappings. In particular, a 3-step walk uses paths such as:

- meeting $\overset{has-term}{\longrightarrow}$ term $\overset{as-term^{-1}}{\longrightarrow}$ person $\overset{alias}{\longrightarrow}$ email-address
- meeting $\overset{has-term}{\longrightarrow}$ term $\overset{has-term^{-1}/has-subj-term^{-1}}{\longrightarrow}$ file $\overset{sent-to/from-email}{\longrightarrow}$ email-address
- meeting $\overset{on-date}{\longrightarrow}$ date $\overset{on-date^{-1}}{\longrightarrow}$ file $\overset{sent-to/from-email}{\longrightarrow}$ email-address

In addition, a graph walk would give higher weight to frequently-used email-addresses over rarely used ones. This is a desired property in this case.

Learning

As shown in Table 5.9 and Figure 5.3, learning the graph edge weights gives preferable performance to the all-purpose uniform-weighted graph walks. Statistical significance could not be obtained due to the small number of examples. Reranking gave similar results to the initial graph walk. We conjecture that learning a reranking model could benefit from a larger training set. Finally, the path-constrained graph walks gave the best performance and perfect accuracy, where a relevant email-address has been identified at the top rank for every meeting description.

5.4.4 Alias Finding

The task of alias finding is defined as the retrieval of all *email-addresses* pertaining to an individual (or a mailing-group). The query may consists of a *person* node, or the corresponding name *term* (Section 5.2). In the experiments conducted, we consider the latter settings.

Datasets

	MAP	Accuracy
Meetings		
String sim.	0.55	0.67
G:U	0.61	**0.83**
G:L	0.55	0.67
G:reranked	0.59	**0.83**
G:PCW	**0.68**	**0.83**
Personal		
String sim.	0.54	0.69
G:U	0.72	0.77
G:L	0.73	0.77
G:reranked	0.63	0.85
G:PCW	**0.74**	**0.96**

Table 5.11: Alias Finding Results

We evaluate the task of alias finding using two corpora, as detailed in Table 5.10. For both corpora, we use a manually labelled list of email-address aliases per person. All of the examples considered refer to individual users (versus mailing lists) that have two to five email-addresses. In the experiments, we require the full set of email-addresses to be retrieved given the person's name. Elsewhere, settings in which the query included the person's full name represented as terms proved to be an easier problem [72]. Querying by the person's first name only, however, may be faster and more convenient for an end user and can be used also when a user is not certain about the full name.

Baseline: String matching

As a baseline, we use here the string matching approach described earlier (Section 5.4.3). The results of applying string matching are given in Table 5.11 in terms of MAP and accuracy, and in Figure 5.4, as an 11-point precision-recall curve. String matching is successful in this case in identifying email-addresses that are similar to the person's first name. There are, however, email-addresses that are similar to a last name only, or to that are not similar to neither the person's first or last name. Such instances bound the recall of this approach.

Graph walks

We apply a 3-step walk. In addition to the previously described edge types, we add here to the graph schema links that denote string similarity between *email-address* nodes. Specifically, email-address pairs for which string similarity is higher than a threshold are

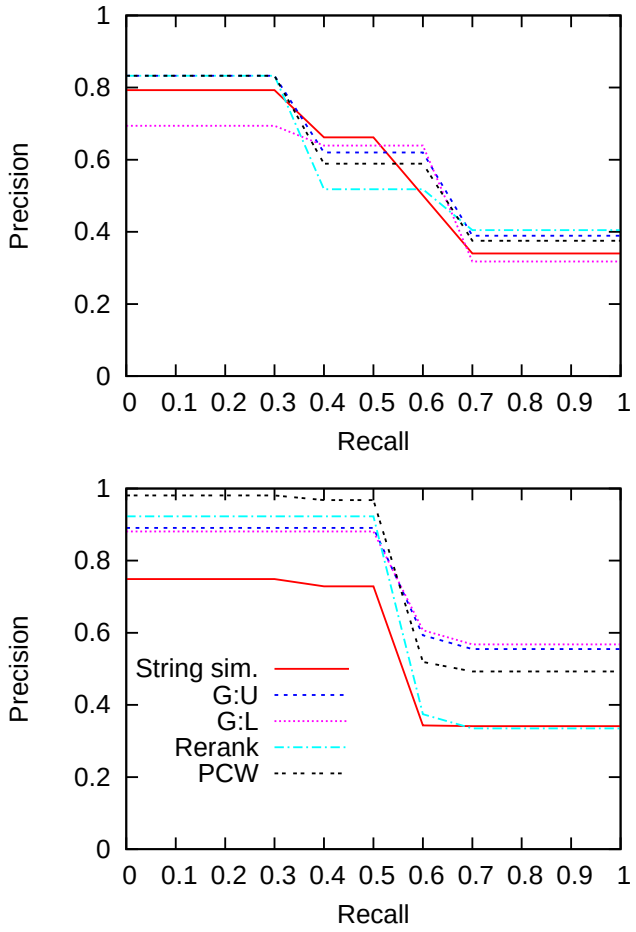


Figure 5.4: Person to email-address mapping: Precision-recall curve

linked over two *string similarity* symmetrical directed edges. In general, graph walks are expected to be effective in realizing co-occurence information and retrieving highly used email-address nodes. However, rarely used email-addresses may be harder to find using graph walks. Incorporating string matching into the graph links should thus increase graph walk recall.

As shown, the performance of the graph walk is better than string matching for both corpora. It results in MAP of 0.61 and 0.72 for the Meetings and 'Personal' corpora respectively, compared with 0.55 and 0.54 using string matching. Some relevant paths in a 3-step walk are as follows:

- term $\overset{as-term^{-1}}{\longrightarrow}$ person $\overset{alias}{\longrightarrow}$ email-address
- term $\overset{has-term^{-1}/has-subj-term^{-1}}{\longrightarrow}$ file $\overset{sent-to/from-email}{\longrightarrow}$ email-address
- term $\overset{has-term^{-1}/has-subj-term^{-1}}{\longrightarrow}$ file $\overset{sent-to/from}{\longrightarrow}$ person $\overset{alias}{\longrightarrow}$ email-address

In addition, similarity edges can be added as a "tail" to the previous paths. That is, once the graph walk reaches an email address node, the next step propagates some probability mass to similar email-address nodes over "similar-string" edges.

Learning

As shown, learning the graph edge weights resulted in comparable performance to the graph walks with uniform weights. While particular edge sequences are meaningful for the alias finding task, weight tuning only uses local information. We conjecture that this limits performance in this case, also given well adapted edge weight parameters.

For reranking, we used *edge bigram* features in the reported experiments. Reranking performance was comparable to the initial graph walks. Using edge-trigram features (and adding feature selection to avoid overfitting in this larger feature space) may yield better performance.

Finally, the path constrained walks gave the best results for both datasets. This is probably due to modeling information about the full paths traversed.

The differences between the methods were not found statistically significant.

5.4.5 Predicting person-activity future involvement

In the person-activity prediction task we are given a *folder* that is associated with a project activity as a query. The entities retrieved are of type *email-address*. We assume that the email messages in the folder provide textual and social network evidence, which allows the prediction of persons likely to get involved in the project in the future. As described earlier, we consider a snapshot of an email corpus at a particular point in time. Predictions are evaluated based on the email traffic that took place later in time.

Datasets

We evalute this task using the saved email of four different users. Each of the individual users mailbox was truncated up to a particular point in time (individual to each user, adapted to their individual periods of activity). The mailboxes include foldering information, as created by the users. For each folder, up to 80 most recent messages are

	corpora					dataset		
	date	files	nodes	edges	#persons	folder	#known	#targets
Kaminski V.	*Feb 1, 01*	1193	10005	102984	611	London	111	19
Beck S.	*Oct 1, 00*	1334	12944	174886	635	Europe	144	33
Kitchen L.	*Sep 1, 01*	1065	11762	149274	552	Portland	106	25
						East-power	156	14
						Mexico	49	9
Farmer D.	*Jul 1, 00*	741	7354	64556	336	Ces	55	13
						Wellhead	38	9

Table 5.12: Activity-person prediction corpora and dataset details.

Folder	Cosine	DP	G:U	G:L	WPT
London	0	0	**1**	-	-
Europe	0	0	**4**	**4**	3
Portland	7	7	3	3	**9**
East-power	**1**	0	0	0	0
Mexico	0	0	1	1	**2**
CES	0	0	1	**2**	0
Wellhead	**2**	0	0	0	0

Table 5.13: Person-activity prediction results: Recall at 20

maintained. (This allows both efficiency in maintaing email history and also keeps the corpus up-to-date.) That is, for each user we consider a snapshot of his or her mailbox, where history is limited. The relevant corpora statistics for the four users are presented in Table 5.12.

The dataset consists of seven folders, drawn from the described corpora, that are associated with project activity. Table 5.12 details for each folder the number of persons that are already associated with the messages in the folder by the corpus snapshot date ('known'). In the experiments, a query is defined as the node representing the subject *folder*, and all of the entities of type *email-address* (the size of this set is detailed in the column named 'persons' in the table) are ranked. Only addresses which are have not appeared in the subject folder prior to the snapshot time stamp are considered valid answers.

Baseline

A different approach previously suggested in the personal information management domain [76] is to model the various entities using word distributions. In this approach, the word distribution assigned to an email-address, for example, is the sum of the word

distributions for each message sent by, and received by that email-address. We represent *folder*, or *activity*, entities in the same fashion, that is, as a summation of the word distributions corresponding to the messages that they are known to be related to. Inter-entity similarity can then be estimated by the dot product or cosine similarity between the pairs of word distribution vectors. We apply the vector space model, where we match the TF-IDF weighted word distribution representing the folder with the distributions that describe each of the relevant email-address entities.

Results are shown in Table 5.13. It is reasonable that a user be presented by the recommending system with a relatively short ranked set of email addresses (or names). We therefore evaluate performance in terms of recall at the top ranks, considering the top 20 rated nodes. Recall is measured by the number of email-addresses that indeed appeared in the folder later in time. (The number of correct answers per folder is specified in Table 5.12 as 'targets'.)

Given the TF-IDF weighted representations, we ranked email-addresses by both cosine and dot-product (DP) similarity. Cosine similarity gave preferable results, where at least one correct email-address was predicted for three of the eight folders. Dot-product similarity performance was inferior, resulting in positive recall at the top 20 ranks only for one of the folders.

Graph walks

We applied graph walks of $k = 4$ steps for this task. As shown in Table 5.13, the graph walk method gives better predictions for five out of the eight folders compared with cosine similarity, which is perferable for the three remaining folders.

Learning

The *London* folder was used for training the learning models. Learning the graph edge weights led to improved recall at rank 20 for one of the test folders (namely, for the the *'Ces'* folder, increasing the number of predicted email-addresses from one to two).

We did not apply reranking in this case, as there was only little training data available. Path-constrained walks, however, led to improvements for two of the folders and somewhat degraded performance for two other folders. Overall, the path contrained walks performance was better than the cosine similarity baseline. Highly predictive paths in the constructed path-tree included:

- folder $\overset{in-folder^{-1}}{\longrightarrow}$ message $\overset{sent-from/to-email}{\longrightarrow}$ email-address $\overset{sent-to-email^{-1}}{\longrightarrow}$ message $\overset{sent-to-email}{\longrightarrow}$ email-address
- folder $\overset{in-folder^{-1}}{\longrightarrow}$ message $\overset{has-term}{\longrightarrow}$ term $\overset{has-term^{-1}}{\longrightarrow}$ message $\overset{sent-to-email}{\longrightarrow}$ email-address

	corpora				dataset		
	files	folders	nodes	edges	folder	files	examples
Kaminski V.	859	33	8925	81728	Conferences	80	10
					London	80	10
					Resumes	80	10
					Satnford	27	8
Beck S.	1131	89	12149	146746	Congratulations	28	5
					Recruiting	61	10
					Europe	80	10
Kitchen L.	1085	32	11758	142432	HR	80	10
					East-power	80	10
Farmer D.	635	16	6741	52968	Wellhead	80	10

Table 5.14: Message foldering and tracking: corpora and dataset details.

That is, noth social network information ans textual evidence were found informative for this task. While the task of prediction future involvement of persons from the enterprise in an ongoing activity is challenging, the results indicate that nevertheless, several correct predictions are likely to be included in short lists introduced to users. The success ratios are much higher than random chance. Consider also that our form of evaluation is strict, and it is possible that email-addresses (persons) predicted, who have not in fact appeared in the folder later in time, may be informative.

In addition to quantifiable performance, a potential advantage of the graph walk methods is that they can provide an explanation about the project-person relationship in the corpus. That is, the primary paths leading from a folder to a person, including the traversed relation types, can be presented to the user. An explanation mechanism should be useful in motivating recommendations to a user who is reasonably familiar with the corpus.

5.4.6 Message foldering and tracking

Given an email *message*, we are interested in ranking existing user-created *folders* by their relevancy to to the message. We also consider the inverse problem: ranking untagged *messages* by their similarity to a given *folder*. The folders considered denote user *activities*, where other eclectic folders, such as “sent-items” folder, etc. are ignored. A more elaborate description of these tasks is given earlier in this chapter (Section 5.2).

Datasets

We consider the same corpora constructed for the task of activity-person prediction (Section 5.4.5). These corpora consist of the saved email of four Enron employees, drawn from the Enron corpus. For the evaluation of the message foldering and tracking tasks, we removed folders that were evaluated as eclectic, i.e., non-topical. The modified corpora statistics for the four users are presented in Table 5.14.

Overall, the dataset includes 10 different folders per the 4 users. For each of these folders, we consider the consequent "future" messages in each folder (right past the corpus snapshot date). In the graph representation, the test messages are linked to all of the entities in the graph (persons, email-addresses, terms etc.) that it relates to. These messages, however, are not linked to their corresponding folder. Since in the Enron email corpora a message is attached to a single folder, the actual folder assigned to a message is considered as a single correct answer to each query in the foldering task. The inverse problem is addressed in a similar fashion. In this case the folder forms the query, and all of the test messages pertaining to the folder are the correct answers. Table 5.14 details the folder names and the number of test messages per each folder. The number of candidate folders to be ranked in the foldering task is detailed in the 'folders' column.

Baseline: TFIDF

As a baseline, we generate inter-entity similarity scores in the vector space, similarly to the baseline for the person prediction task described in the previous section. In this paradigm, a folder is represented as a TF-IDF weighted vector, which is the average of the vector representations of all of the messages that it is known to be associated with. Folders are ranked for each test message, and all messages are ranked for a given folder (for the foldering task and the message tracking task, respectively) using cosine as well as dot-product similarity measures.

The results of the foldering task are presented in Table 5.15 in terms of mean average precision. As shown, performance using cosine similarity gives very good results for all folders. Dot-product similarity (which is identical to the cosine similarity measure, except for message length normalization) gives reasonable performance, but not as good as the cosine measure.

The results of the inverse folder-message tracking task are presented in Table 5.16. For this task, the cosine similarity measure results in surprisingly good performance, considering the large number of candidate messages being ranked. In this case, the performance of the dot-product similarity is far behind its cosine counterpart. This suggests that lengthy messages are ranked higher with the dot-product measure, whereas length normalization in the cosine measure accounts for this factor.

Graph walks

Folder	Cosine	DP	G:U	G:L	PCW
Conferences	**0.79**	0.41	0.54	-	-
London	**0.95**	0.90	0.93	-	-
Resumes	**0.85**	0.72	0.62	-	-
Stanford	**1.00**	0.94	0.81	-	-
Congratulations	0.47	0.27	**0.53**	**0.53**	0.36
Recruiting	**0.88**	0.73	0.83	0.82	0.84
Europe	**1.00**	0.55	0.90	**1.00**	0.95
HR	**0.58**	0.17	0.53	0.57	0.45
East-power	**0.95**	0.50	0.77	0.67	0.79
Wellhead	0.76	0.69	0.95	**1.00**	**1.00**

Table 5.15: Message foldering results: MAP

We applied graph walks of length $k = 4$ for both tasks.

As shown, the performance of graph walks for the foldering task (Table 5.15) is comparable or somewhat worse than cosine similarity in most cases. Graph walks give better performance than cosine for two of the folders. The graph walk results are superior to dot-product similarity.

In the case of message tracking, however, the results are quite different (Table 5.16). The graph walk performance is quite poor compared with the cosine similarity measure. Interestingly, the graph walk results are comparable to dot-product similarity for most examples.

One explanation for the major difference in performance of the graph walk between the two tasks is as follows. Lets assume that the folder "recruting" is characterized by frequent use of the term "recruit" in the messages linked to it. In the *foldering* task, the graph walk propagates probability mass from the query node, representing a message, to the terms that appear in the message within one walk step; that is, the message is represented as a bag of terms. In the next steps of the walk, probability mass is conveyed from the terms to other messages, and then to folder nodes. Should the original message include the term "recruit", this term node (which is assigned the same weight as other terms included in that message) will convey most of its allocated proability to files that are associated with the folder "recruiting". Other, less predictive terms, on the other hand, will distribute their probability to files that are associated with various folders, resulting in smaller contributions per folder. For the inverse task, however, the graph walk behavior is different. In this case, the graph walk starts with a *folder* node. Probability is then propagated to the messages linked to the folder. In the next graph walk step, each (uniformly weighted)

Folder	Cosine	DP	G:U	G:L	PCW
Conferences	**0.34**	0.15	0.04	-	-
London	**0.34**	0.08	0.01	-	-
Resumes	**0.35**	0.08	0.32	-	-
Stanford	**0.99**	0.12	0.13	-	-
Congratulations	**0.37**	0.02	0.06	0.05	0.06
Recruiting	**0.50**	0.09	0.11	0.11	0.43
Europe	**0.27**	0.05	0.04	0.05	0.25
HR	**0.26**	0.06	0.03	0.03	0.07
East-power	**0.97**	0.43	0.12	0.12	0.34
Wellhead	**0.60**	0.09	0.08	0.09	0.27

Table 5.16: Message tracking results: MAP

message node, distributes its probability to the terms it contains. This means that *common*, or central, terms that are contained by many messages are likely to be assigned high weights. For the task of message tracking, a bias towards node centrality is not desired, as it results in high rankings of messages that include common terms. This suggests that beyond stop word elimination, which was already performed in constructing the graph, assigning continuous *weights* to edges, downweighting some terms, may be beneficial.

Another similar phenomenon concerns the file length normalization, which is included in the TFIDF cosine measure, but not in the graph walk. In the graph framewrok, long messages contain a larger bag of words. This means that even if a message contains frequent words, that propagate small fractions of their probability to many message nodes in the graph walk, the sum of these probability contribution may be significant in the case of long messages. In other words, this phenomenon also suggests that the graph walk bias toward central, i.e., well-connected nodes in the graph, is not suitable for this task.

In general, an adapted edge weighting scheme that contradicts the bias toward those nodes that are central in the graph (e.g., using a TF-IDF weighting schema) may improve performance. We leave this to future work. Another conclusion that rises from the results is that graph walks may give good performance for classification purposes (in the foldering task, folders can be seen as classes in a semi-supervised classification settings). The inverse task, however, may result in poor performance due the inherent perference for discriminative nodes rather than central nodes in this case.

Learning

For learning, the *Kaminski* corpus, including four folders, was allocated for training

purposes. Testing took place using the remaining corpora.

Learning the graph edge weights resulted in an overall improvement of the results for both foldering and message tracking. The improvements, however, are relatively minor in both cases. We conjecture that graph topology dominates the graph weights, especially for the task of message tracking.

We did not apply reranking to this problem. We do believe that reranking can be helpful, especially if specialized features are used; for example, features that encode the IDF value of traversed *terms*, or the length of the ranked messages. This is left for future work.

Finally, path-constrained walks resulted in major improvements for the message tracking task. (although, not reaching the level of performance achieved using cosine similarity). Paths in the path-tree that were associated with high positive probability to reach a target node included:

- folder $\overset{in-folder^{-1}}{\longrightarrow}$ message $\overset{sent-to-email/sent-from-email}{\longrightarrow}$ email-address $\overset{sent-from-email^{-1}}{\longrightarrow}$ message
- folder $\overset{in-folder^{-1}}{\longrightarrow}$ message $\overset{has-term}{\longrightarrow}$ term $\overset{as-term^{-1}}{\longrightarrow}$ person $\overset{sent-from^{-1}}{\longrightarrow}$ message
- folder $\overset{in-folder^{-1}}{\longrightarrow}$ message $\overset{sent-to-email}{\longrightarrow}$ email-address $\overset{alias^{-1}}{\longrightarrow}$ person $\overset{sent-from^{-1}}{\longrightarrow}$ message
- folder $\overset{in-folder^{-1}}{\longrightarrow}$ message $\overset{has-term}{\longrightarrow}$ term $\overset{has-term^{-1}}{\longrightarrow}$ message

How does this align with the previous explanation - bias towards central nodes, etc., if paths with terms are important?

5.5 Related Work

This section first discusses works that are related to personal information processing in general. It then proceeds to reviewing previous work relevant for the each task studied.

There has been an increased interest in the applying machine learning techniques, or artificial intelligence in general, to the area of personal information management in recent years. In particular, a variety of email-related tasks have been studied with the goals of facilitating email management and utilizing the information that resides in email corpora.

Example tasks include email foldering [9], automatic finding of experts at the enterpise using email resources (e.g., [8, 82]), recommendation of recipients for a given message (e.g., [20, 80]), identifying possible email leakage to wrong recipients [19], and more.

Naturally, email, as well as other entities at the work station such as documents, calenders and information organization, correspond to different facets of underlying user *activities*. Research has been conducted concerning activity-centred collaboration [44]. Methods have been developed with the goal of identifying threads of activities based on the contents of emails and documents that people are working on [64, 18]. The problem of classifying email messages into activities has been studied as well [36]. Mitchell et-al [76] have suggested a framework for the automatic extraction of user activities, based on the user's email, calendar, and the entire workstation content accessible via Google Desktop Search. In the work of Belloti et-al [11, 12], a user interface is suggested that is adapted to activity management. Their goal is to support common activities such as organizing a meeting, planning a trip or conducting a performance review, as well as other user-defined activities, side by side with existing email functions. The suggested interface, named Activity-Centered Task Assistant (ACTA), was designed to create an efficient personal information management environment and provide context metadata for machine learning and automation techniques. For example, it is desired that relevant emails, people, and email addresses be suggested to a user when viewing a meeting related to a particular activity on their calendar. The activity-centered contextual search problems described in this chapter can naturally complement frameworks such as ACTA.

Unlike most previous works, the graph walk framework addresses the structural aspect of personal information, where meta-data is represented explicitly, as well as inter-entity relations. There are only few previous works in the literature that integrate meta-data and text in email. One example examines clustering using multiple types of interactions in co-occurence data [10]. Another paper [1] proposes a graph-based approach for email classification. They represent an individual email message as a structured graph, including both content and header, and find a graph profile for each folder; incoming messages are classified into folders using graph matching techniques.

Another advantage of the graph walk paradigm compared with other approaches, is that it addresses various tasks similarly. That is, the same underlying graph, interface and query language are used for multiple tasks. Previous works, however, treated individual tasks separately, adapting data representation and using different methods per task.

In what follows we describe earlier works for each of the tasks included in our test study.

Person name disambiguation. The particular task of person name disambiguation has

been studied in the field of social networks and applied also to email data (e.g., [68, 34]). Diehl et-al have suggested to perform name disambiguation in email using traffic information, as derived from the email headers [34]. In their approach, a candidate set is first generated, including network references with identical names to the name mentions, for which at least one email communication has been observed with the sender. They then suggest a scoring formula based on the counts of message exchange between each candidate and the sender, or between each candidate and all of the message recipients, summarizing over different ranges of history. Our approach mainly differs in that it allows integration of email content and a timeline in addition to social network information in a unified framework. In addition, rather than evaluate a pre-filtered set of candidates, we use the graph walk to rank all of the graph nodes using network topology.

Recently, Elsayed et-al [37] suggested a generative model for resolving name mentions in email. They annotate the corpus with person names, mapping them to their email-addresses and nicknames (as deduced from email salutations and signatures). In their experiments, persons that have a first name or nickname that exactly matches the name mention are considered as candidates for matching. The model suggested can be thought of as a language model over a set of personal references. A contextual space of a name mention m is defined as a mixture of 4 types of contexts: the email message; all messages in the thread; discussions that some or all of the message participants (sender and receivers) joined or initiated at around the date of the considered message; and messages that are topically similar to the message that took place at around the same time. Each name mention is then resolved based on the corpus' person-email-nickname mappings. Overall, the described approach is highly specific to the subject problem, and does not employ learning. In addition, incorporating other types of evidence such as lexical similarity or meeting objects requires manual adaptation of the model.

Threading. Email thread structure can be used to improve email search, summarization, classification and visualization. Thread information was also used in the chaining of speech acts [18]. Thread structure recovery is therefore still under active research.

Lewis and Knowles [66] considered email threading as a retrieval problem. They applied text matching methods to the textual portions of messages. They suggest a strategy of using the quotation of a message as a query and matching it against the unquoted part of a target message. Yeh et-al [105] extend this approach. They suggest using string similarity metrics and a heuristic algorithm to reassemble threads in the absence of header information. In addition to message content similarity they consider heuristics, such as subject, timestamp, and sender/recipient relationships between two messages. They also introduce a time window constraint to reduce the search scope in the corpus. Unlike these works, the graph walk framework relies on the graph structure, and does not rely on man-

ually encoded world knowledge. Instead, learning allows to adapt the general graph walks to the special characteristics of the threading task.

Finding meeting attendees. We are not aware of previous works exploring the task of finding a set of relevant meeting attendees, in planning or updating a meeting. Previous research focused mainly on automatic *meeting scheduling* (e.g., [91, 75]). Our work facilitates semi-automatic construction of a meeting attendees' list, which is a preliminary step to meeting scheduling. An *author-recipient-topic* (ART) generative model has been recently suggested [69] for clustering persons by their inter-similarity, assuming a joint model of email recipients and topic. This approach may be adapted to predict relevant persons given text. Another recent work [76] uses desktop search to create a bag-of-words representation of email messages, and also of persons and meetings. According to this method, cosine or another similarity measure between the bag-of-words representation of a meeting and a person could be used to identify relevant meeting attendees. One difference between their approach and ours is that we consider the data structure in evaluating entity relationships; that is, we can tune the importance of particular connecting paths in this framework. In addition, in order to achieve high performance, vector-based similarity measures require sufficient data. This may be an obstacle given short meeting descriptions.

Alias finding. The task of finding a person's set of email-addresses in an email corpus given the person's *name* is novel as well. This task is related, however, to the task of identifying email aliases (given an email-address) in a corpus. Previous works explored the information residing in social network co-occurrences for this task, which resulted in performance better than random [53], and attempted to combine social network information and string similarity measures for this task [54]. Our approach allows integration of header information and string similarity measures, as well as email content and time in a unified framework. In addition, the above works did not consider inter-entity relations, which according to our results are beneficial for this task.

Message foldering and tracking. The foldering task [52] has been considered in the past. Mainly, it was phrased as a classification problem, with the goal of classifying an email message to a single relevant folder. Previous works have shown good performance using algorithms such as TF-IDF [90] as well as Naive Bayes, MaxEnt and SVM [9].

A graph-based approach for email classification has been proposed previously [1]. They represent an individual email message as a structured graph, including both content and header, and find a graph profile for each folder; incoming messages are classified into folders using graph matching techniques.

We are not aware of previous works that considered the inverse task, of message foldering.

Activity-person prediction. There has been much interest recently in the task of expert finding given email corpora and other person-document associations. Researchers have employed a language modeling approach for this problem [8, 82]. While content is important to the prediction of person involvement in a project, our results show that social network information is more crucial for this task. Another similar task studied is finding relevant recipients for a given message or a meeting invitation. It has been shown recently that a simple K-nearest-neighbors approach gives good performance for this problem [20]. In the settings considered, however, the 'query' pretains to a single message, and partial information about the recipients is available. In the task defined as predicting future persons involvement in an activity, the available information is a whole folder. As was the case for other tasks described thus far, the main difference between other works and our framework is that we do not pre-process the data and adapt a model for a specific problem. In this testcase we compare against cosine similarity of TF-IDF vector representations, which is another prominent vector-space model.

5.6 Summary

Compared with other works, this framework can not address tasks that require fine text processing such as language models. Also, as seen in the message tracking task, text has the problem of central nodes, that the model is biased to.

Future work: Finally, having time information in the graph allows for incorporating a time notion in finding relevant persons. It is straight-forward for example to add edges between date nodes according to time proximity, thus modeling a timeline as required additional graph walk steps. This is not used here, but might be beneficial for richer corpora.

... this last section is to be finished tomorrow ...

Chapter 6

Applications of Parsed Text

While text is often processed used as bag-of-words (this was the case in the PIM representation schema discussed in the previous chapter), it is desirable, and often beneficial, to utilize the information residing in the syntactic structure binding words in text processing tasks. In this chapter, we suggest to represent text as an entity-relation graph that includes sentences and their underlying dependency structure. Previous works have applied graph walks to draw a notion of semantic similarity over graphs that were carefully designed and manually tuned, using WordNet reations [100, 30, 55]. However, while these and other researchers have used lexicons such as WordNet to evaluate similarity between words, there has been much interest in extracting such a measure directly from text corpora (e.g., [93, 78]). We suggest processing dependency parse trees within the general framework of directed labelled graphs. We construct a graph that directly represents a corpus of structured (parsed) text. In the suggested graph scheme, nodes denote words and weighted edges represent the dependency relations between them. We apply graph walks to derive an inter-word similarity (relatedness) measure. We further apply the set of learning techniques available to improve the derived corpus-based similarity measure.

The graph representation and the set of graph walk similarity measures are empirically evaluated on the task of *coordinate term* extraction, from small to moderately sized corpora, where we compare them against vector-based models, including a state-of-the-art syntactic distributional similarity method [78]. It is shown that the graph walk based approach gives preferable results for the smaller datasets (and comparable otherwise), where learning yields significant gains in performance. We also present results for the extracting word synonyms from a corpus, which are consistent with these findings.

CORRECT THIS:

Below we first outline our proposed scheme for representing a dependency-parsed text corpus as a graph, and provide some intuitions about the derived similarity metric. We then discuss the representation of the evaluated tasks as search queries in this framework. We next present the empricial evaluation settings and results for the coordinate term extraction task **and for the extraction of general word synonyms.** The chapter concludes with review of related work, and a summary.

6.1 Parsed Text as a Graph

In recent years, there has been increasing interest in using dependency parses for a range of NLP tasks, including machine translation, relation extraction and question answering (e.g., [93, 102]). Such applications benefit particularly from having access to dependencies between words, since these provide information about predicate-argument structure that is not readily available from phrase structure parses. At the same time, dependency parsers of higher quality and speed are becoming available.

A typed dependency parse tree consists of directed links between words, where dependencies are labelled with the relevant grammatical relation (e.g., *subject*, *indirect object* etc). We suggest representing a text corpus as a connected graph of dependency structures, according to the scheme shown in Figure 6.1. The graph shown in the figure includes the dependency analysis of two sentences: "boys like playing with all kinds of cars", and "girls like playing with dolls". In the graph, each word mention is represented as a node, which includes the index of the sentence in which it appears (marked as circles in the figure). The "type" of each word – henceforth a *term* node – is denoted by a square in the figure. Each word mention is linked to the corresponding term; for example, the nodes "like$_1$" and "like$_2$" represent distinct word mentions and both nodes are linked to the *term* "like". For every edge in the graph, we add another edge in the opposite direction (not shown in the figure); for example, an link exists from "like$_1$" to "girls$_1$" with an edge labelled as "nsubj-inv". The resulting graph is highly interconnected and cyclic.

We will apply graph walks to derive an extended measure of similarity, or relatedness, between word *terms* (as defined above). For example, starting from the term "girls", we will reach the semantically related term "boys" via the following two paths:

girls $\stackrel{mention}{\longrightarrow}$ girls1 $\stackrel{nsubj}{\longrightarrow}$ like1 $\stackrel{as-term}{\longrightarrow}$ like $\stackrel{mention}{\longrightarrow}$ like2 $\stackrel{nsubj-inverse}{\longrightarrow}$ boys2 $\stackrel{as-term}{\longrightarrow}$ boys , and

girls $\stackrel{mention}{\longrightarrow}$ girls1 $\stackrel{nsubj}{\longrightarrow}$ like1 $\stackrel{partmod}{\longrightarrow}$ playing1 $\stackrel{as-term}{\longrightarrow}$ playing $\stackrel{mention}{\longrightarrow}$ playing2 $\stackrel{partmod-inverse}{\longrightarrow}$ like2 $\stackrel{nsubj-inverse}{\longrightarrow}$ boys2 $\stackrel{as-term}{\longrightarrow}$ boys .

Intuitively, in a graph representing a large corpus, terms that are more semantically

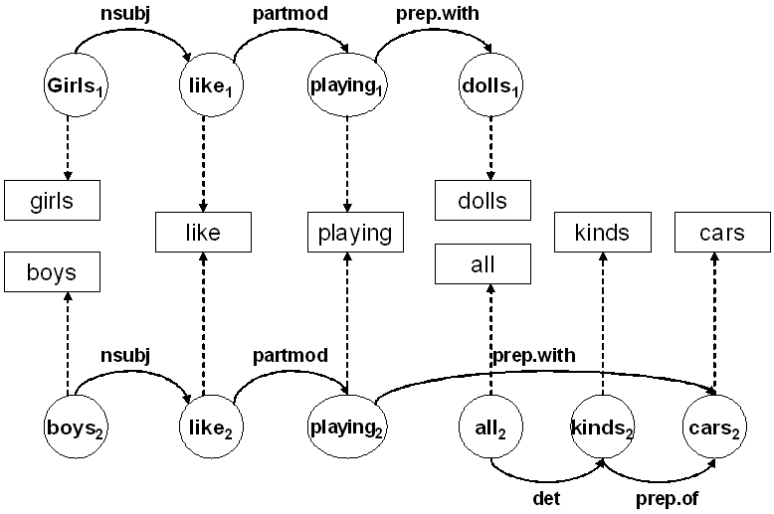


Figure 6.1: A joint graph of dependency structures

related will be linked by a larger number of connecting paths. In addition, different edge types as well as the path traversed are expected to have varying importance.

6.2 Text Processing Tasks as Queries

We evaluate the induced graph-based similarity measures and the text representation schema on the task of *coordinate term* extraction. Coordinate terms are defined as a symmetric semantic relations between words that share a hypernym in a thesaurus. For example, *wolf* is a coordinate term of *dog*, and dog is a coordinate term of wolf, since both are instances of *canine*. Similarly, named entities of the same class are coordinate terms. For instance, *New-York*, *Paris* and *Rome* are coordinate terms as they are all instances of a *city name*. Coordinate terms reflect a particular type of word similarity (relatedness), and are therefore an appropriate test case for our framework. In general, automatic extraction of coordinate terms, as well as other inter-word semantic relations, is required for the automatic construction of word thesaurus and databases of world knowledge.

While coordinate term extraction is often addressed by a rule-based (templates) approach [50], rule based extraction is best adapted for very large corpora such as the Web, where information is highly redundant and precision oriented extraction gives good results. In this test case, we shall focus on relatively small corpora. Small limited text collections

may correspond to documents residing on a personal desktop, email collections, discussion groups and other specialized sets of documents. In limited-size text collections word mentions may be scarce, and 'deeper' text processing methods should yield higher recall.

In this test case, we evaluate the extraction of *city names* and *person names* from small to medium corpora of newswire data. The task defined in the experiments is to retrieve a ranked list of city or person names given a small set of seeds. This task is implemented in the graph as a query, where we let the query distribution V_q be uniform over the given seeds (and zero elsewhere). Ideally, the resulting ranked list will be populated with many additional city or person names. Since named entities (NEs) such as cities and persons often contain more than one token (e.g., "New York", or "William Cohen") we apply available tools to first segment the text, and also identify named entity spans. Text segmentation and named entity recognition are both well studied problems, and there are various tools available that are sufficiently fast for the pre-processing of limited size corpora. Given NEs chunks, it is possible to filter the query results by node type $\tau =$"named entity". We apply this filter in the experiments. In addition, filtering results by the relevant part-of-speech tag (ADD), capitalization patterns etc. may be useful, eliminating irrelevant candidates.

Notice that high-quality retrieved lists, in which the top ranks are densely populated with correctly identified coordinate terms, should allow an iterative *bootstrapping* process. That is, given an initial seed and a retrieval machanism, the system can automatically select additional seeds using the produced ranked lists, and *re-query* the corpus, with the goal of increasing coverage. Bootstrap extraction was not included in the experiments conducted.

General word *synonym extraction* is a similar task considered in this test case. Identifying semantic relations between words using parsed text is a well studied problem. We are interested in applying the graph walk techniques to this generic problem. The query distribution V_q will consist in this case of the term of interest. The objects retrieved can be of a general type, $\tau =$*terms*. It is also possible to query for only *nouns*, *verbs* etc., according to the query word and the user intentions.

6.3 Experimental Corpora

The following corpora are used in the experiments.

MUC-6. We use the training set portion of the MUC-6 corpus [MUC6]. The MUC corpus contains articles of the Wall Street journal. It is fully annotated with named entity tags.

Another corpus used consists of articles of the Associated press (AP), extracted from

Corpus	*words*	*nodes*	*edges*	*unique NEs*
MUC	140K	82K	244K	3K
MUC+AP	2,440K	1,030K	3,550K	36K
BNC+AP	1,333K	462K	1,731K	-

Table 6.1: Corpus statistics

the **AQUAINT** corpus [15]. The AQUAINT corpus includes automatically generated, noisy, named entity tags.

Finally, we use a subset of the British National Corpus (**BNC**) **REF!**. The full BNC corpus is a 100-million word collection of samples of written and spoken language from multiple sources, designed to represent a wide cross-section of contemporary British English. We use this corpus for the evaluation of the synonym extraction task.

All corpora were parsed using the Stanford dependency parser [33].[1]. Statistics of the experimental corpora constructed and their corresponding graph representations are given in Table 6.1. The MUC corpus is relatively small, containing about 140 thousand words. A corpus constructed that included the untion of MUC data and a random subset of the AP experimental corpus (MUC+AP) is substantially larger, containing about 2.5 million words. The number of unique named entities annotated in both corpora is detailed in the "unique NEs" column in the table.

The BNC+AP corpus contains mainly texts from the BNC corpus, as well as a small addition of sentences from the AP corpus (BNC+AP), including a total of about 1.3 million words.

Further details about the considerations in corpus construction are provided in the experimental settings description.

6.4 Exeperiments and Results

We evaluate performance using graph walks with *uniform* edge weights Θ, i.e., $\theta_\ell = \theta_{\ell'}, \forall \ell$ (denoted by G:Uw), and also for graph walks where the edge weights have been tuned (G:Lw). We apply the same weight tuning procedure described in the previous chapter. Reranking is applied on top of the modified graph walk results, using the learned edge weights. For every example, the top 200 message nodes have been reranked (denoted as 'Rerank'). Path trees were learned using the positive and the top ranked negatively

[1]http://nlp.stanford.edu/software/lex-parser.shtml; sentences longer than 70 words omitted.

labelled nodes. In general, we required the number of positive and negative examples to be balanced. All models were trained using examples allocated for training and tuning purposes, whereas performance was evaluated on separate test examples.

In all of the experiments reported in this chapter we applied a stay probability $\gamma = 0.5$.

The research described in this chapter is perhaps most related to syntax-based vector space models, which derive a notion of semantic similarity from statistics associated with a parsed corpus [46, 67, 78]. In most cases, these models construct vectors to represent each word w_i, where each element in the vector for w_i corresponds to particular "context" c, and represents a count or an indication of whether w_i occurred in context c. A "context" can refer to simple co-occurrence with another word w_j, to a particular syntactic relation to another word (e.g., a relation of "direct object" to w_j), etc. Given these word vectors, inter-word similarity is evaluated using some appropriate similarity measure for the vector space, such as cosine vector similarity, or *Lin's similarity* [67] that was designed for this domain.

Recently, Padó and Lapata [78] have suggested an extended syntactic vector space model called *dependency vectors* (DV), in which rather than simple counts, the components of a word vector consist of *weighted scores*, which combine both co-occurrence frequency and the importance of a context. They considered two different context-based weighting schemes: a *length* weighting scheme, assigning lower weight to longer connecting paths (computed as inverse of path length); and an *obliqueness* weighting hierarchy [60], assigning higher weight to paths that include gramatically salient relations. Another parameter controlling the computed scores in their framework limits the set of considered paths to a manually designed set, representing various types of linguistic interesting phenomena. In an evaluation of word pair similarity based on statistics from a corpus of about 100 million words, they show improvements over several previous vector space models.

In the experiments, we compare the graph walk framework against the main two following models.

Co-occurrence model. We compare against a vector-based bag-of-words co-occurrence model. The co-occurrence model represents a more traditional approach, where text is processed as a stream rather than syntactic structures. A co-occurrence vector-space model was applied using a window of two tokens to the right and to the left of the focus word. Inter-word similarity was evaluated using cosine similarity, where the underlying co-occurrence counts were normalized by log-likelihood ratio [78].

Dependency vectors (DV). We compare graph walks also to *dependency vectors*, being a state-of-the-art syntactic vector-based model. In implementing this method, we used

code made available by the authors[2], where we converted the underlying syntactic patterns to the Stanford dependency parser conventions.

In applying the vector-space based methods, we compute a similarity score between *every* candidate from the corpus and each of the query terms, and then average these scores (as the query distributions are uniform) to construct a ranked list.

Below, for every task the specific experimental settings and datasets are presented. We will discuss the results and derive conclusions from each experiments regarding the framework.

6.4.1 Coordinate Term Extraction

In this task, queries are submitted that include a small number of seed examples representing a named entity class of interest. We require the retrieved nodes to be of type τ =*named entity*, using available corpus annotations. Empirical evalutation includes the named entity classes of *city* and *person* names.

Datasets

The MUC-6 collection provides gold standard annotations of named entities (NEs) and their types—e.g., "New York" is annotated as "Location". For the city name extraction experiments, we hand-labeled all location NEs as to whether they were city names. Overall, we identified 185 unique city names in the corpus.[3] We then generated 10 queries comprised of cities' names. Each query includes 4 city names, selected randomly according to the distribution of city name mentions in the MUC-6 corpus.

For the person name extraction task, we also generated 10 queries. Each query includes 4 randomly selected person names included in the MUC-6 corpus. For every dataset, we use 5 labelled queries for training and tuning purposes, and reserve the remaining 5 queries for testing.

In addition to the small MUC-6 corpus we constructed a larger corpora (MUC+AP), by adding to the MUC-6 corpus parsed articles extracted from the AQUAINT corpus. The AQUAINT corpus has been annotated with named entities automatically, so it includes noisy tags; nevertheless, we processed queries in the these graphs in the same manner described above. The same queries were applied to both corpora.

Statistics of the corpora and their corresponding graph representation are included in the previous section.

[2] http://www.coli.uni-saarland.de/ pado/dv.html

[3] The list was not normalized—e.g., it includes synonyms like "New York" and "N.Y".

Vector-space similarity

We evaluated the cooccurrence model (CO) and the depenednecy vector model using the test queries on both corpora. The parameters of the DV method were set based on a cross validation evaluation (using the MUC+AP corpus). The *medium* set of dependency paths and the *oblique* edge weighting scheme were found to perform best. We experimented with cosine as well as the Lin similarity measure in combination with the dependency vectors representation.

For both the graph walk models and the vector-based models, we limit the considered set of candidates to named entities. This allows us to reduce the size of the co-occurrence matrix maintained by the vector models, thus overcoming memory requirement constraints. In addition, given the large number of candidates in the MUC+AP corpus (Table 6.1), we show the results of applying the considered vector-space models to the top, high-quality, entities retrieved with reranking for this corpus. (We process the union of the top 200 results per each query, that is consider 1,000 candidates overall.)

Figure 6.2 gives results for the city name (top) and the person name (bottom) extraction tasks. The left part of the figure shows results using the MUC corpus, and its right part – using the MUC+AP corpus. The given curves show precision as a function of rank in the ranked list, up to rank 100. (We hand-labeled all the top-ranked results as to whether they are city names or person names.) Included in the figure are the results for the co-occurrence model (CO), and for the syntactic vector-space DV model, using the Lin similarity measure (DV:Lin). Performance of the DV model using cosine similarity was found comparable or inferior to using the Lin measure, and was omitted from the figure for clarity.

Out of the vector-based models, the co-occurrence model is preferable for the city name extraction task, and the syntactic dependency vectors model gives substantially better performance for person name extraction. We conjecture that city name mentions are less structured in the underlying text. In addition, the syntactic weighting scheme of the DV model is probably not optimal for the case of city names. For example, a *conjunction* relation was found highly indicative for city names (see below). However, this relation is not emphasized by the DV weighting schema. As expected, the performance of the vector-based models is improved with the size of the corpus [97]. These models demonstrate good performance for the larger MUC+AP corpus, but only mediocre performance for the smaller MUC corpus.

Graph walks

We evaluated cross-validation performance over the training queries in terms of mean average precision on varying walk lengths K. We found that beyond $K = 6$ improvements were small (and in fact deteriorated for $K = 9$). We therefore set $K = 6$. The results of

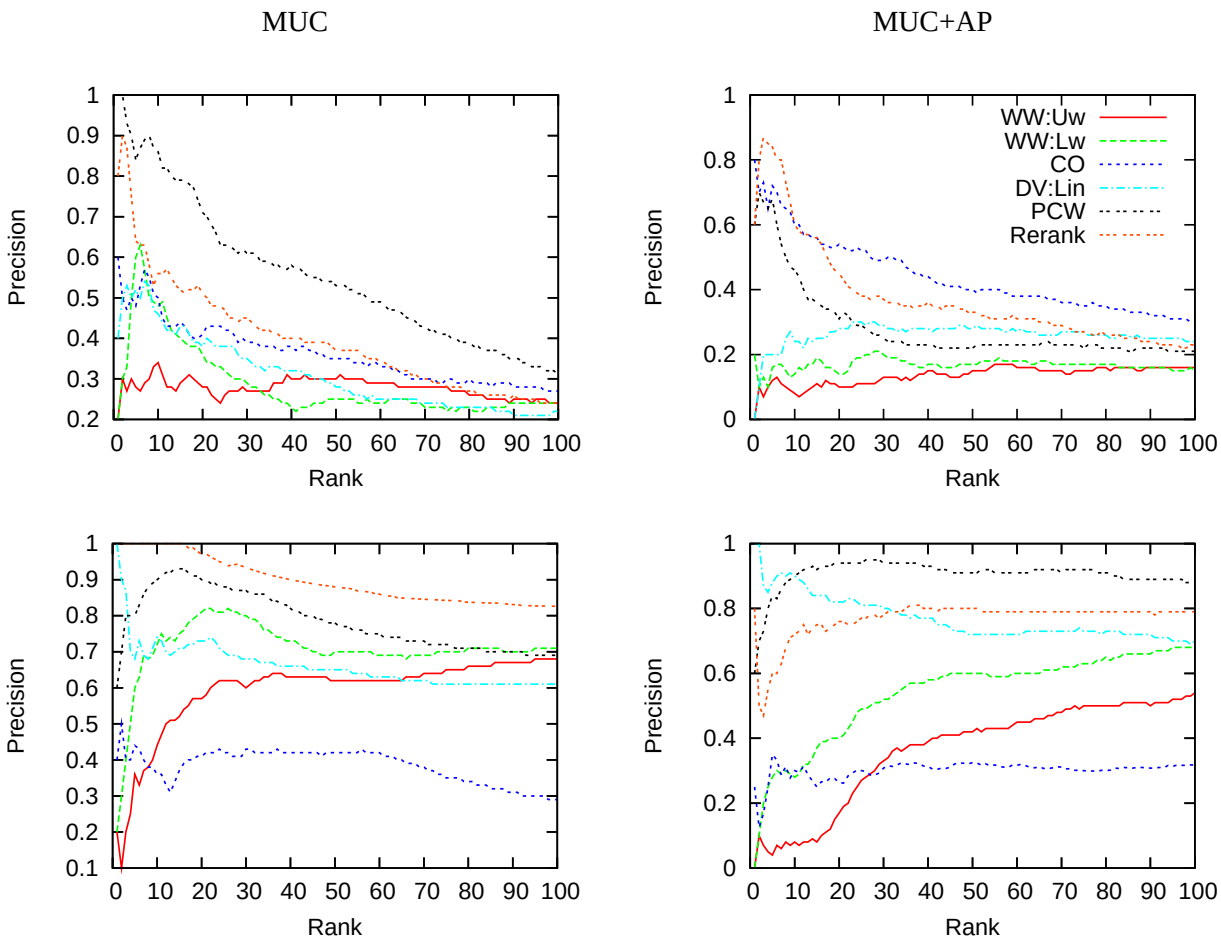


Figure 6.2: Test results: Precision at the top 100 ranks, for the city name extraction task (top) and person name extraction task (bottom).

appyling graph walks with uniform weights are shown in Figure 6.2 (G:Uw).

The performance of graph walks is inferior to the vector space model for the city name extraction task. For person name extraction, the graph walks achieve higher recall than the cooccurrence model, but lower accuracy at the very top ranks. In studying the results, we observed that words (more specifically, named entities) adjacent to the seed named entity in a sentence, got high weights in general. That is, due to the exponential decay over path length embedded in the graph walk paradigm, other named entities that were linked to the query nodes over possibly a large number of longer paths, were ranked below the immediate query node neighbors. As shall be shown, in the problem settings considered, there are longer connecting paths that are more meaningful than arbitrary shorter connecting paths. We next discuss leanring results, and show how learning allows to promote node that are distant, yet connected to the query over a large number of meaningful paths.

Learning

Weight tuning was trained using the training sets and two dozens target nodes. High weights were assigned to edge types such as *conj-and*, *prep-in* and *prep-from*, *nn*, *appos* and *amod* for the city extraction task. For person extraction, prominent edge types included *subj*, *obj*, *poss* and *nn*. The latter preferences are similar to the liguistically motivated weights of the dependency vectors model.

For reranking, we set a feature count cutoff (at 3) in order to avoid over-fitting. Reranking was applied to the top 200 ranked nodes output by the graph walk using the tuned edge weights.

High weight features assigned by reranking for city name extraction included, for example, lexical features such as “based” and “downtown”, and edge bigrams such as “prep-in-Inverse→conj-and” or “nn-Inverse→nn”. Positive highly predictive paths in the constructed path tree included many symmetric paths, such as ...→conj_andInverse...→.conj_and..., ...→prep_inInverse...→.prep_in..., for the city name extraction task.

Finally, path-trees were constructed using the top 20 correct nodes and 20 incorrect nodes retrieved by the uniformly weighted graph walk. More specifically, labels about the relevancy of each node retrieved were available to us in the MUC corpus. In the MUC+AP corpus, however, we could not readily identify negative examples. For this corpus, we therefore considered nodes not known to be correct answers as incorrect responses; i.e., the training data in this case is noisy. In the experiments, we apply a threshold of 0.5 to the path constrained graph walk method.

Figure 6.2 shows the precision-at-rank curves for the graph walk method using the learned weights (G:Lw), the same graph walks with reranking (G:Lw+RR) and a path-constrained graph-walk (PCW).

Several trends can be observed from the results. The graph walk using the learned edge weights consistently outperforms the graph walk with uniform weights. Reranking and the path-constrained graph walk, however, yield superior results. Both of these learning methods utilize high-level features compared with the graph walk and weight tuning, which consider only local information. In particular, while the graph walk paradigm assigns lower importance to longer connecting paths, reranking and the path-constrained walker allow to discard short yet irrelevant paths, and by that eliminate noise at the top ranks of the retrieved lists.

Contrasting the graph-based methods with the vector-based models, the difference in performance in favor of reranking and PCW, especially for the smaller corpus, can be attributed to two factors. The first factor is learning, which optimizes performance for the underlying data. A second factor is the incorporation of non-local information, which allows to consider high-level properties of the traversed paths.

6.4.2 General Word Similarity

The queries are typed. That is, if the query term is a *noun*, it is required to retrieve terms that are tagged as nouns in the corpus (part-of-speech tags are included in the parse information).

Datasets

We collected pairs of word synonyms from teaching materials for foreign students. We then constructed an experimental corpus, extracting sentences that contain the subject words. For common words that appeared in thousands of sentences, we didn't use all the sentences available. On the other hand, for scarce words, we added to the corpus additional sentences from the AP corpus.

Table 6.2 details the synonyms pairs. We distinguish between nouns, adjectives and verbs. For each part-of-speech type, we use 10 synonym pairs as queries (the term mentioned first for each pair is the query, and the other term is considered as a correct answer). The remaining synonym pairs were used to construct test queries.

Vector-space similarity

For scalability reasons, rather than store statistic and rank all words in the vector space models, we first applied the graph walks, and consider the top 400 words retrieved per query (the union).

We applied a simple co-occurrence model (CO) that does not consider the syntactic relations available in the parse structure, as described above.

	Training examples	*Test examples*
Adjectives	contemporary : modern immediate : instant lethal : deadly particular : specific deliberate : planned gay : homosexual dubious : doubtful infamous : notorious imperative : vital lucid : clear	infrequent : rare dedicated : committed necessary : essential pressing : urgent informal : casual isolated : lonely legitimate : valid constant : fixed exact : precise economic : profitable essential : fundamental attractive : appealing intelligent : clever prosperous : affluent
Nouns	commencement : graduation convention : conference destiny : fate hunger : starvation hypothesis : speculation material : fabric movie : film possibility : opportunity remorse : regret association : organization	murderer : assassin disaster : catastrophe discount : reduction impediment : obstacle homicide : murder measure : degree interplay : interaction inflow : influx meeting : assembly ballot : poll bid : tender comfort : consolation
Verbs	answered : replied conform : comply disappeared : vanished cited : quoted diminished : decreased enquire : investigate evaluated : assessed inspected : examined renewed : resumed demonstrated : protested	oversee : supervise received : got admitted : confessed began : started closes : shuts confine : restrict disclose : reveal illustrate : demonstrate assure : guarantee illuminated : clarified nominated : appointed responded : replied renewed : resumed

Table 6.2: Word synonym pairs: train and test examples

	CO	DV:Cos	DV:Cos-ll	DV:Lin	G:Uw	PCW	Rerank(PCW)
Adjectives	0.07	0.18	0.21	0.41		0.34	0.34
Nouns	0.05	0.21	0.29	0.55		0.17	0.36
Verbs	0.04	0.13	0.22	0.45		0.40	0.27

Table 6.3: Message tracking results: MAP

In addition, we applied the depenedncy vectors model, using a simple cosine similarity measure (DV:Cos); using a cosine similarity measure, where the statistics have been first normalized by log-likelihood (DV:Cos-ll), and using the Lin similarity metric (DV:Lin).

The results are presented in Table 6.3 in terms of mean average precision. While we evaluate performance using our self constructed experimental corpus, the results are consistent with similar experiments reported in the literature [78]. That is, the model of dependency vectors, which considers syntactic relations in computing cooccurence weights performs preferably to a simple cooccurence model. Adding pre-processing of log-likelihood normalization of the statistics improved DV performance further. Lastly, using the Lin measure, specialized for this model fo inter-word similarity, reaches high levels of performance.

Graph walks

The results of graph walks using uniform weights (G:Uw) are ...

Learning

In preliminary experiments where we evaluated this framework on the task of extracting general word synonyms, using a relatively large corpus of 15 million words, we found the graph-walk performance to be better than DV using cosine similarity measures, but second to DV using Lin's similarity measure. While this set of results is incomplete, we find that it is consistent with the results reported in this paper.

What are the paths learned by PCW? and by reranking?

Any reason why models should be separated at all for nouns / verbs / adjs?

The Lin measure is better, but here a lot of data has been used... also, there are many possible directions for further improvements of the learning models. And, the graph walks were in fact used here as a preliminary mechanism for retrieving relevant candidates. This is an issue in vector space models (see email from Lin), in terms of scalability.

6.5 Related Work

This work is not the first to apply graph walks to obtain a notion of semantic similarity for NLP problems. Toutanova et-al [100] constructed a directed graph, where nodes represented word types, and the edges denoted various types of inter-word semantic relations, extracted from WordNet. They applied graph walks to infer a measure of word similarity. The semnatic similarity scores were used for lexical smoothing for the task of prepositional word attachment. Recently, Hughes and Ramage [55] constructed a similar graph, representing various types of word relations from WordNet, and compared the random-walk generated word similarity to similarity assessments from human-subject trials. Another work in the field of information retreival employed random walks in a graph that included word relations from WordNet and other resources as well as corpus co-occurrence based measures, for query expansion [30]. In this chapter we consider *corpus-based* word similarity measures, using syntactic information. While previous works were tailored to extract a particular flavor of word similarity, with the goal of improving the performance of a specific end application, we use learning to tune the generated similarity measure per task. In the experiments reported, we were mainly interested in comparing the graph walk performance against corpus-based vector-space models that use parsed text as their input. It is straight-forward, however, to add external resources such as WordNet relations to the graph, thus integrating corpus-based and lexicons. This is a possible direction for future research.

There have been multiple works that applied PageRank style graph walks for natural language applications; i.e., using node centrality (or prestige) scores. For example, it has been suggested to construct text graphs for automatic text summarization, where nodes are sentences and (undirected and weighted) links are drawn between similar sentences [38, 71]. In these graphs, sentences assigned a high centrality scores are considered as salient, and can be used to construct an automatically generated summary. Mihalcea [70] applied graph walks also for word sense disambiguation. She used a graph representing the possible WordNet synsets of words per sentence, where the synset nodes were connected with directed edges, weighted by the overlap in their definition in WordNet. Graph walks using the pageRank paradigm were then used to select the most probable synset per word. To our knowledge the research presented in this chapter is novel in representing a corpus as a graph that includes syntactic information (in particular, dependency-parsed text), and is novel in exploring the use of random-walk similarity on such a graph. Compared with the abovementioned works, our goal is not to deduce centrality scores, but to learn inter-word similarity measures.

We note that graphs derived from individual sentences have been widely used. For

example, Snow et al [93] used dependency paths in order to extract hyponyms from a corpus of parsed text. In particular, they extracted patterns from the parse tree of sentences in which hyponym word pairs co-appear, and trained a hyponym classifier using these patterns as features. Overall, they created a feature lexicon of about 70,000 dependency paths, consisting of frequent dependency paths that occurred between noun pairs in their corpus. The authors indicate that due to data sparsity, the ratio of relevant sentences in the corpus is low. In contrast, we represent text corpora as a connected graph of dependency structures, using either within- or across-sentence paths.

Dependency paths of individual sentences have been used also for general relation extraction. Culotta and Sorenson [32] explore the detection and classification of instances of relations, where relations correspond to meaningful connections between two entities (e.g., “based-in”, “member”, “spouse”). They represent each relation instance as a dependecy tree, augmented with features for each node, including part-of-speech tags, entity type, WordNet synsets etc. For each pair of entities in a sentence, the smallest common subtree in the dependency tree that includes both entities is found. Based on the hypothesis that instances containing similar relations share similar substructures in their dependency trees, the authors propose kernel functions that estimate the similarity between the subtrees. Empirical evaluation results showed that the tree kernel approach outperformed a bag-of-words kernel, implying that the structural information represented in the tree kernel is useful for the relation extraction problem. Bunescu and Mooney [16] observed that the information required to assert a relationship between two named entities in the same sentence is typically captured by the shortest path between the two entities in the undirected version of the dependency graph. Words are tagged with part-of-speech, entity type and other features. They propose a kernel which captures a dot product of the common features in the shortest paths, using it with SVM to classify new instances.

General relation extraction requires the analysis of individual sentences. The graph representation we suggest, however, may be used also in such settings. The rich feature set encoded by the described kernels can be represented in the graph and given to a suitable reranker. Rather than address the relation extraction problem as a classification problem, graph walks would approach it as a ranking (or, retrieval) problem. General relation extraction is a challenging problem, and it is an exciting direction for future work.

6.6 Summary

In this paper we make several contribuions. First, we have explored a novel but natural represention for a corpus of dependency-parsed text, as a labelled directed graph. We

have evaluated the task of coordinate term extraction using this representation, and shown that this task can be performed using similarity queries in a general-purpose graph-walk based query language. Further, we have successfully applied learning techniques that tune weights assigned to different dependency relations, and re-score candidates using features derived from the graph walk.

Empirical evaluation of the coordinate term extraction task suggest that the graph-based framework performs better than state-of-the-art vector-space models given small corpora. Overall, we find that the suggested model is suitable for deep (syntactic) processing of small specialized corpora.

The framework presented can be enhanced in several ways. For instance, WordNet edges and morphology relations can be readily encoded in the graph. Finally, we believe that this framework can be applied for the extraction of more specialized notions of word relatedness, as in general relation extraction.

Chapter 7

Design and scalability considerations

Given the two test cases that have different characteristics, interested in drawing some conclusions about the design choices. In practice, scalability plays a major role in desinging a system. Scalability considerations are also included in this chapter.

7.1 Graph walk parameters

As described in Chapter 3, the graph walk formula includes two parameters: the stay probability γ (also known as "reset" or "damp" probability); and, since we perform *finite* graph walks, another parameter is the length of the walk k.

In this section, we give the results of experiments, where we vary these parameters to evaluate their effect on the result.

7.1.1 Walk Length

Table 7.1 gives the results of graph walks for varying number of walk steps k, in terms of mean average precision. These results include the person name disambiguation, threading and alias finding tasks, for a total of eight datasets. For every dataset, the best results are marked in bold.

The table shows the following trends. For the person name disambiguation task, performance is comparable and slightly improves with the length of the walk, for the management game corpus. In the case of the two Enron datasets, however, performance is substantially better for the short walk length, where $k = 2$, converging to an inferior result

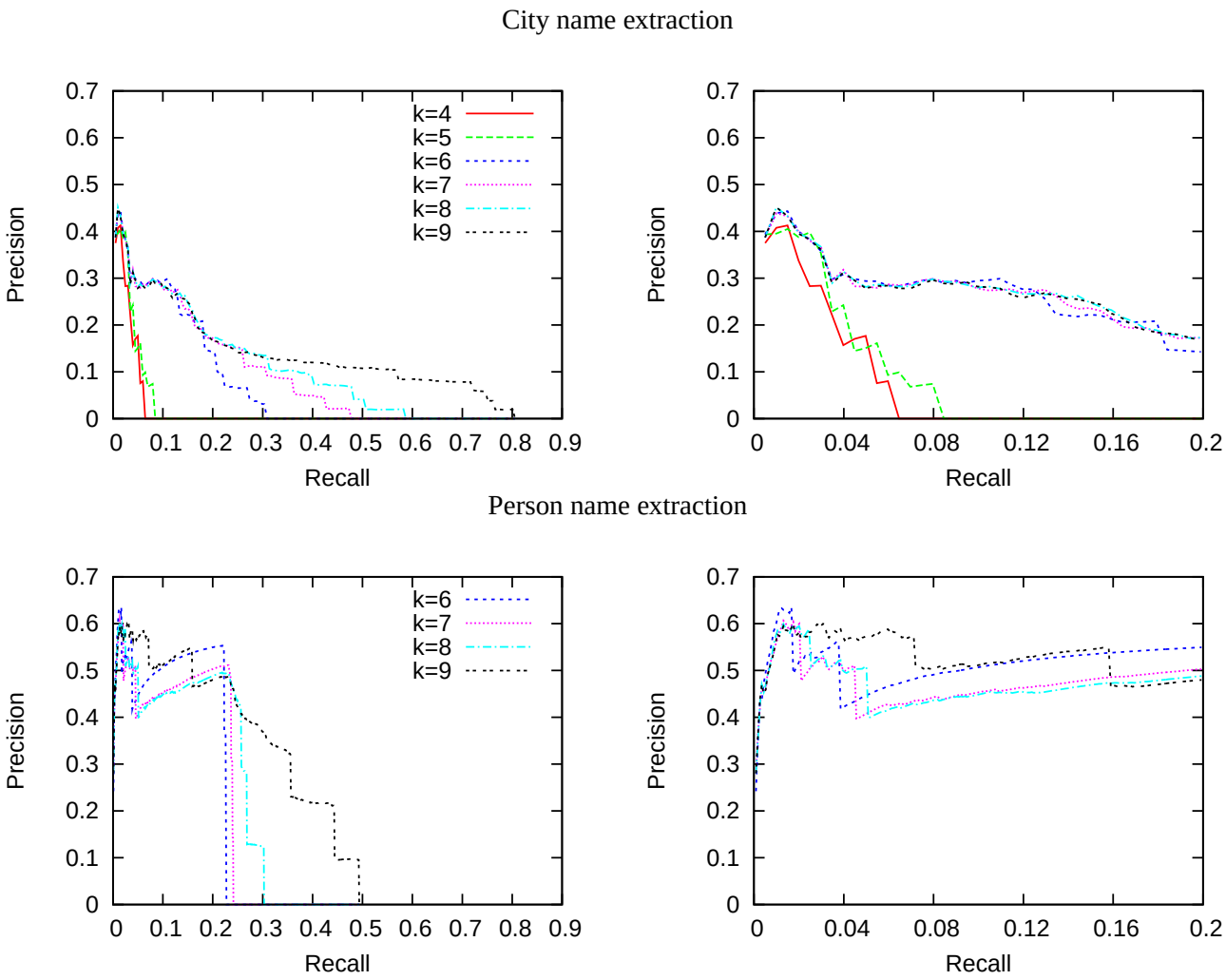


Figure 7.1: Precision-recall curves varying the walk length k for city name extraction (top) and person name extraction (bottom). The left graphs show the full curves, and the right graphs focus on the top of the lists (down to recall 0.2). These results were all generated using the MUC corpus.

Corpus	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
Person name disambiguation							
M.Game	0.65	0.67	0.66	0.66	**0.67**	**0.67**	**0.67**
Sager	**0.67**	0.56	0.56	0.56	0.56	0.56	0.56
Shapiro	**0.61**	0.46	0.44	0.43	0.43	0.43	0.43
Threading							
M.Game	**0.53**	0.52	0.50	0.50	0.50	0.49	0.49
Germany	0.55	**0.56**	0.49	0.49	0.49	0.48	0.47
Farmer	**0.65**	0.64	0.58	0.58	0.57	0.56	0.56
Alias finding							
Meetings	0.60	0.72	**0.73**	**0.73**	**0.73**	0.72	0.72
Personal	0.58	0.61	0.62	**0.63**	**0.63**	**0.63**	0.63

Table 7.1: Results of applying graph walks using uniform edge weights, varying the graph walk length parameter k ($\gamma = 0.5$).

as the length of the walk increases. For the threading task, performance is better for short walks of length $k = 2$ or $k = 3$ across all corpora. Finally, for the alias finding task, there is a substantial improvement increasing the walk length from $k = 2$ to $k = 3$, where performance only improves slightly for longer graph walks.

In all of the experiments, performance converges for walks shorter than $k = 8$ steps.

In another set of experiments, we evaluate the effect of the walk length for the language domain. In this domain, unlike the previous experiments, every query is associated with a large number of correct answers. The mean average precision in this case is therefore fairly low (since many correct answers that are not reached contribute zeros to the overall mean average precision score). On the other hand, having a many correct answers in a fully annotated corpus such as MUC (see Section x), allows us to show a precision-recall curve – thus, evaluate the effect of walk length on both precision and recall performance.

Figure 7.1 shows the precision-recall curve for the city name extraction task (top) and the person name extraction task (bottom). The left part of the figure show the full curve for the two task; the right part shows the curve for the top of the lists retrieved (down to recall of 0.2) in larger detail. The left graphs show that increasing the graph walk length increases recall. Indeed, longer walks allow probability mass to propagate to nodes that are more distant from the query nodes. In particular, for both tasks, short walks of $k <= 4$ yielded poor recall. (For the person name extraction task, recall was near zero, and we therefore eliminate these curves from the figure.) Indeed, in this domain, there are very few relevant nodes that can be reached in very short walks – for example, over the

Corpus	k	$\gamma = 0$	$\gamma = 0.15$	$\gamma = 0.3$	$\gamma = 0.5$	$\gamma = 0.7$	$\gamma = 0.85$
Person name disambiguation							
M.Game	2	0.30	0.66	0.67	0.65	0.66	0.66
Sager		0.18	0.67	0.67	0.67	0.67	0.67
Shapiro		0.14	0.60	0.61	0.61	0.61	0.61
Threading							
M.Game	2	0.02	0.53	0.53	0.53	0.53	0.53
Germany		0.01	0.55	0.55	0.55	0.55	0.55
Farmer		0.01	0.65	0.65	0.65	0.65	0.65
Alias finding							
Meetings	3	0.22	0.60	0.60	0.60	0.60	0.60
Personal		0.38	0.67	0.67	0.67	0.67	0.67

Table 7.2: Results of applying graph walks using uniform edge weights, varying the graph walk length parameter k ($\gamma = 0.5$):MAP.

path "contains – conj-and – containsInv". Such "syntactic co-occurrence" phenomenon is relatively scarce, and appears more for city names compared with person names in our experimental corpora. Once the length of the walk allows reaching longer significant paths (modeling inter-sentence syntactic ressemblence, in this domain – e.g., the path that models a common direct object argument is modeled by a path of length six: , as well as many other paths), then increasing the walk length further improves recall, but the additional entities retrieved are generally ranked below the entities reached earlier, due to the exponential decay embedded in the graph walk. The figure shows that a change in the rankings of the nodes already reached is minor.

Check - at what stage is does the walk reach all of the graph nodes?

We also applied a branching threshold of 20,000 nodes.

7.1.2 Stay Probability

In all of the experiments reported thus far, the stay probability was somewhat arbitrarily set as $\gamma = 0.5$. It has been shown that this parameter, while it affects the actual scores assigned to the graph nodes, does not change their relative rankings [79]. A common choice of the reset probability (clarify - which side of the reset are we talking about) is 0.15 [79] – also cite here a work by Faloutsos and Daniel Ramage.

Table 7.2 show the results of tuning the stay probability parameter γ from 0, to a high

Corpus	G:U^n	G:L^n	G:U^{un}	LGW^n	LGW^{un}
Person name disambiguation					
M.Game	0.65	0.67	**0.68**	**0.68**	0.62
Sager	0.67	**0.81**	0.67	0.76	0.78
Shapiro	0.61	**0.80**	0.61	0.78	0.70
Threading					
M.Game	0.53	0.59	0.65	**0.66**	0.62
Germany	0.55	0.55	0.58	0.47	**0.60**
Farmer	0.65	**0.72**	0.65	**0.72**	0.55
Alias finding					
Meetings	**0.61**	0.55	0.60	0.60	0.55
Personal	**0.72**	**0.72**	**0.72**	**0.72**	0.60

Table 7.3: Results of applying graph walks using uniform edge weights, varying the graph walk length parameter k ($\gamma = 0.5$):MAP.

0.85, in terms of mean average precision. Specifically, $\gamma = 0$ equals a different model, of a surfer that continues from a node to its neighbors, without re-setting. In general, the PageRank and Personalized Pagerank models requires $\gamma > 0$.

The table shows that large changes in γ have no to negligible effect on the actual produced rankings.

In all experiments, we assign stay probability $\gamma = 0.5$ to the graph walk, leading to fast convergence.

7.1.3 Graph walk variants

As described in Section 3.2.1, there are several variants of graph walk schemas that researchers have adopted in the past in performing random graph walks over a typed graph.

In the experiments conducted so far, we adopted the prevalent Personalized Pagerank graph walk model, where the perference vector corresponds to qyery distribution over the graph node.

In this section, we provide empirical results for applying different graph walk schemes. More specifically, we evaluate the following variants:

Personalized Pagerank graph walk, with uniform weights (PP:U) – this is the Personalized Pagerank model, applied so far.

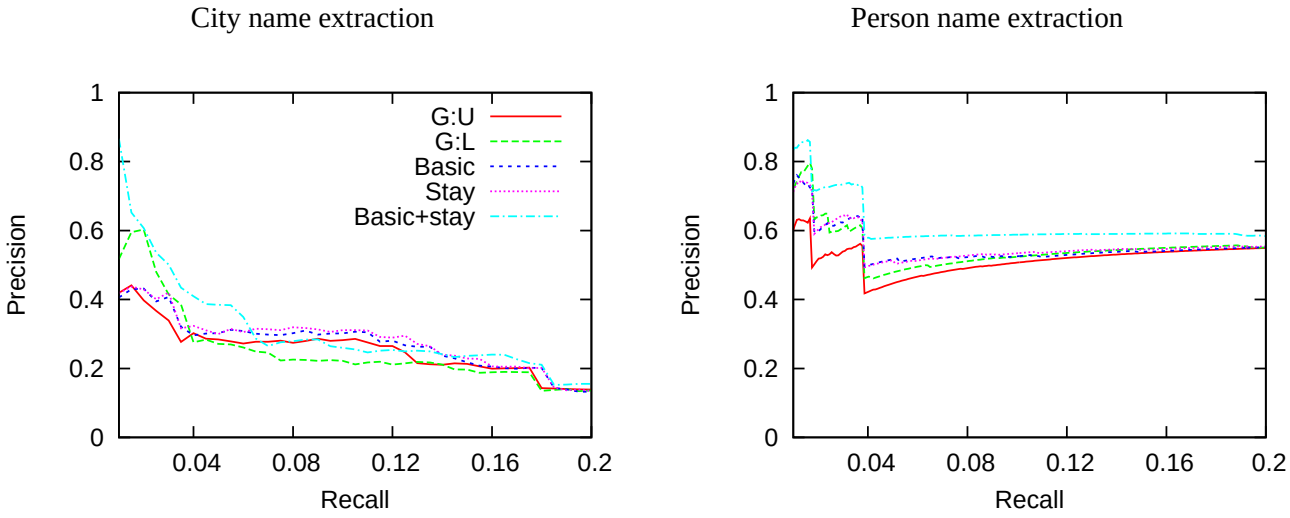


Figure 7.2: aa.

Lazy graph walk (LGW) – graph walks, where at each step of the walk, the random walker stays at the current node with probability $\gamma = 0.5$ and continues to the neighboring nodes with the remaining probability $1 - \gamma = 0.5$.

In addition, we evaluate two edge weighting schemas, in conjunction with each of the models:

Normalized edge weights – this is the weighting schema used in the experiments reported. Namely, this means that each outgoing edge is assigned a typical edge weight by its type, and the walker picks an edge at random according to its weight (give formula).

unnormalized edge weights – in this model, the walker first picks an edge type at random. Given the edge type, a specific edge is selected uniformly. In these settings, the total outgoing probability mass is deficient in case that not all of the possible outgoing edge types are present.

Table 7.3 gives the results for the combinations of the two walk paradigms and the two edge weighting schema (denoted by a superscript n for the 'normalized' version, and with a superscript un, for the 'unnormalized' variant. For all methods, edge weights were assigned uniformly. In addition, we included the results of graph walk after learning edge weights, per the previously reported experimental results. The table includes results for three tasks, and multiples datasets.

The results indicate that applying the unnormalized edge weigting schema is in fact

preferable to the normalized edge weighting schema. In addition, the lazy graph walk version often yields the best results (LGW with normalized weights is best in four out of the eight datasets, and LGW using unnormalized is best for another instance).

Figure 7.2 show performance in terms of a precision-recall curve for city name extraction and person name extraction from the MUC corpus. (As before, since there are many relevant correct answers and recall is partial, this presentation is preferable in this case to mean average precision.) These graphs deliver similar results, also for the different graph representation of a parsed corpus. In this case, the lazy graph walk, using unnormalized weights gives a superior precision-recall curve. In fact, its performance is also preferable to the Personalized Pagerank graph walk schema, where the edge weights have been learned.

We conjecture the following about the causes to thie phenomena.

WHAT ARE THE REASONS??

Are the methods equivalent, given optimal edge weights?

Do the methods of weight tuning, reranking and path constrained walks apply here as well (think that yes.).

Would like to expand the exeriments and include learning also for the other walk variants. A bottom line is that the selection of the graph walk paradigm should not be arbitrary, as it may have a large effect on the results.

7.2 Learning

As discussed in Chapter 5, several researchers have suggested schemes for adjusting the set of edge weights using hill-climbing methods in the Personalized Pagerank settings [35, 77, 3]. However, we have shown that for some tasks, considering edge *sequences* that are encountered in travelling from the source nodes to a target node is informative in evaluating the inter-relatedness of these nodes. Adjusting the graph parameters based on "local" information only may be thus sub-optimal. (The notion of global and local information has been introduced in Section 4.5.)

The reranking approach parameterizes the graph walk with a set of representative features, which allow one to capture certain global properties of the graph walk. However, this representation loses some quantitative information compared with exact gradient computing. Given this qualitative difference, we conducted an empirical evaluation of the weight tuning and reranking. We present and discuss the results in Section 7.2.1.

Corpus	G:R	G:L	$\text{Rrk}_{G:R}$	$\text{Rrk}^{+}_{G:R}$	$\text{Rrk}_{G:L}$	$\text{Rrk}^{+}_{G:L}$
Person name disambiguation						
M.Game	0.61	0.67	0.63	0.83*	0.65	**0.85***†
Sager	0.65	0.81*	0.72	**0.89***	0.72	0.83*
Shapiro	0.70	**0.80***	0.52	0.75	0.52	0.79*
Threading						
M.Game	0.52	0.59*	**0.75***†	-	0.74*†	-
Germany	0.51	0.55*	0.66*†	-	**0.68***†	-
Farmer	0.68	0.72	0.83*†	-	**0.87***†	-

Table 7.4: Performance comparison (MAP) of graph walks with random weights (G:R), weight tuning (G:L), reranking using edge sequence features ($\text{Rrk}_{G:R}$) and the combination of weight tuning and reranking ($\text{Rrk}_{G:L}$). Reranking using the full set of features is denoted as Rrk^{+}.

The path constrained walk approach considers global information as well, whereas the path tree vertices represent the history of the walk such that probabilities are derived at path level, rather than locally. In the experiments, however, we construct path trees using acyclic paths only. Section 7.2.1 includes a comparison of path constrained walks with weight tuning in terms of performance.

Finally, it is possible to combine several learning methods. In particular, it is straightforward to apply weight tuning and reranking as a pipeline, where the output of the graph walk with the set of tuned weights is provided to the reranker. Similarly, path constrained walks can be used to generate an initial ranking to be processed by the reranker. In Section 7.2.2, we provide empirical results for the utility of these learners combinations.

Finally, Section 7.2.3 discusses the effect of the theshold applied to the path constrained walks, including a set of relevant empirical results.

7.2.1 Local vs. Global Learning

Reranking vs. weight tuning.

We compare the gradient descent and reranking as follows. Since the gradient descent algorithm is prone to converge to a local minima, we ran the algorithm for every task and corpus (train set) combination for 5 randomly generated initial graph edge weight parameter sets Θ, out of which we consider the parameters for which the best end result is reached by the gradient algorithm, Θ^0. The output of this procedure is a modified set

Corpus	G:U	PCW	Rrk$_{G:U}$	Rrk$^{+}_{G:U}$	Rrk$_{PCW}$	Rrk$^{+}_{PCW}$
Person name disambiguation						
M.Game	0.65	0.65	0.65	**0.85***†	0.69	0.84*†
Sager	0.67	0.76*	0.72	0.82*	0.68	**0.89***†
Shapiro	0.61	0.62	0.52	**0.78***†	0.65	0.77*†
Threading						
M.Game	0.53	0.73*	0.73*	-	**0.75***	-
Germany	0.55	0.65*	**0.72***	-	0.67*	-
Farmer	0.65	0.76*	0.83*	-	**0.85***	-

Table 7.5: Performance comparison (MAP) of graph walks with uniform weights (G:U), path constrained walk (PCW), reranking using edge sequence features (Rrk$_{G:U}$) and the combination of path constrained walks and reranking (Rrk$_{G:L}$). Reranking using the full set of features is denoted as Rrk^{+}.

of weights Θ^G; we then applied graph walks using Θ^G to evaluate performance on the test set queries.

Re-ranking was trained separately, using both the *train* and *development* sets, where for comparison reasons, the same set of initial random graph edge weights Θ^0 was used to generate the graph walk output. Thus, both methods are compared against the same baseline. (Conversly, in the previous experiments reported in Chapter 5, re-ranking was given the output of the graph walk with uniform weights.) For every example, the top 50 nodes were re-ranked.

In this section, we compare reranking with graph parameter tuning as *alternative* learning methods that improve on graph walk performance. The features we use for reranking will therefore be derived from the set of paths leading to every candidate node (that is, the same information available to the error backpropagation algorithm), describing non-local properties of these paths. In particular, the considered features include *edge label bigrams* and the *source count* feature. As defined in Chapter **??**, the latter feature indicates the number of different source nodes in the set of connecting paths leading to the candidate node. Performance for reranking using the full set of features (as defined in Chapters 5 and 6 are given as well.

Table 7.4 includes results for the contextual version (where queries consist of file and term nodes) of the *person name disambiguation* task and *threading*, using the set of relevant corpora, in terms of mean average precision. The table includes the evaluation of graph walk with the baseline set of randomized weights Θ^0 (G:R). It also gives the result of applying graph walks with the learned set of edge weights (G:L); reranking of graph

walk using edge weights Θ^0, where only path-describing features are used (Rrk$_{G:R}$); and reranking using a full feature set (Rrk$^*_{G:R}$). Results that were found significantly different, using a two-sided Wilcoxon test at 95% confidence level, are marked with a asterix, with respect to the random weights baseline. Results that were found significantly different than the weight tuning performance are marked with a dagger in the table.

The results show that weight tuning is more effective than reranking in using graph walk information for the person name disambiguation task. For two of the corpora, weight tuning gives a significantly better result than the baseline graph walk, whereas the improvements of the reranking method are not significant. In one case reranking performance is inferior to the baseline (for the *Shapiro* dataset). For the threading task, however, reranking gives significantly better results using path information only compared with both the baseline and weight tuning, for all datasets.

There are several reasons for the observed trends. In the threading task, an adjacent message in a thread is often a reply-to message, where a recipient becomes the sender and vice versa, etc. This composite relation is captured by edge bigrams such as *sent-to*$\rightarrow$ *sent-from-inverse*. The gradient descent, however, does not model multi-steps dependencies, and therefore yields smaller improvements for this task. In the person name disambiguation task, on the other hand, it appears that name resolution is based on entity associations (co-occurrences), and edge sequence specification is less useful in this case.

As shown in the table, however, using a richer set of features in reranking, namely string similarity measures, allows reranking to eliminate noisy nodes from the ranked list, which results in superior performance.

In addition, we refer the reader to the results reported earlier for the city name and person name extraction tasks (Figure 6.2). In this domain of parsed text there are dozens of different edge types. While weight tuning improved the graph walk performance compared to using uniform weights, reranking gave superior results to weight tuning across tasks and corpora in this domain. There may be a couple of main reasons explaining this behavior. First, gradient descent is more likely to reach local minima in an environment that includes a large number of variables (i.e., the error surface becomes even less smooth). Another reason, which is supported by the subject matter, is that edge sequences are very meaningful in determining the relevance of words in a parse structure.

We therefore conclude that reranking, while losing some quantitative data considered by the weight tuning algorithm, leads to preferable results compared with weight tuning for various tasks, due to its modeling of global properties of the walk, and its capacity for representing additional relevant features.

Path constrained walks vs. weight tuning.

Table 7.5 gives results for applying path constrained walks to the same tasks and datasets. In the experiments we constructed path trees using the top nodes ranked by graph walks with uniform weights (denoted as G:U). We applied path constrained walks using a threshold of 0, i.e., considering all paths. The results of applying the path constrained walks are shown in the Table (PCW). Results that were found to be significantly different from the the graph walks with uniform weights are marked with an asterix.

Contrasting the contrained graph walk and weight tuning results (G:L in Table 7.4 reveals similar trends, as observed with reranking. That is, weight tuning gives preferable results for the person name disambiguation task, whereas for threading, the path constrained walks yield better results for all datasets. This again supports our claim about the importance of modeling edge sequence information for tasks such as threading.

Referring to the results in the domain of text representation (Figure 6.2) – also in this case, trends are consistent with our conclusions above. Namely, the path constrained walks give much better performance than weight tuning. This again shows the importance of modeling edge sequences rather than local information for this domain.

Path constrained walks vs. reranking.

Table 7.5 details the outcome mean average precision of performing path contrained walks (PCW), as well as reranking the output of the baseline graph walks using uniform weights. Results for applying reranking using graph walk describing features only (see above) is denoted as $\text{Rrk}_{G:U}$. The performance of reranking using the full set of features is given in the $\text{Rrk}^{*}_{G:U}$ column.

Overall, the performance of path constrained walks and reranking, using path describing features only, is comparable. Reranking using string similarity and the *source-count* feature ($\text{Rrk}^{*}_{G:U}$), however, gives superior results for the person name disambiguation task. Indeed, the path constrained walk approach is more restricted and less 'global' then reranking, as it cannot not accomodate external sources of information (such as string similarity in this case) or model properties of the set of paths connecting to a node (such as the *source-count* feature), as opposed to individual paths.

In the language domain, the performance of path constrained walks and reranking (using both edge sequence features and lexical evidence) was shown to be roughly comparable. In this domain, where graphs are larger, the path constrained walks also have the utility of improving the graph walk scalability. This as well as other implementation aspects are dicussed later in Section 7.3.

Mention that reranking only has bigram features, versus full paths.

7.2.2 Combining Learning Methods

Reranking can be affected by the quality of the input ranked lists in two ways. First, as rernking is applied to the top k nodes, its recall is limited by the number of correct answers retrieved by the initial ranker in the top k positions. Secondly, the original node score assigned by the initial ranker to the output nodes is used as a feature by the reranker. Therefore, better initial scoring should contribute also to the reranking process.

In this section, we consider the setting of improving the graph walk rankings using learning, and then applying reranking to the modified ranked lists.

We combined weight tuning and reranking, as follows: graph rankings were generated using the set of weights as modified by the gradient learner; then, reranking was applied given these output ranked lists. The results are included in Table 7.4. The combined approach, where reranking uses graph walk describing features, is denoted by 'Rrk$_{G:L}$'. Results of reranking using the full feature set are given in the column named 'Rrk$^*_{G:L}$. Results that are significantly better than weight tuning are marked with a dagger.

Similarly, we consider the 'concatenation' of path constrained walks and reranking. Table 7.5 includes the results of the combined approach, where reranking using graph walk describing features is denoted as 'Rrk$_{PCW}$', and reranking using the full feature set is denoted as 'Rrk$^*_{PCW}$.

Overall, the performance of the combined approach is better or comparable to applying reranking on top of the graph walks using random weights ('Rrk$_{G:R}$ and 'Rrk$^*_{G:R}$, in Table 7.4) or uniform weights ('Rrk$_{G:U}$ and 'Rrk$^*_{G:U}$, in Table 7.5). In both sets of experiments, the combined approach gives the best results for three of the six datasets. In the case of weight tuning and reranking combination, the combined approach significantly improves upon weight tuning for one of the datasets. For the path constrained walks and reranking combination, the pipeline approach is significantly better than the path constrained walks for the three person name disambiguation datasets.

Nevertheless, we also conclude from the results that reranking, given ranked lists of sufficient quality and adequate features, is relatively insensitive to small perturbations in the initially ranked lists.

7.2.3 PCW thresholding

In this section we are interested in evaluating the effect of path constrained walks thresholding on performance. Table 7.6 shows the performance the path constrained walks in terms of mean average precision for the thresholds of 0 (PCW:0), considering all the paths

Corpus	G:U	PCW:0	PCW:0.5	PCW:0.8
Person name disambiguation				
M.Game	0.65	0.65	0.74	0.00
Sager	0.67	0.65	0.65	0.00
Shapiro	0.71	0.76	0.79	0.00
Threading				
M.Game	0.53	0.65	0.64	0.10
Germany	0.55	0.76	0.73	0.00
Farmer	0.65	0.62	0.63	0.00
Alias finding				
Meetings	0.61	0.68	0.66	0.58
Personal	0.72	0.74	0.63	0.63

Table 7.6: A comparison of path constrained walks performance, for different thresholds (MAP).

in the corresponding path tree; 0.5 (PCW:0.5), following paths that lead to a majority of relevant nodes only; and 0.8 (PCW:0.8), following paths that lead to a strong majority of relevant nodes. We note that separate path tree were constructed per each corpus, using the training and developlment examples.

According to the results, a threshold of 0.5 is roughly comparable to a threshold of 0 for all datasets. A threshold of 0.8 captured very few to no paths using the person name disambiguation and threading describing path trees. High probability paths were included in the two path trees corresponding to the alias task, and a threshold of 0.8 yielded comparable or somewhat degraded performance for this task.

Figure 7.6 gives the results of applying path constrained walks with varying threshold to the task of city name extraction, using the MUC corpus, in terms of precision-recall curves. The figure gives also the performance of graph walks with uniform weights as reference. In this domain, the constrained graph walk paradigm dominates the graph walk. As discussed in Section **??**, there is greater importance to the edges traversed in the underlying graph of parsed text, whereas the graph walk embeds a perference to nodes that are proximate, due to the graph walk exponential decay over path length. This phenomenon accounts for the relative success of path constrained walks in this domain.

Comparing performance for different thresholds, as depicted in Figure **??** shows that applying higher threshold to the path constrained walk leads to improved performance in this case. It is also shown that higher threshold yield lower overall recall, as expected due to narrowing path coverage.

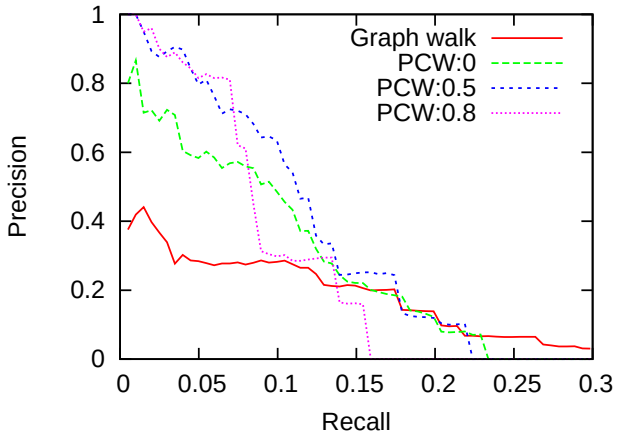


Figure 7.3: Precison-recall performance for city name extraction from the MUC corpus for path constrained walks with varying thresholds, and graph walks with uniform weights.

In general, our conclusion is that eliminating paths associated with low probability of reaching relevant target nodes from the path tree can often boost performance. The utility of applying a threshold of a particular value is domain-dependent, and should be tuned as a system parameter.

7.3 Scalability

In this section, we discuss the scalability aspects of the graph walk paradigm and give details regarding our graph walk implementation. In particular, we demonstrate the contribution of the path constrained graph walk approach to graph walk scalability. In addition, we review in this section the scalability aspects and implementation details of the proposed learning approaches.

7.3.1 Graph walks

In general, the paradigm of personalized Pagerank (and its variants) poses a major scalability challenge. While the PageRank algorithm [79] corresponds to a *single* special steady-state node distribution, which can be updated periodically, in personalized PageRank different distributions are associated with each possible query/preference vector V_q.

There are two distinct approaches for applying the personalized PageRank framework.

The first approach is to compute the personalized views at query time. This requires an iterative computation over the graph, where response time is linear with respect to the number of iterations and the number of edges traversed.

Another approach for implemeting personalized PageRank is an 'offline' computation, where personalized views are pre-processed and stored. Pre-processing all of the personal views (queries) possible is infeasible, due to time and space constraints. There are $O(2^n)$ and $\Omega(n^2)$ different queries possible for graphs with n vertices [42].

A variety of techniques have been discussed in the literature that address the computational aspects of personalized PageRank. The vast majority of the methods suggest efficient indexing of a reduced set of personalized PageRank vectors, trading some decrease in precision with significant savings of space and improved response times. We review these and other relevant methods in Section 7.4. Our contribution in the area of graph walk scalability is the path constrained graph walk approach. Earlier, we have shown that the path constrained walk often leads to improvements in performance compared with the unconstrained graph walks (see Chapters 5 and 6). In this section, we focus on the impact of the path constrained walk on scalability.

We next give details about our implementation of the graph walk, including empirical query processing time per our experimental corpora. We then proceed to discuss the effect of the path constrained walk on processing time and memory requirements.

Implementation details and running times.

We store the grpahs in secondary memory. In particular, we use the open-source database package Sleepycat [Sleepycat] to store the user-defined nodes and edges. In the experiments conducted we take the 'online' approach, where graph walks are conducted anew given each query.

Table 7.7 shows the average processing time per query for the tasks of person name disambiguation, threading and alias finding and multiple corpora, varying the walk length from $k = 2$ to $k = 8$. The results were obtained using a commodity PC with 4GB of RAM. The quoted execution times include the sorting of the graph walk output distribution. We note that the experiments reported in the table were not tightly controlled. That is, processing times may have flunctuated in the experiments given the machine load.

The sizes of the experimental corpora range from 6K to 14K nodes, and from 60K to about 200K edges (see Chapter 5). As expected, the processing times increase with the number of walk steps k. Processing times increase also with the number of graph edges (longer processing times are required for the Shapiro corpus compared with the smaller Management game corpus). The cardinality of the query distribution V_q affects processing time as well, as the number of nodes expanded in the graph walk may be multiplied by

Corpus	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
Person name disambiguation							
M.Game	1.1	0.9	2.5	4.4	6.5	8.4	10.1
Sager	0.4	1.9	5.0	8.1	11.8	14.9	18.6
Shapiro	4.2	3.0	7.4	11.5	15.5	21.4	25.0
Threading							
M.Game	0.1	0.7	1.8	3.4	6.1	7.4	9.7
Germany	1.1	1.6	3.9	7.7	12.4	16.8	21.4
Farmer	0.7	1.8	5.0	9.7	15.2	21.1	27.4
Alias finding							
Meetings	0.3	0.7	1.1	5.2	2.6	3.2	4.2
Personal	0.5	1.0	2.0	4.9	8.2	11.7	15.2

Table 7.7: Average query processing time per dataset and different walk length k.

this factor. Thus, processing times are longer for the two-node query in the case of person name disambiguation for any given k, compared with the single-node threading queries, both using the Management corpus.

The times given in Table 7.7 are satisfying for real-time applications. In particular, we have shown earlier in this chapter (Section 7.1.1 that short graph walks yield performance that is perferable or comparable to longer walks in this domain. Short walks of 2 or 3 steps require an average processing time of up to three seconds.[1]

The implementation scheme applied in our experiments can be readily improved by using better machinery and by common means such as distributed computing. We therefore expect processing times to be shorter than a single second for small to medium size corpora (as the ones included in Table 7.7) using well adapted systems.

The option of 'online' computation may be undesirable given larger graphs, in case that real-time response is required. Table 7.8 includes the average processing time of graph walks of length $k = 6$ for the city name and person name extraction tasks (in the column named 'Graph walks'). We remind the reader that the queries constructed per these tasks include each four graph nodes compared with a single query node in the case of threading, and two query nodes in the person name disambiguation task. As discussed above, this factor adds to the spreading of the graph walk.

Average processing times are stated in the table per the relatively compact MUC corpus (including about 80K nodes and 245K edges), and the larger MUC+AP corpus (containing

[1]The result for the Shapiro corpus and $k = 2$ seems to be an outlier.

Corpus	nodes [K]	edges [K]	Graph walk	PCW:0	PCW:0.5	PCW:0.8
City name extraction						
MUC	82	244	3.3	3.7	2.3	1.6
MUC+1/4AP	326	1,077	316.4	16.8	8.6	7.3
MUC+1/2AP	564	1,910	447.5	145.0	67.7	58.8
MUC+AP	1,030	3,550	6779.2	1993.0	1839.0	1881.6
Person name extraction						
MUC	82	244	1.5	0.5	0.4	0.2
MUC+1/4AP	326	1,077	4.9	2.4	1.9	0.7
MUC+1/2AP	564	1,910	161.1	4.3	3.1	1.0
MUC+AP	1,030	3,550	817.8	99.0	157.4	28.4

Table 7.8: Average query processing time for the named entity extraction tasks, using graph walks of 6 steps and corresponding path constrained graph walks with varying thresholds.

about 1M nodes and 1.7M edges). (Exact node and edge counts are detailed in Table 6.1.) While the MUC corpus is larger than the abovementioned email corpora, and the query distribution includes more nodes, query processing time is only 1.5 seconds on average for the person name extraction task and 3.3 seconds on average for the city name extraction task. The difference in processing times between the two tasks is due to a larger number of city name occurrences in the corpus. (Most of the person names included in the experimenal datasets correspond to only few mentions in the corpus.)

As shown in Table 7.8, the average query processing times for the MUC+AP corpus are very long. The steep increase in processing times, given the increase in the number of nodes visited, is due to a heavy load on the machine's memory management operations. **Say something more?**

Next we discuss how the path constrained walks contribute to scalability, susbstantially shortening processing times for large corpora.

Path constrained walks impact on scalability.

Constraining the graph walk to follow a path tree reduces the number of nodes (and edges) traversed in the walk, for several reasons. First and foremost, the path trees constructed in our experiments discard cyclic paths. In addition, represented in the path tree are the paths that correspond to the top nodes observed in the set of training examples, thus many other arbitrarily possible paths are eliminated. Finally, it is possible to apply a threshold on the probabilities associated with the path tree edges above which the graph walk is terminated, thus eliminating more (possibly frequent) paths.

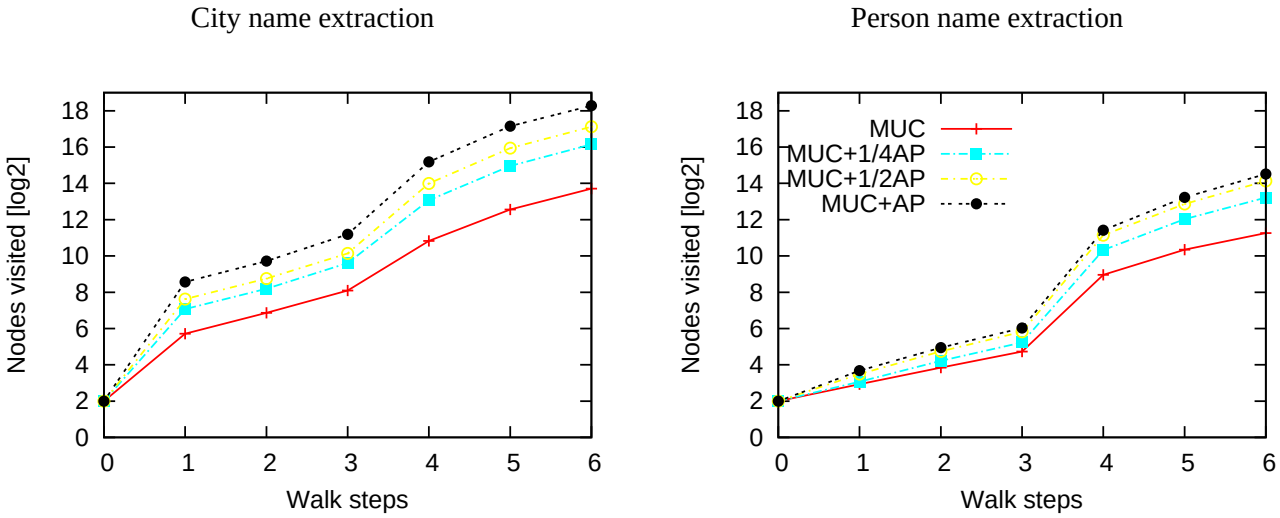


Figure 7.4: The cumulative number of nodes visited at each step of the graph walk, for the city name extraction and person name extraction datasets, for increasingly larger corpora.

The cumulative number of nodes visited at each step of the graph walk for the MUC and MUC+AP corpora are presented in Figure 7.4 (logarithm scale). The reported results refer to the MUC and MUC+AP corpora, as well as two intermediate corpora sizes, corresponding roughly to 25% of the MUC+AP corpus (*MUC+1/4AP*) and to 50% of MUC+AP corpus (*MUC+1/2AP*). According to the figure, the overall number of nodes visited increases roughly by a factor of 2, given a double sized corpus. (Especially if the MUC corpus, which is characterized by somewhat different named entity distribution is disregarded.) The figure also shows that the graph walks for the city name extraction task spread more in the graph, compared with person name extraction.

Table 7.8 shows the empirical average query processing times, applying the path constrained walk with no threshold (PCW:0); with a threshold of 0.5 (PCW:0.5); and with a high threshold (PCW:0.8) on these corpora. The average processing times required for the execution of unconstrained graph walk are given in the column named ‘Graph walk’. The reported figures indicate large savings in processing time due to the path constrained walks, compared with the unconstrained graph walks. Overall, the path constrained graph walks require less than 30% of query processing time compared with the unconstrained graph walk in the case of city name extraction, and less than 20% of the original procesing time for the person name extraction task.

Figure 7.5 provides another view of these scalability gains. The left part of the figure

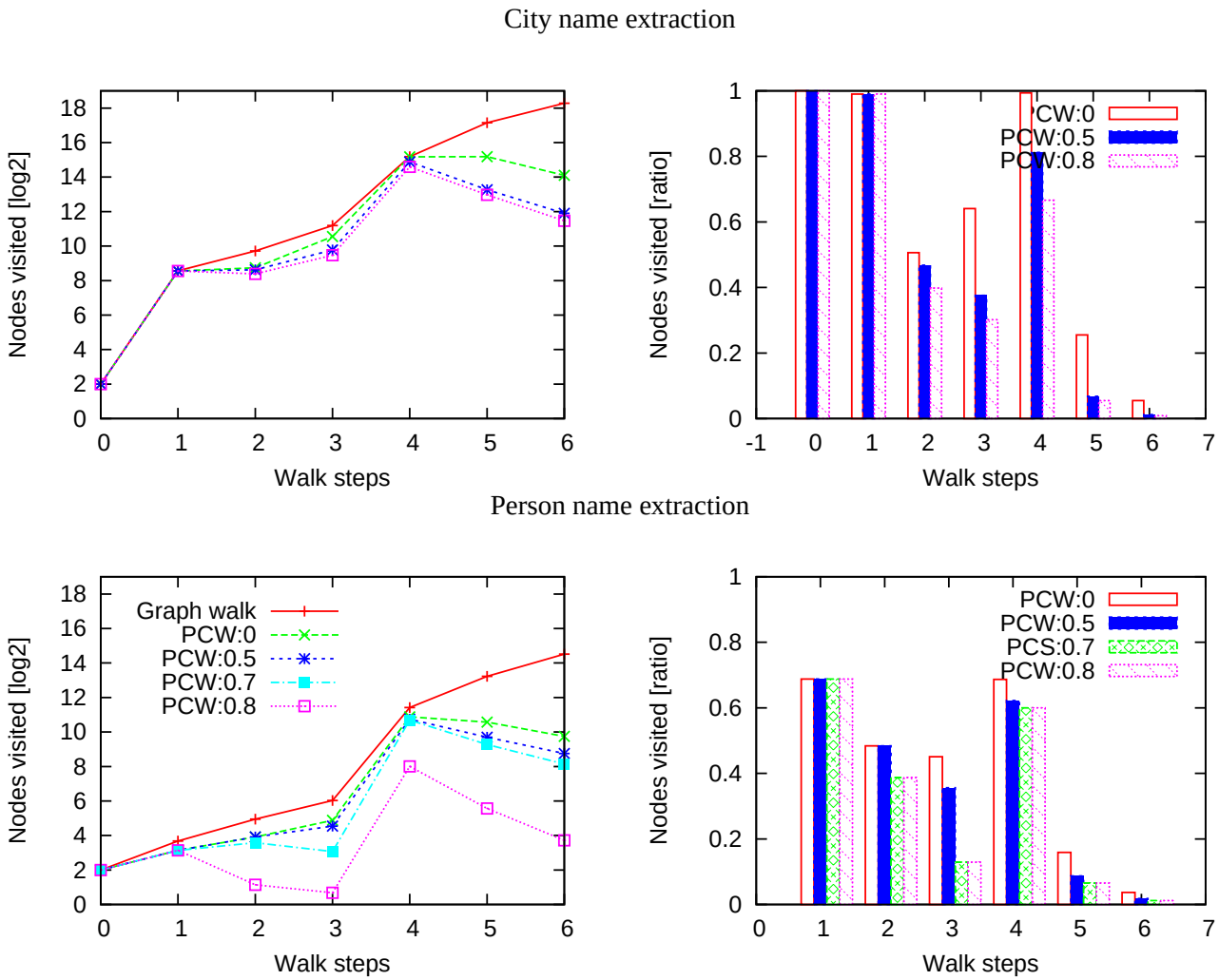


Figure 7.5: The cumulative number of nodes visited at each step of the graph walk (logarithm scale), for city and person name extraction and the MUC+AP corpus, using path-constrained walks with varying thresholds (left). The ratio between the number of nodes visited by the path-constrained walks relative to the unconstrained graph walks is presented in the right graphs.

shows the number of nodes visited (logarithmic scale) at each step of the walk, for a walk of $k = 6$ steps. It is clearly shown that, using the path constrained walk method, the number of nodes maintained at each step of the walk starts dropping at $k = 4$. (In practice, graph nodes that do not have an outgoing edge associated with a high probability edge in the path tree are discarded.) The graphs depicted on the right part of Figure 7.5 show the ratio of nodes visited by the constrained graph compared with the plain graph walk. **Double check why the decrease is not monotonic.**

Finally, we note that the path constrained walks require the maintenance of all combinations of a node and its unique histories (represented as graph node and path tree node pairs, see Section 4.4). However, the added memory requirements are negligible with respect to total savings.

7.3.2 Learning

This section is still under construction...

Reranking - deal with a large feature space; generation of the feature vectors.

Caching?

A major advantage of the path constrained walk, however, is that it is integrated with the graph walk paradigm, and does not require post-processing. In addition, reranking performance is affected by the quality of the ranked lists provided to it as input (a discussion about this issue follows in the next section), whereas the path constrained walk method directly affects the results.

7.4 Related Work

As mentioned in the previous section, the paradigm of personalized PageRank requires either power iteration computation given a query, which may translates to impractical response times, or pre-processing of a large set of 'personalized' distributions, where storing distribution per all of the possible node combinations is infeasible.

In order alleviate the scalability problem, one must therefore either exploit special features of the web graph or relax the exact problem to an approximate one [99]. This section reviews the efforts made in these direction, including techniques developed in the context of personalized PageRank, as well as some general relevant techniques.

Haveliwala [48] suggested the topic biased PageRank model, where his implementa-

tion of the model applied restricted personalization. That is, as part of offline preprocessing, a small number (16) of topic-sensitive PageRank vectors were generated. At query time, the similarity of the query to each of the pre-indexed topics was calculated. Then, instead of using a single global ranking vector, a linear combination of the topic-sensitive vectors, weighted using the similarities of the query to the topics, was considered.

Kamvar et-al [58] suggested the BlockRank model. This model was adapted to the Web and restricted personalization to hosts. The authors point out that the web link graph has a nested block structure: most hyperlinks link pages on a host to other pages on the same host. They exploit this structure by computing local PageRank scores within each host, and combining these local PageRank scores based on the importance of each host. In this work, the personalized pagerank model is modified such that rather than reset the walk to a biased distribution of webpages, a random surfer is assumed to be choosing hosts. The personalization vector therefore becomes a distribution over different hosts in this case.

Jeh and Widom [57] presented the *linearity theorem*, which proved to be a fundamental tool for scalable personalization. Informally, the linearity theorem states that the solution to a linear combination of preference (query) vectors u_1 and u_2 is the same linear combination of the corresponding personalized PageRank vectors (PPVs) v_1 and v_2. This means that if PPVs are available for some preference vectors, then PPVs can be easily computed for any combination of these vectors. Jeh and Widom therefore suggested to encode personalized views as partial vectors. In their work, the set of personalized vectors was restricted to a set of hub nodes H, selected as those more important for personalization. The size of H can be viewed as the available degree of personalization. Further, to compute a large number of hub vectors efficiently, the hub vectors are decomposed into partial vectors and a skeleton, components from which hub vectors can be constructed at query time. One partial vector is computed for each hub page p, which encodes the part of the hub vector unique to p. The complement to the partial vectors is the hubs skeleton, which captures the interrelationships among hub vectors. The authors present dynamic programming iterative algorithms for computing the partial vectors.

Balmin et-al presented the ObjectRank model [7], which was concerned with applying personalized PageRank to the setting of retrieval from relational datasets. The authors have pointed out that in this case it is not possible to define a set of hub nodes [57], since any node of the database may be included in a query. In the ObjectRank model, a personalized PageRank vector is therefore precomputed for each word in the corpus vocabulary. A few monotone score-combining functions for multi-word queries were suggested by the authors. They also proposed a method for reducing personalized PageRank vectors computation time for 'almost-acyclic' graphs. Further, to save cache space, the ObjectRank implementation truncates elements smaller than some threshold from the produced

vectors.

Fogaras et-al [42] were interested in their work in achieving full personalization, enabling online serving of personalization queries for any set of nodes (as opposed to Jeh and Widom [57], who restricted personalization to a set of hub nodes). They precompute *fingerprints* per each of the graph nodes, and store them in a database. A *fingerprint path* of a vertex u is defined as a random walk starting from u. The authors exploit the graph walk representation as a geometric distribution, i.e., after each step the walker takes a further step with probability $1-c$ and ends with probability c. A *fingerprint* of a node u is defined as the ending vertex of a fingerprint path of u. As a random variable, the fingerprint of u has the distribution of the personalized PageRank vector of u. The authors suggest a Monte Carlo algorithm to compute approximate values of personalized PageRank, where for each node u, N independent fingerprints are produced by simulating N independent random walks starting from u. The personalized PageRank vector for u is approximated with the distribution of the corresponding fingerprints, and indexed in a dataset. The output ranking is computed at query time from the indexed fingerprints using the linearity theorem. In order to increase the precision of the approximated vectors, the authors suggest to use the fingerprints generated for the neighbors of u (somewhat similarly to the dynamic programming approach suggested by Jeh and Widom [57]). The authors suggest also to sample finite graph walks. That is, instead of allowing very long fingerprint paths, they suggest to drop all fingerprints longer than length L.

Using sampling trades full personalization with precision. However, the authors show that a relatively small number of fingerprints allows to distinguish between the high, medium and low ranked nodes according to the personalized PageRank scores. (In particular, experiments conducted on 80 million webpages and $N = 1,000$ yielded good performance.) The order of the low ranked nodes is usually not as accurate using sampling. It is argued that PageRank itself was shown to be unstable around low ranked nodes, in the sense that a small perturbation of the graph can cause a very low ranked node to move to the middle of the ranking [65].

Recently, Chakrabarti [21] has suggested an algorithm named HubRank for computing personalized PageRank scores in entity-relation graphs, where edges are directed and typed. The algorithm indexes fingerprints (following Fogaras et-al, [42]) for a small fraction of nodes, chosen using query log statistics. According the proposed approach, only 'entity' nodes (where textual information is excluded) are pre-loaded, to form a skeleton of the graph. Given a keyword query, the query words are instantiated as nodes in the graph, which are linked to the entity nodes in which the words appear. A graph walk starting from the query nodes spreads over a small 'active sub-graph', which is bounded by the node distance (or, in our terminology – by the number of walk steps), and otherwise

by nodes for which the personalized PageRank score has been indexed. Once the active subgraph is set up, scored are propagated from the indexed nodes to the other nodes in the subgraph, using a dynamic programming computation [57]. In this work, elements in the fingerprint vectors smaller than a threshold are pruned, where experimental results have confirmed that this operation has minimal effect on accuracy. In summary, this method approximates the personalized PageRank vectors due to indexing of selected nodes only, sampling the graph walk and pruning the resultant personalized. In addition, limiting computation to a sub-graph implies that the grpah walk is approximated locally. Experimental results have shown this apporach to be preferable to ObjectRank [7] implementation in terms of pre-processing time, indexing space and online computation time.

The graph walk can be executed using implementations of matrix multiplication. The area of accelerating matrix multiplication is well studied, and there are various techniques available that reduce this opreration complexity and processing time.

Kamvar et-al [59] suggested the *quadratic extrapolation* technique that is applied periodically to enhance the convergence of PageRank. The authors claim that quadratic extrapolation eliminates the bottleneck for the Power Method, namely the second and third eigenvector components in the current iterate, thus boosting the effectiveness of the simple power method.

Sun et-al [95] suggest a matrix multiplication approximation for the personalized PageRank settings. They utilize the fact that real graphs are organized in a block-wise structure (i.e., communities). Using this property of the transition matrix, they propose to perform random walk with restart (i.e., personalized Pagerank) only on the partition of the graph that contains the query node; that is, output a local estimation of the personazlied Pagerank vector. Tong et-al [99] suggest an enhanced approach, which allows a global estimation of the personalized PageRank vectors. They consider both the block-wise structure property of the graph, as well as linear correlations across rows and columns of the adjacency matrix. In their apporach, low-rank approximation matrices are pre-computed and stored. Given a query, a few matrix-vector multiplication operations are required. Experimental results show that this approximation preserves high quality of the computed values, and improves the pre-computation and storage cost, as well as query processing times. **improves over what, plus – expand this para. a bit**

Cohen and Lewis [23] suggested a general algorithm for approximating large matrix multiplication. They propose a sampling algorithm that identifies high-values entries of matrix products, without full computation of the product. In their method, the expected values of the scores are equal to the true value that would be obtained with the full computation. The variance of the scores depends on the (relative) value of the entry, and decreases for high-value entries. That is, their algorithm returns exact scores for the top

ranked entries.

Simply put, in the suggested algorithm the matrices product is represented as a graph, where the edge weights of this graph are calculated backwards, measuring the impact of the multiplication represented by an edge on the end result. The multiplication graph is then sampled from, where each sample amounts to a random walk on the graph. According to the authors, this method is particulary effective for dense matrices. Otherwise, for sparse matrices, where only the top scoring instances are needed and exact values are not necessary, other methods are available, including: compressing inverted files, using lower precision arithmetic, ignoring some parts of inverted lists and limiting the number of document score accumulators maintained.

Appendix A

I like Pie

Yum! Pie is good! 3.1415926535897932384, I adore thee!

Bibliography

[1] Manu Aery and Sharma Chakravarthy. emailsift: Email classification based on structure and content. In *ICDM*, 2005. 5.5

[2] Alekh Agarwal and Soumen Chakrabarti. Learning random walks to rank nodes in graphs. In *ICML*, 2007. 4.6

[3] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. Learning to rank networked entities. In *KDD*, 2006. 1.2, 4, 4.1, 4.6, 7.2

[4] Shivani Agarwal. Ranking on graph data. In *ICML*, 2006. 4.6

[5] Kemafor Anyanwu, Angela Maduko, and Aamit Sheth. Semrank: Ranking complex relationship search results on the semantic web. In *WWW*, 2005. 1, 3.4

[6] Philip Ball. Prestige is factored into journal ratings. *Nature*, 439, 2006. 3.3.2

[7] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, 2004. 3.4, 7.4

[8] Krisztian Balog, Leif Azzopardi, and Maarten de Rijke. Formal models for expert finding in enterprise corpora. In *SIGIR*, 2006. 5.2, 5.5

[9] R. Bekkerman, A. McCallum, and G. Huang. Automatic categorization of email into folders: Benchmark experiments on enron and sri corpora. In *Technical Report, Computer Science department, IR-418*, 2004. 5.2, 5.5

[10] Ron Bekkerman, Ran El-Yaniv, and Andrew McCallum. Multi-way distributional clustering via pairwise interactions. In *ICML*, 2005. 5.5

[11] V. Bellotti and J. D. Thornton. Managing activities with TV-ACTA: Taskvista and activity-centered task assistant. In *Personal Information Management Workshop, SIGIR*, 2006. 5.5

[12] V. Bellotti, J. D. Thornton, A. Chin, D. J. Schiano, and N. Good. TV-ACTA: embedding an activity-centered interface for task management in email. In *CEAS*, 2007. 5.5

[13] Helmut Berger, Michael Dittenbach, and Dieter Merkl. An adaptive information retrieval system. based on associative networks. In *APCCM*, 2004. 4.6

[14] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, 2002. 3.3.2, 3.4

[15] Matthew W. Bilotti, Paul Ogilvie, Jamie Callan, and Eric Nyberg. Structured retrieval for question answering. In *SIGIR*, 2007. 6.3

[16] Razvan C. Bunescu and Raymond J. Mooney. A shortest path dependency kernel for relation extraction. In *HLT-EMNLP*, 2005. 6.5

[17] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullende. Learning to rank using gradient descent. In *ICML*, 2005. 4.1

[18] Vitor R. Carvalho and William W. Cohen. On the collective classification of email "speech acts". In *SIGIR*, 2005. 5.4.2, 5.5

[19] Vitor R. Carvalho and William W. Cohen. Preventing information leaks in email. In *SDM*, 2007. 5.5

[20] Vitor R. Carvalho and William W. Cohen. Ranking users for intelligent message addressing. In *ECIR*, 2008. 5.2, 5.5

[21] Soumen Chakrabarti. Dynamic personalized pagerank in entityrelation graphs. In *WWW*, 2007. 7.4

[22] Huan Chang and David Cohn. Learning to create customized authority lists. In *ICML*, 2000. 4.6

[23] Edith Cohen and David D. Lewis. Approximating matrix multiplication for pattern recognition tasks. *Journal of Algorithms*, 30(2), 1999. 7.4

[24] William W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18(3): 288–321, 2000. 1

[25] William W. Cohen and Einat Minkov. A graph-search framework for associating gene identifiers with documents. *BMC Bioinformatics*, 7(440), 2006. 1.2, 4.6

[26] William W. Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWEB*, 2003. 5.4.1, 5.4.3

[27] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research (JAIR)*, 10:243–270, 1999. 4.3, 4.6

[28] Michael Collins. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *ACL*, 2002. 4.3, 4.6

[29] Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69, 2005. 4.3, 4.5, 4.6

[30] Kevyn Collins-Thompson and Jamie Callan. Query expansion using random walk models. In *CIKM*, 2005. 1, 3.3.2, 6, 6.5

[31] Fabio Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6), 1997. 2.1.1

[32] Aron Culotta and Jeffery Sorensen. Dependency tree kernels for relation extraction. In *ACL*, 2004. 6.5

[33] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *LREC*, 2006. 6.3

[34] Christopher P. Diehl, Lise Getoor, and Galileo Namata. Name reference resolution in organizational email archives. In *SIAM*, 2006. 5.5

[35] Michelangelo Diligenti, Marco Gori, and Marco Maggini. Learning web page scores by error back-propagation. In *IJCAI*, 2005. 1.2, 4, 4.2, 7.2

[36] M. Dredze, T. Lau, and N. Kushmerick. Automatically classifying emails into activities. In *IUI*, 2006. 5.5

[37] Tamer Elsayed, Douglas W. Oard, , and Galileo Namata. Resolving personal names in email using context expansion. In *HLT-ACL*, 2008. 5.5

[38] Güneş Erkan and Dragomir R. Radev. Lexpagerank: Prestige in multi-document text summarization. In *EMNLP*, 2004. 6.5

[39] Gunes Erkan and Dragomir Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research (JAIR)*, 22, 2004. 3.3.2

[40] S. E. Fahlman. *NETL: A System for Representing and Using Real-World Knowledge*. MIT Press, Cambridge, MA, 1979. 2.1.1

[41] Christiane Fellbaum. *WordNet: An electronic lexical database*. MIT Press, 1998. 3.3.1, 3.3.2

[42] D. Fogaras, B. Rácz, K. Csalogány, , and T. Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3), 2005. 7.3.1, 7.4

[43] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3), 1999. 4.3

[44] W. Geyer, J. Vogel, L. Cheng, and M. Muller. Supporting activity-centric collaboration through peer-to-peer shared objects. In *ACM GROUP*, 2003. 5.5

[45] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity search in databases. In *VLDB*, 1998. 3.4

[46] Gregory Grefenstette. *Explorations in Automatic Thesaurus Discovery*. Kluwer Academic Publishers, Dordrecht, 1994. 6.4

[47] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD*, 2003. 1

[48] Taher H. Haveliwala. Topic-sensitive PageRank. In *WWW*, 2002. 3.4, 7.4

[49] Jingrui He, Mingjing Li, Hong-Jiang Zhang, Hanghang Tong, and Changshui Zhang. Manifold-ranking based image retrieval. In *MM*, 2004. 3.3.2

[50] Marti Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, 1992. 6.2

[51] Marti Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997. 1

[52] S. Henderson. Genre, task, topic and time: facets of personal digital document management. In *CHI*, 2005. 5.2, 5.5

[53] Ralph Holzer, Bradely Malin, and Latanya Sweeney. Email alias detection using social network analysis. In *LinkKDD*, 2005. 5, 5.5

[54] Paul Hsiung, Andrew Moore, Daniel Neill, and Jeff Schneider. Alias detection in link data sets. In *Proceedings of the International Conference on Intelligence Analysis*, May 2005. 5, 5.5

[55] Thad Hughes and Daniel Ramage. Lexical semantic relatedness with random graph walks. In *EMNLP*, 2007. 3.3.2, 6, 6.5

[56] Glen Jeh and Jennifer Widom. Simrank: A measure of structural-context similarity. In *SIGKDD*, 2002. 3.4

[57] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW*, 2003. 7.4

[58] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Exploiting the block structure of the web for computing. In *Stanford University Technical Report*, 2003. 7.4

[59] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating pagerank computations. In *www*, 2003. 7.4

[60] Edward Keenan and Bernard Comrie. Noun phrase accessibility and universal grammar. *Linguistic Inquiry*, 8, 1977. 6.4

[61] John Kleinberg. Authoritative sources in a hyperlinked environment. In *SODA*, 1998. 1, 4.6

[62] Brown Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *ECML*, 2004. 5.2, 5.3

[63] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *ICML*, 2002. 2.3, 3.4

[64] N. Kushmerick and T. Lau. Automated email activity management: an unsupervised learning approach. In *IUI*, 2005. 5.5

[65] R. Lempel and S. Moran. Rank stability and rank similarity of link-based web ranking algorithms in authority-connected graphs. *Information Retrieval*, 8(2), 2005. 7.4

[66] David E. Lewis and Kimberly A. Knowles. Threading electronic mail: A preliminary study. *Information Processing and Management*, 1997. 5.2, 5.5

[67] Dekang Lin. Automatic retrieval and clustering of similar words. In *COLING-ACL*, 1998. 6.4

[68] Bradely Malin, Edoardo M. Airoldi, and Kathleen M. Carley. A social network analysis model for name disambiguation in lists. *Journal of Computational and Mathematical Organization Theory*, 11(2), 2005. 5.5

[69] Andrew McCallum, Andres Corrada-Emmanuel, and Xuerui Wang. Topic and role discovery in social networks. In *IJCAI*, 2005. 5, 5.5

[70] Rada Mihalcea. Unsupervised large-vocabulary word sense disambiguation with graph-based algorithms for sequence data labeling. In *HLT/EMNLP*, 2005. 6.5

[71] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. In *HLT/EMNLP*, 2004. 6.5

[72] Einat Minkov and William W. Cohen. An email and meeting assistant using graph walks. In *CEAS*, 2006. 5.4.3, 5.4.4

[73] Einat Minkov, William W. Cohen, and Andrew Y. Ng. Contextual search and name disambiguation in email using graphs. In *SIGIR*, 2006. 1, 1.1, 1.2, 4.6

[74] Einat Minkov, Richard Wang, and William Cohen. Extracting personal names from emails: Applying named entity recognition to informal text. In *HLT-EMNLP*, 2005. 5.3, 5.4.1

[75] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7), 1994. 5.5

[76] Tom Mitchell, Sophie Wang, Yifen Huang, and Adam Cheyer. Extracting knowledge about users activities from raw workstation contents. In *AAAI*, 2006. 5.4.5, 5.5

[MUC6] MUC6. Proceedings of the sixth message understanding conference (muc-6). In *Morgan Kaufmann Publishers, Inc. Columbia, Maryland.*, 1995. 6.3

[77] Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, and Wei-Ying Ma. Object-level ranking: Bringing order to web objects. In *WWW*, 2005. 1.2, 4.6, 7.2

[78] Sebastian Padó and Mirella Lapata. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2), 2007. 6, 6.4, 6.4.2

[79] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. In *Technical Report, Computer Science department, Stanford University*, 1998. 1, 2.2, 3.3.1, 3.4, 7.1.2, 7.3.1

[80] Chris Pal and Andrew McCallum. Cc prediction with graphical models. In *CEAS*, 2006. 5.5

[81] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, 2004. 3.3.2, 3.4

[82] D. Petkova and W. B. Croft. Hierarchical language models for expert finding in enterprise corpora. In *ICTAI*, 2006. 5.2, 5.5

[83] M. R. Quillian. *Semantic memory. In M. Minsky (Ed.), Semantic information processing*. MIT Press, Cambridge, MA, 1968. 2.1.1

[84] Matthew Richardson and Pedro Domingos. The intelligent surfer: Probabilistic combination of link and content information in PageRank. In *NIPS*, 2002. 3.4

[85] D. Rumelhart, J. McClelland, and PDP Research Group. *Parallel Distributed Processing: exploration in the microstructure of cognition*. MIT Press, Cambridge, MA, 1986. 2.1.1

[86] Gerard Salton and Chris Buckley. On the use of spreading activation methods in automatic information retrieval. In *SIGIR*, 1988. 4.6

[87] Gerard Salton, Amit Singhal, Mandar Mitra, and Chris Buckley. Automatic text structuring and summarization. *Information Processing and Management*, 33(2): 193–208, 1997. 1

[88] Jacques Savoy. Bayesian inference networks and spreading activation in hypertext systems. *Information Processing and Management*, 28(3), 1992. 3.3.1

[89] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999. 4.3

[90] R. Segal and J. Kephart. Incremental learning in swiftfile. In *ICML*, 2000. 5.2, 5.5

[91] Sandip Sen. Developing an automated distributed meeting scheduler. *IEEE Expert*, 12(4), 1997. 5.5

[Sleepycat] Sleepycat. Sleepycat software. http://www.sleepycat.com. 7.3.1

[92] Henry Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24, 1973. 3.3.1

[93] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *NIPS*, 2005. 6, 6.1, 6.5

[94] John F. Sowa. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann Publishers, San Mateo, CA, 1991. 2.1

[95] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, 2005. 7.4

[96] Martin Szummer and Tommi Jaakkola. Clustering and efficient use of unlabeled examples. In *NIPS*, 2001. 2.3, 3.3.2, 3.4

[97] Egidio Terra and C. L. A. Clarke. Frequency estimates for statistical word similarity measures. In *NAACL*, 2003. 6.4.1

[98] Naftali Tishby and Noam Slonim. Data clustering by markovian relaxation and the information bottleneck method. In *NIPS*, 2000. 3.3.2

[99] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *ICDM*, 2006. 7.4

[100] Kristina Toutanova, Christopher D. Manning, and Andrew Y. Ng. Learning random walk models for inducing word dependency distributions. In *ICML*, 2004. 1, 3.3.2, 4, 4.6, 6, 6.5

[101] Ah Chung Tsoi, Gianni Morini, , Franco Scarselli, Markus Hagenbuchner, and Marco Maggini. Adaptive ranking of web pages. In *WWW*, 2003. 4.1, 4.6

[102] Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. What is the jeopardy model? a quasi-synchronous grammar for qa. In *EMNLP-CONLL*, 2007. 6.1

[103] Wensi Xi, Edward Allan Fox, Weiguo Patrick Fan, Benyu Zhang, Zheng Chen, Jun Yan, and Dong Zhuang. Simfusion: Measuring similarity using unified relationship matrix. In *SIGIR*, 2005. 1

[104] Y. Yang and C.G. Chute. An example-based mapping method for text classification and retrieval. *ACM Transactions on Information Systems*, 12(3), 1994. 1

[105] Jen-Yuan Yeh and Aaron Harnly. Email thread reassembly using similarity matching. In *CEAS*, 2006. 5.5

[106] Dengyong Zhou, Bernhard Scholkopf, and Thomas Hofmann. Semi-supervised learning on directed graphs. In *NIPS*, 2005. 3.4

[107] Dengyong Zhou, Jason Weston, Arthur Gretton, Olivier Bousquet, and Bernhard Schölkopf. Ranking on data manifolds. In *NIPS*, 2004. 3.4

[108] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, 2003. 3.4