Adaptive Graph Walk Based Similarity Measures in Entity-Relation Graphs

Einat Minkov

CMU-LTI-09-004

December 2008

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
William W. Cohen, Chair
Tom Mitchell
Christos Faloutsos
Raymond J. Mooney, University of Texas at Austin

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

Abstract

Relational or semi-structured data is naturally represented by a graph schema, where nodes denote entities and directed typed edges represent the relations between them. Such graphs are heterogeneous in the sense that they describe different types of objects and multiple types of links. For example, email data can be described in a graph that includes messages, persons, dates and other objects; in this graph, a message may be associated with a person with different relations, such as ”sent-to”, ”sent-from” and so on. In the past, researchers have suggested to apply random graph walks in order to elicit a measure of similarity between entities that are not directly connected in a graph. In this thesis, we suggest a general framework, in which different arbitrary queries (for instance, ”what persons are most related to this email message?”) are addressed using random walks. Naturally, there are many types of queries possible that correspond to various flavors of inter-entity similarity; several learning techniques are therefore suggested and evaluated that adapt the graph-walk based search to a query type.

The framework is applied in the thesis to two different domains. The first domain is personal information management, where it is shown how seemingly different tasks like alias finding, intelligent message threading and person name disambiguation, can be addressed uniformly as search queries using the adaptive graph-walk based similarity measure. The second domain evaluated is the processing of parsed text, where a graph represents corpora of structured parsed text, and adaptive graph walks are applied to induce inter-word similarity measures for tasks such as coordinate term extraction.

Finally, design and scalability considerations are discussed.

Acknowledgments

This thesis concludes a period of five years that I spent as a graduate student at CMU. At this point, I can only hope that my future holds as interesting, challenging and fun experiences as those that I had during these years. I got to meet many interesting people, both inside and outside the classroom, had the opportunity to attend many inspiring lectures and talks, traveled around the globe to participate in conferences, and got used to small robots running around in the corridors. It is true that along with these perks came a heavy workload; however, luckily, doing things that one loves is a great motivation. At CMU I've seen many people who are enthusiastic and creative about their research.

First and foremost in the list of people that I should thank for making these years great is my advisor, William Cohen. I am grateful to him for allowing me both the freedom and responsibility to follow directions that I liked, for letting me make mistakes and nudging me back on track when necessary. There is so much that can be learned from William, and I hope that I was able to acquire some fraction of it during these years. In addition to that, William is a very kind and supportive advisor, and that undoubtedly made a difference. Last but not least, the annual barbeques hosted by William and Susan were a lot of fun, and are to be remembered!

This thesis has been improved based on detailed feedback from the distinguished committee members: Tom Mitchell, Christos Faloutsos and Ray Mooney. I wish to thank them for their reviews, and for keeping their door open for any discussion.

During the graduate studies, I spent a very interesting summer at Microsoft Research in beautiful Seattle. My mentors at MSR, Kristina Toutanova and Hisami Suzuki, made it an engaging and productive experience. Other collaborators and friends from whom I got good advice and ideas are Anthony Tomasic, HangHang Tong, Kevyn Collins-Thompson, Oren Kurland, Benjamin Van Durme, Elchanan Mossel, Noah Smith and Andrew Ng.

As can be expected, my experience of CMU is made up of the people I shared it with, who have mostly already spread in many directions. My fellow CMU-ers, from first year to last, are Richard Wang, Vitor Carvalho and Yifen Huang, with whom I shared offices, travels, as well as some important life events. Ehud Halberstam, Anat Talmy, Oren Dobzinski, Katharina Probst, Guy Lebanon, Arye Kontorovich and Guy Zinman are dear friends; I am happy to have spent a chunk of life in Pittsburgh together with them, and look forward to crossing paths in the future. Ricardo Silva is a memorable friend, and I wish him best of luck in his continued quest of latent variables.

My partner and friend, Avshalom, gave me his unlimited support in anything and anytime, including in the period of thesis writing.

This thesis is dedicated to my late mother Tzila, who herself studied Computer Science when it was even less popular for women to do so, and who raised me to believe that I was special and capable. It is also dedicated with love to my father Leonid, who made sure to be very involved and supportive in every aspect in the course of these years, and to my dear sister Ortal.

Contents

List of Figures

List of Tables

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 [141], data integration [27], summarization [116] and document segmentation [60].

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 [56], the Semantic Web [6]; or email, where an email message is connected via header information to other emails and also to the recipient’s social network [95].

The famous algorithms of PageRank [102] and HITS [75] 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. An infinite 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, these algorithms discard by definition the initial distribution of the graph walk. An alternative line of research in this respect, which has also been used in IR, is *spreading activation*. As in the case of linked webpages, 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, constraints over paths and other constraints on activation flow had to be pre-set.

The focus of this thesis is a framework of *finite graph walks* over entity-relation structures. Nodes in the underlying graph are typed, and the directed edges are labeled with the relevant relations. Similarly to PageRank, we are interested in 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. We therefore adopt the paradigm of *Personalized PageRank* [102, 57], where we conduct finite graph walks. In Personalized PageRank, rather than let the surfer reset the graph walk randomly in the graph, the reset distribution is biased to a distribution of interest. If the reset distribution includes the starting points of the walk, then the probability of reaching nodes in the graph decays exponentially with their distance from the starting points. Thus, rather than modeling "centrality" of nodes, this type of a 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 resulting 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 we study several approaches to learning to better rank graph nodes for a given labeled examples: tuning the graph edge weights; 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; and a path constrained graph walk variant, in which high-level information about the usefulness of the paths traversed is used to guide the graph walk process.

Previously, Personalized PageRank graph walks over graphs have been used for estimating word dependency distributions [136]: 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 researchers have used graph walks over graphs for query expansion [140, 33] and other applications. In contrast with past works, in this thesis we are interested in a general graph representation of a given domain. We claim that if the graph is not especially engineered for a specific application, then various types of queries can be performed using the same underlying graph. We expect random walks to generate useful similarity measures given arbitrary queries, and apply learning to further adjust the graph-walk based similarity measure per task.

In the following, we describe the main components of this thesis (Sections 1.1–1.4), and outline it main contributions (Section 1.5).

1.1 Contextual Search and Disambiguation

Compared with ad-hoc traditional IR, the suggested framework has the advantage of representing various objects types. In contrast, common IR indexing methods, like the TF-IDF vector model, are strictly textual. Thus, depending on application design, *implicit* context may be easily incorporated in the graph walk search. For example, in email data, terms are linked to the message file in which they appear; if a user initiates a query while browsing an incoming message, then the node that represents that email message can be added to the initial distribution of the graph walk.

Why is such context-enriched (aka, contextual) search useful? the motivation for using context is two-fold. First, expanding a set of items with related information potentially increases the recall of a retrieval system. Second, incorporating context also assists in identifying the search results that are most relevant to the query settings, thus improving accuracy. For example, we have shown that graph walks are effective for tasks that involve entity disambiguation [95]. Consider a term that is known to be a personal name mention, like “Andrew”. We have shown that the graph-walk based framework is successful in mapping name mentions to the corresponding *person* nodes in a graph, by virtue of co-occurrence. However, name mentions in free text are often ambiguous; e.g., “Andrew” may refer to multiple persons that are included in one’s email collection, such as “Andrew Ng”, “Andrew McCallum” etc. In such ambiguous cases, starting the graph walk from both the term and the email file in which this term appeared improves the rankings of the relevant persons in the output distribution, due to the social network context provided by file node.

Another important advantage of the graph walk paradigm is that while traditional IR considers direct links between terms and files, a graph walk over multiple steps allows to reach items that are only indirectly linked to the query, i.e., via a longer chain of dependencies (considering the similarity of similar objects). This results in improved recall.

Finally, the representation of data as a graph of linked entities is relatively compact, and it allows using different contexts per demand efficiently.

The formulation of the suggested framework and a discussion of its properties are included in Chapter 2.

1.2 Learning

The described graph framework can be used for many types of queries, 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 [39, 100, 4]. This group of methods can be adjusted from infinite to finite graph walks. We have adapted an error backpropagation gradient descent algorithm [39] for tuning the graph weights in our framework.

A different approach that we suggest in this thesis learns to re-order an initial ranking, using features describing the edges in the traversed graph paths. This method parameterizes the graph walk with a set of representative features, and thus loses some information; however, unlike the graph walk and weight tuning, which can only consider local edge information, the features modeled can capture high-level properties of the graph walk. For example, 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. Another feature suggested is *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.

As mentioned, 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 reranker can use features derived from the full paths. However, for practical reasons, reranking processes only the top nodes retrieved in an initially ranked list, and its performance is affected by the quality of the initial ranker. it is therefore desired to incorporate high-level information already in the graph walk process. We suggest a path constrained graph walks variant, in which the graph walk process is guided by high-level information about path relevancy. In this approach, the edge weights in the graph are estimated dynamically, given the history of the walk.

The learning settings and methods are described in detail in Chapter 3.

1.3 Case Studies

The proposed framework is general whereas it is applicable to various domains. We investigate two different domains in this thesis.

One domain studied is Personal Information Management (PIM). We show that email data, as well as meeting entries and other objects from one's personal information organizer, can be represented as a joint graph, including entities like *persons*, *dates*, *email-addresses* etc. We find that unlike other methods, a graph walk over this network naturally integrates relevant textual and non-textual objects – that is, it combines text, recipient information and a timeline. We show that many useful email- and meeting-related applications can be phrased as search queries in the suggested framework. For example, we evaluate the tasks of *person name disambiguation*, *threading*, *finding meeting attendees* and *finding email aliases*. At least for some tasks, labeled data can be acquired automatically, facilitating the optimization of system performance via learning. We have evaluated each task individually, comparing it to viable baselines. The conducted experiments, using multiple corpora (including the public Enron corpus) show that the proposed framework performs favorably to other methods. Beyond the "hard" performance measures considered, we believe that the search framework in which ranked results are presented to the user is adequate for the purpose of personal information management. In particular, a human 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.

Chapter 4 provides an overview of personal information representation as a graph, and the related tasks. In addition to the above-mentioned tasks, we also define a set of activity-centered tasks, where the query consists of a *folder* that represents a user activity. The empirical evaluation of all tasks includes the graph walks in combination with the various learning methods, comparing them against relevant baselines.

A second domain that we apply our framework to is processing of parsed text. We suggest representing word mentions as nodes, and the syntactic structure that binds these words as labeled edges denoting inter-word relations. In particular, we consider dependency parse structures. A representation schema is suggested that unifies individual dependency structures derived per sentence from a large corpus (or several corpora) in a single graph. We apply graph walks and learning to adapt 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 a concept of interest. In particular, we show that the task of coordinate term extraction from parsed corpora can be phrased as a query in this framework, where the query defines a seed of relevant items. This task, which has drawn some interest in the research community in recent years, is useful for automatic or semi-automatic construction of knowledge bases. A detailed evaluation is given for city name and person name coordinate term extraction. It is shown that for small corpora, the graph-based approach gives better results than alternative vector-space methods, including a syntactic state-of-the-art model [101]. Additional experiments of general word synonymy extraction are described, and shown to give encouraging results.

The proposed graph representation of parsed text and the empirical evaluation of the related tasks are described in Chapter 5.

1.4 Implementation Considerations

Given the case studies of personal information management and parsed text processing, we are interested in drawing general conclusions regarding the design considerations of this framework.

Regarding the parameters involved in the graph walk process, we show that the graph walk length affects performance in general, and that short graph walks are preferable in some cases. In addition, we show that the choice of graph walk variant may affect performance as well.

Considering the set of learning approaches, we show that high-level information is useful in some problems. For example, path information is highly informative in the language domain, where local graph walks and weight tuning assign high weights to proximate yet irrelevant nodes. We also show that the path constrained walk approach is effective in eliminating irrelevant paths and improving performance. The path constrained walk paradigm involved additional cost in terms of runtime in our experiments; however, applying a threshold to the path constrained graph walk schema can improve both its accuracy and scalability.

Overall, processing times, given short walk length and medium-sized corpora (up to 100K nodes) are fast and appropriate for online settings, where the graph walks are computed 'on-the-fly'. Other algorithms developed that improve the scalability of the Personalized PageRank paradigm for larger graphs can be readily implemented in our framework.

The design and scalability aspects related to the proposed framework are discussed in Chapter 6.

1.5 Summary of Thesis Contributions

Following is a summary of the main contributions of this thesis.

- The thesis presents a general framework that uses finite random graph walks to generate an inter-entity similarity measure in the graph, where the generated similarity measure can be further adapted per task with learning.

- It is claimed and shown that given a general representation of structured and semi-structured data, multiple tasks can phrased as queries in the same underlying graph.

- The thesis presents the concept of learning to rank graph walk using global information, as opposed to the graph walk and known weight tuning methods that consider only local information.

- Reranking is applied as a learning approach for improving graph walks; and a set of generic task-independent global features is suggested.

- A method of path constrained walks is proposed, that incorporates global features in the graph walk process.

- The framework is applied to the personal information management domain, where a variety of tasks, some of which are novel, are addresses uniformly as queries; performance is shown to exceed alternative methods for some of the tasks evaluated.

- The framework is applied to the domain of parsed text processing, where it is shown to give superior results to state-of-the-art method for small corpora.

- Empirical evaluation shows that finite graph walks give better performance than infinite graph walks in some cases.

Chapter 2

Framework

In this chapter, we begin with the formalization of the graph schema and present a user interface, including a query language for using graph similarities (Section 2.1). We then define random graph walks as a technique of choice for evaluating the similarity of the graph nodes to a given query (Section 2.2). The applicability of the framework and the various motivations for using the framework are discussed in Chapter 2.3. Finally, in Section 2.4, we review research related to the framework in general, and to graph walks as a method for evaluating similarity in graphs in particular.

2.1 Definitions and Notation

This section defines relevant notations in terms of the underlying graph structure and the interface between a user and the system. In general, the graph is assumed to be directed, where typed nodes represent entities and labeled edges represent the relations between them. The interface defined consists of a query language that allows the user to search for similarity between any entities represented in the graph, where the response provided is in the form of a ranked list. Another component of the user-system interface is *user feedback* that includes judgments about which nodes are relevant to a query. Finally, we define the notion of *tasks*, relating to query classes, as various flavors of inter-node similarity may be sought in a single graph.

2.1.1 The Graph

A graph $G =< V,E >$ consists of a set of nodes V, and a set of labeled directed edges E. 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 node 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. This implies that the graph is cyclic and highly connected.

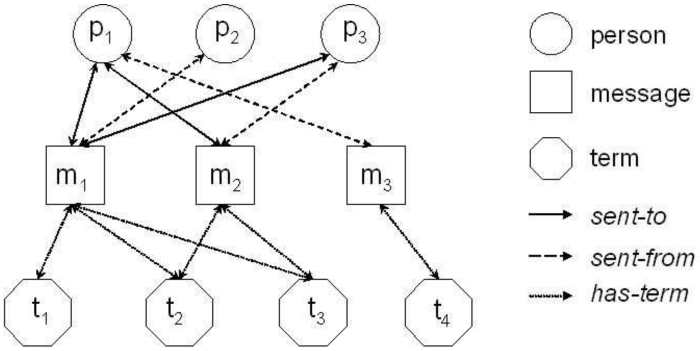


Figure 2.1: A simple example of the considered graph scheme

For example, consider the graph depicted in Figure 2.1. In the figure, node types are denoted by the different shapes of a circle, square and hexagon. The edges have different types as well, denoted by different line styles. Suppose that a circle represents a node 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 separate 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 the sender of that message. As shown in the figure, there may be multiple types of relations between the same types of nodes. For example, *email-messages* are connected to *person* nodes also over a relation of *sent-to*. This relation is denoted by solid edges in the figure.

2.1.2 Graph-based Similarity and Query Language

Given a graph that represents entities and their inter-entity relations, we are interested in inducing a general similarity measure between entities in the graph.[1] That is, we are interested in evaluating similarity between entities that are not directly connected, based on the information encoded in the graph. For example, according to the schema described in Figure 2.1, *messages* are not directly inter-connected in the graph; rather, *messages* are linked to the *terms* they contain, to *persons* who are recipients or the sender of the message, etc. As the graph represents a heterogeneous domain, containing multiple types of entities and relations, a good similarity measure should integrate multiple types of evidence into a single similarity score.

In this thesis, we take an information retrieval approach, where given a query, which is *any* combination of entities, a list of entities, ranked by their similarity to the query, is returned to a user. Formally, we define a query language, as follows.

Definition 1 *A* query *includes an initial distribution V_q over nodes, and a desired output type τ_{out}.*

Definition 2 *A* response *to a query $< V_q, \tau_{out} >$ is a ranked list of nodes z of type τ_{out}.*

Consider our running example. In the described domain, one may wish to find people that are related to a particular term, such as “learning”. In general, a term or a set of terms may represent a concept of interest, corresponding as a project name, or specialized key words. The relevant query in this case is $< V_q = \{t_2\}, \tau_{out} =$‘*person*’$>$ (where the term “learning” is represented by the graph node t_2).

2.1.3 Tasks and Feedback

Queries can be specified ad-hoc and arbitrarily by a user, according to the definition above. However, it is reasonable to expect that particular *query types* be executed frequently or on a regular basis in a given domain. We define *tasks* (or, query types), as follows.

[1] We use similarity and relatedness interchangeably.

Definition 3 *A* task *is a distinct relation r sought between a query distribution V_q and nodes of the specified type τ_{out} in the graph. Queries Q_1 and Q_2 are instances of the same task if V_{q_1} and V_{q_2} include nodes of the same types, and if both queries are to retrieve nodes of type $\tau_{out^1} = \tau_{out^2}$ that are related to V_{q_1} and V_{q_2}, respectively, with the same relation r.*

Notice that it is optional to specify the relevant task (r) along with a query $< V_q, \tau_{out} >$.

As examples of possible relations, consider the queries $< V_q = \{t =$"learning"$\}, \tau_{out} =$'*person*'$>$ and $< V_q = \{t =$"recruiting"$\}, \tau_{out} =$'*person*'$>$. These queries are instances of an association between a topic (represented by *terms*) and *persons*. A *terms-persons* mapping query can also reflect a different notion of entity relatedness. For example, in the query $< V_q = \{t =$"Bill"$\}, \tau_{out} =$'*person*'$>$, a user may be specifically interested in retrieving persons who are referred to by the name Bill. Similarly, a user may be interested in retrieving *messages* that are topically similar to a given *message*, or in recovering a *thread*, in which case messages that belong to the same chain of correspondence are sought.

In other words, we distinguish between different relations (*tasks*) in the graph, considering them as private cases of general inter-entity similarity (or, relatedness). A key feature of the proposed framework is that various queries, which are instances of different *tasks*, are to be performed using the same underlying graph.

It is expected that a *response* have a varying value that depends on the underlying task. Such value is therefore a function of the user's intentions. Next, we define the concept of *user feedback*.

Definition 4 User Feedback *includes the specification of correct (relevant) and incorrect (irrelevant) graph nodes per query.*

For example, consider the query mentioned above: $< V_q = \{t =$"Bill"$\}, \tau_{out} =$'*person*'$>$. In case that the underlying task is to find nodes in the graph that denote persons who are called "Bill", correct answers may include "William Scherlis", whereas "William Cohen" may be found to be an incorrect response by the user (as he does not use the nickname Bill), as well as "Einat Minkov" and other person nodes.

User labels, denoting node relevance, will be used to evaluate the quality of a response to a query, where a good response has the relevant nodes appear at the top of the ranked list, and irrelevant nodes – at lower ranks.

User feedback that both specifies the task and provides feedback about node relevancy will be used to learn a specialized similarity measure for that task. In general, it is a basic requirement of the proposed framework that a general ('default') similarity measure result in useful performance for arbitrary queries. Nevertheless, we are interested in enhancing that general measure to reflect a particular similarity flavor of interest, in cases where the task is specified.

Finally, we notice that a "user" may not be a human being, but a machine application, which conducts automatic information processing. For example, following the name resolution example, a human may be interested in resolving a name mention, while reading a message. It is possible that the interface be adapted to accommodate this task, in the form of a special button or menu option. In the case automatic email processing, the machine will first recognize person names in the text, and then use the querying mechanism to map each name mention to the corresponding person entity.

In the next section we describe how random graph walks are employed for extracting a general similarity measure from the graph. Section 2.3 discusses the scope of domains and applications that are considered by the framework, given its the graph walk properties. The adaptation of the graph walk based similarity measure to particular tasks given user feedback is the subject of Chapter 3.

2.2 Graph Walks

We adopt the *Personalized PageRank* random graph walk model as the paradigm of choice for producing a similarity measure between nodes in the graph. In this section, we first introduce the Personalized PageRank method, and review its relation to PageRank (Section 2.2.1). Probabilistic random graph walks require transition probabilities to be defined. In Section 2.2.2 we derive edge probabilities, where parametric edge weights that are a function of the type of the relation represented, together with graph topology, determine the probability of transitioning from a given node to its neighbors. Section 2.2.3 discusses several variants of Personalized PageRank that have been used in the literature. Finally, we discuss the properties of random graph walks in inducing similarity scores between graph nodes, that are shared to Personalized PageRank and variants 2.2.4.

2.2.1 Personalized PageRank

As a preliminary, we first introduce the general *PageRank* algorithm [102]. The PageRank model 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. This model can be viewed as a simple associative network, where nodes are of uniform type, and there is a single type of edges. The surfer behavior is modeled as follows: given that the surfer is at node (page) i, then with probability $\gamma \in (0,1)$ the user will "jump" (reset) randomly to some page in the network, and with probability $(1-\gamma)$ the surfer chooses to move to node j that has an outgoing link from i. 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, a random walk process is constructed as follows:

$$V_{d+1} = \gamma[\frac{1}{N}]_{1\times N} + (1-\gamma)\mathbf{M}V_d \tag{2.1}$$

where the total number of nodes (pages) is N, and $\mathbf{M}$ is a transition matrix, indexed by nodes. $\mathbf{M}$ distributes a node's probability uniformly among the pages it links to, i.e.

$$\mathbf{M}_{ij} = \begin{cases} \frac{1}{|ch(i)|} & \text{if there is an edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

where $ch(i)$ is the set of nodes that have an outgoing link from i (the children of i).

Under this model, the transition matrix is ergodic and has a unique stationary distribution V_* (i.e., V_d converges to V_*). The damping factor γ prevents the chain from getting stuck in small loops [16]. The *PageRank score* of node j, p_j, is defined as its probability in the stationary state V_*, giving a measure of document centrality in the network.

Node scores can be computed by repeating the following recursive formula until convergence:

$$p_j = \gamma\frac{1}{N} + (1-\gamma)\sum_{i\in pa(j)} \frac{p_i}{|ch(i)|} \tag{2.3}$$

where $pa(j)$ is the set of nodes that link to j.

The idea of biasing the PageRank computation for the purpose of personalization was first suggested in [102]. Other researchers have explored ways to bias the model to preserve an association between rankings and user preferences, or a query. The *Intelligent Surfer* model [112], for example, suggests that the surfer only follows links to pages whose content has been deemed relevant to a given query. Similarly, the distribution of PageRank's "jump" operation can be skewed to include only pages relevant to the query. A similar approach has been suggested for 'topic-sensitive' search [57], in which 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 graph walk paradigm defined as *Personalized PageRank* [102] is defined as follows:

$$V_{d+1} = \gamma V_0 + (1-\gamma)\mathbf{M}V_d \tag{2.4}$$

$$V_{d+1} = \gamma I V_d + (1-\gamma)\mathbf{M}V_d \tag{2.5}$$

where V_0 denotes a distribution of interest over the graph nodes. The Personalized PageRank scores are derived from the corresponding stationary state distribution. This formula of graph walk generalizes PageRank (Equation 2.1), in which V_0 is uniform.

It has been shown that the Personalized PageRank score for a target node z and a query node x equals a summation over all the paths between x and z (including cyclic paths, and paths that cross z multiple times), where paths are weighted by their probability [67, 48, 28]. Specifically, the Personalized PageRank probability $Q(z|x)$ of reaching z in an infinitely-long walk from x is also defined as:

$$Q(z|x) = \gamma \sum_{d=1}^{\infty} (1-\gamma)^d Q(x \xrightarrow{=d} z) \tag{2.6}$$

where $Q(x \xrightarrow{=d} z)$ is the probability of moving from x to z in exactly d steps, defined recursively as:

$$\begin{array}{l} Q(x \xrightarrow{=d} z) = \sum_y \Pr(x \longrightarrow y) \cdot Q(y \xrightarrow{=d-1} z) \\ Q(x \xrightarrow{=0} z) = 1 \text{ , if } x = z. \end{array}$$

The graph walk distributes probability mass from a start distribution over nodes through edges in the graph—incidentally accumulating evidence of similarity over multiple connecting paths. The reset probability γ applies an exponential decay over the length of the paths between x and a destination node z. In practice, this means that the infinite graph walk probabilities can be effectively approximated by limiting the summation to some maximal value k [136, 48, 28].

Example. As an illustrative example to the operation of the graph walk, consider a graph walk starting from node m_1, in the graph described in Figure 2.1. 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. The node m_2, for instance, would acquire probability mass due to the following set of connecting paths, after two time steps:

Node m_3 would acquire probability mass due to the paths:

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

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

2.2.2 Parameterized Edge Weights

The graph walk process (and accordingly, the similarity measure generated) is determined by graph topology.[2] In addition, the walk on the graph is controlled by a set of edge weight parameters Θ. Throughout the graph, edges of type ℓ are assigned an edge weight $\theta_\ell \in \Theta$. 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 (corresponding to the transition probability $\mathbf{M}_{x,y}$) is defined as:

$$Pr(x \longrightarrow y) = \frac{\sum_{\ell \in L_{xy}} \theta_\ell}{\sum_{y' \in ch(x)} \sum_{\ell' \in L_{xy'}} \theta_{\ell'}} \tag{2.7}$$

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. (PageRank schema of edge weighting given in Formula 2.2 is a special case, where the graph includes a single edge type, and weights are distributed uniformly.)

Continuing the example of the graph in Figure 2.1, 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 weights of these edges. For example, one arbitrary possible assignment of the parameter values Θ is the following:

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, and the edge weight parameters, as follows:

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

[2] The reset probability γ has negligible effect on the generated rankings; see a related discussion in Section 6.1.2.

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

The graph edge weights Θ can be set uniformly; randomly; manually, according to prior beliefs; or using a learning procedure, as discussed in Chapter 3.

2.2.3 Graph Walk Variants

In this thesis, we adopt Personalized PageRank as the graph walk mechanism for generating similarity scores. There are, however, other related variants of graph walks that have been used in the literature.

One 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 γ, which we call a "stay probability" [95]. The transition matrix in this case is defined as follows:

$$\mathbf{M}_{xy} = \begin{cases} (1-\gamma)Pr(x \longrightarrow y) & \text{if } x \neq y \\ \gamma & \text{if } x = y \end{cases} \tag{2.8}$$

A walk of k steps can be defined by finite matrix multiplication: specifically, if V_0 is some initial probability distribution over nodes, then the distribution after a k-step walk is $V_T = V_0\mathbf{M}^K$. 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 is interpreted as the similarity measure between x and z.

Another graph walk variant concerns how the probability mass associated with edges is distributed [7, 95, 28]. Rather than normalize the outgoing edge probabilities from each node according to their weights, another possible schema is the following:

$$Pr(x \longrightarrow y) = \sum_{\ell} Pr(x \xrightarrow{\ell} y|\ell) \cdot Pr(\ell|x) \tag{2.9}$$

In this schema, as a first stage, the random walker selects an edge type ℓ to be traversed, with probability $Pr(\ell|x)$. $Pr(\ell|x)$ is computed as the ratio between the parametric weight θ_ℓ, and the total outgoing probability mass from x. Let S_{τ_i} be the set of possible labels for an edge leaving a node of type τ_i, and let $S(x)$ denote the set of outgoing edge types that are present at the graph node x ($S_{\tau_i} \subseteq S(x)$); the total outgoing weight from x can be defined either as $\sum_{\ell' \in S_{\tau_i}} \theta'_{\ell} = 1$ [7, 95], or as $\sum_{\ell' \in S(x)} \theta'_{\ell}$ [28]. (In the first case, the considered probability distributions are deficient, as some valid outgoing edge type may not be included in $S(x)$ and thus can not be selected.)

Given the selected edge type ℓ, as a second stage a child node y is selected according to the probability $Pr(x \xrightarrow{\ell} y|\ell)$. In previous works, this probability has been defined to be uniform over the set of nodes connected to the parent node x with the given edge type ℓ.

In the next section we describe general properties that are common to all of the graph-walk variants.

2.2.4 Graph Walk Properties

Personalized PageRank and its variants have several inherent preferences that determine how probability mass is distributed from a query to the graph nodes. Many of the biases incorporated in the graph walk paradigm align well with the potential requirements of a similarity metric. Some preferences may be sub-optimal, however, depending on the task at hand. Following is a detailed discussion of these preferences.

- Personalized PageRank applies an exponential decay over path length (due to the reset 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, in the graph schema described in Figure 2.1, the email-message m_3 is likely to 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.

  A negative correlation between node distance and similarity is well established in graph theory (as discussed in Section 2.4.1). Naturally, in a large graph, nodes that are far away from a given node are less likely to be related to it than its neighbors. On the other hand, we notice that in some relational domains, it is possible that specific long chains of inter-entity relations are significant, so that a bias due to general proximity is less justified. This issue as raised in a case study, and our approach for resolving it using learning (namely, the path-constrained graph walks approach), will be discussed later in Sections 5.4.1 and 3.4 (respectively).

- 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, assume that edge weights are uniform, and $k = 2$. In this case, the person node p_1 will be considered more similar to the (uniformly distributed) query $V_q = \{t_1, t_2\}$ compared with p_2, since there are three paths connecting the query nodes to p_1:

$$\begin{array}{l} t_1 \overset{has-term-inverse}{\longrightarrow} m_1 \overset{sent-to}{\longrightarrow} p_1 \\ t_2 \overset{has-term-inverse}{\longrightarrow} m_1 \overset{sent-to}{\longrightarrow} p_1 \\ t_2 \overset{has-term-inverse}{\longrightarrow} m_2 \overset{sent-to}{\longrightarrow} p_2 \end{array}$$

whereas there are two paths leading to m_2:

$$\begin{array}{l} t_1 \overset{has-term-inverse}{\longrightarrow} m_1 \overset{sent-from}{\longrightarrow} p_2 \\ t_2 \overset{has-term-inverse}{\longrightarrow} m_1 \overset{sent-from}{\longrightarrow} p_2 \end{array}$$

- The edge label weights Θ provide a mechanism for affecting the probability flow in the graph. For example, suppose that the relation $\theta_{sent-from} > \theta_{sent-to}$. In this case, p_2 may be considered more similar than p_1 to the query $V_q = \{t_1, t_2\}$.

- The graph walk based similarity measure is asymmetric; that is, the weight (and rank) assigned in the final distribution V_k (for a graph walk of k steps) to a graph node z, given a query $V_q = \{x\}$ will not necessarily equal the weight (and rank) assigned to node x, by the inverse graph walk, starting from the query $V_q = \{z\}$. For instance, in the graph described in Figure 2.1, given the query $< V_q = \{t_1\}, \tau = term\} >$, a graph walk of $k = 2$ steps will assign equal weights to the nodes t_2 and t_3; however, given the query $< V_q = \{t_2\}, \tau = term\} >$, a higher weight will be assigned to t_3 compared with t_1 (as t_1 is linked to t_2 with a subset of the paths that link t_2 to t_3). Thus, an asymmetric structure of the graph is reflected in asymmetric inter-entity similarities.[3]

- The formulas for deriving edge probabilities (see Formulas 2.7 and 2.8) apply a weighting scheme that is similar to Inverse Document Frequency (IDF). Suppose that 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

[3] In addition, note that the edge labels Θ may be asymmetric (i.e., $\theta_\ell \neq \theta_{\ell-inverse}$).

“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 for that is that as shown in Figure 2.1, t_2 is mapped to two message nodes whereas t_1 is linked to a single message node. Hence, node connectivity has a similar effect as the use of an IDF weighting scheme.

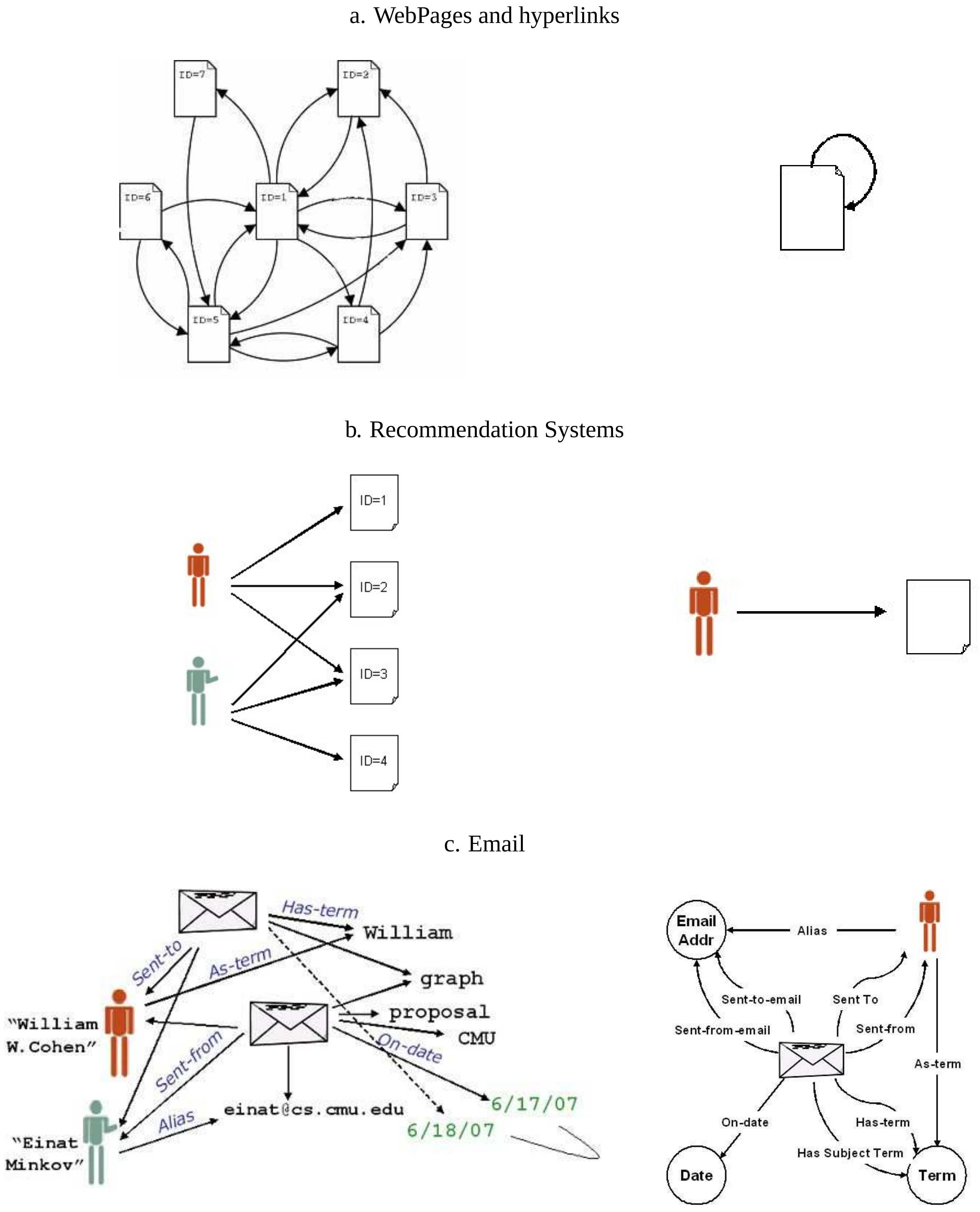
- Finally, as a consequence of probability accumulation at graph nodes throughout the graph walk, the resultant similarity measure is inclined towards high-degree nodes in the graph (as more paths are likely to cross nodes that are connected to many other nodes).

  For instance, in Figure 2.1, the person node p_1 is connected to all of the message nodes. This means that p_1 will be credited with probability mass via a larger number of connecting paths, compared with p_2 and p_3.

  Several researchers have pointed out previously that an inherent bias towards high-degree nodes is often not desired [13, 104, 80]. In general, this means that central nodes “take over” the graph walk, and disassociate the probability distribution in the graph from the query. Several solutions suggested to counteract such bias are reviewed in Section 2.4.1.

2.3 Applicability

So far we have defined the framework, including the graph formalism and the user-system interface. We have also described Personalized PageRank random graph walks as the technique of choice for evaluating node similarity for a given query. In this section, we define the intended scope of the framework, i.e., the types of problems that can be modeled (Section 2.3.1). In addition, we summarize the various motivations for applying this framework (Section 2.3.2). A main motivation that we stress in this thesis is *generality*; our goal is that a graph that represents the ‘natural’ structure of data can be used for a large number of different tasks, rather than having to engineer a graph carefully for each task.


a. WebPages and hyperlinks

b. Recommendation Systems

c. Email

Figure 2.2: Example graphs (left) and their corresponding graph schemas (right).

2.3.1 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 webpages and directed links represent the hyperlinks between them [117, 102] can be considered as a special case of the framework, where the graph includes a single type of nodes and a single type of edges, as shown in Figure 2.2(a). Similarly, scientific papers (or other publications) can be represented as a graph, where relationships between papers are inferred from their cross-citations [126], and so on.

Another generic type of graph that can readily be represented in this framework implements recommender systems. In this case, a user's preference for an item constitutes a relationship between the user and the item. This problem is naturally modeled as bi-partite graph, as demonstrated in Figure 2.2(b) [49]. The classical information retrieval settings, in which documents are indexed by the terms they contain, is another example of a bi-partite graph.

However, the framework is also adequate for richer, structured and semi-structured domains, in which the relation schema includes multiple types of nodes (objects) and various relations between them. For example, Figure 2.2(c) gives a representation schema for email data (this schema extends the toy example from Figure 2.1). 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, linked to the *messages* that contain it. Other examples of semi-structured domains include citation networks, where entities like *author*, *venue* and *publication title* are linked in a relational structure, and are also associated with text descriptions. An ontology such as WordNet [47] forms a network which include a single type of entity (words), but multiple types of edges, including *hypernym*, *hyponym* and so on. Later in this thesis, we consider a corpus of parsed text as a typed network, where nodes represent words, and edges denote syntactic relations (Chapter 5).

Representation Scope

Graph schema. Notice that the above-mentioned examples are characterized by well-defined graph schemas. (These schemas are described in Figure 2.2.) We assume in this thesis that for a given domain, there exists a closed set of entity types and the possible inter-relations between entities are pre-specified. Domains for which the full graph schema is not known in advance (such as, domains that evolve over time) may be modeled by extending the framework. For example, default edge weights θ_{def} may be defined for unknown edge types. This general case, however, is out of the scope of this thesis. Similarly, domains for which the graph schema is very large (e.g., the semantic Web), pose special challenges. We believe that at least some of techniques presented in this thesis will be beneficial in such cases. However, this remains a problem for future study.

Entity attributes. Entity-relation schemas, as used in database terminology, may include entity attributes. For example, an entity such as *employee* may be associated with attributes like *name*, *age*, *address* etc. (A specific entity is then associated with specific attribute values, e.g., *name:* "John", *age:* 42, etc.). In the proposed general graph schema, attribute values can be modeled as additional nodes, and special links can be added that represent relationships between an entity and an attribute value. It is also straight-forward to define such links to be *non-walkable*, and only use them for a *filtering* or other post-processing operations (similarly to "having" statement in SQL), so they do not affect the similarity metric.

Real values. Many graphs have been suggested in the literature that weight edges by real-valued weights, designating link importance, frequency, confidence, etc. For example, a social network that models inter-personal interactions may contain nodes representing *persons*; a single edge type denoting evidence of past *interaction* (e.g., correspondence) between person *x* and person *y*; and edge weights that designate the relative importance/strength of the interaction (e.g., the number of messages exchanged between *x* and *y*). In this thesis, we are interested in avoiding settings in which human judgments about feature relevance are encoded in the graph. Instead, we advocate relational graphs. The social network described above, for instance, can be modeled as a heterogeneous graph, where nodes denote *persons* and *interactions*; persons will be linked in this network via their shared interactions. In general, however, it is straight-forward to extend the framework to accommodate edge weighting. Specifically, formulas 2.7 and 2.8 can be modified such that the outgoing weight from a given node, per (θ_ℓ), be distributed among the edges of that type according to their relative individual values.

One may also be interested in representing real-value nodes (for example, for denoting attribute values, as described above). Real-values, however, are handled as discrete in the framework. This means that in case that the set of values is large, the size of the graph may be boosted. In addition, there is no trivial schema for linking nodes with proximate real values.

Undirected edges. Undirected edges can be trivially replaced with symmetric directed edges, in both directions.

n-ary relations. In general, inter-entity relationship may involve multiple entities. For instance, a relation like "hire" involves the person hired as well as the relevant position, and possibly, the agent responsible for the hiring action. The framework, which underlies this thesis, can not accommodate n-ary relations: rather, only relationships between pairs of entities can be modeled.

2.3.2 Types of Motivating Applications

There are multiple possible motivations for applying graph walks to derive an inter-entity similarity measure. In what follows, we discuss several motivating applications.

Associative retrieval. Graph walks implement a notion of transitive similarity. They can therefore be used as a mechanism for expanding or enriching a set of entities with related objects, in an *associative* manner. In the email domain, for example, a user is oftentimes interested in retrieving a specific item from her mailbox that she remembers. Searching by specifying strings included in the different fields of the message (as common in many commercial email management interfaces) may fail to retrieve the request item due to a possible mismatch between the terms as specified in the query and in the message. Suppose that the user is trying to track a *message* sent recently by William Scherlis, in which a meeting is coordinated. Searching for messages that include the strings "Bill" and "meeting" will fail in case that the message does not include the nickname "Bill" and the term "meeting" is not explicitly mentioned. Associative search using graph walks using the query $V_q = \{$*terms*="Bill, meeting"$\}$ is more likely to include the relevant message in the results, due to graph similarity between "Bill" and "William". (and if semantic relations between words are modeled, a term or expression that are related to a meeting may be reached as well.) Similarly, 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 4.

In the past, graph walks over a network that consists of inter-word semantic relations derived from WordNet [47] and other resources have been applied for query expansion in Information Retrieval [33]. Random walks over a similar graph have also been applied as a smoothing mechanism for the task of prepositional phrase attachment [136]. Automatic image captioning is another domain, in which graph walks have been applied to enrich textual descriptions of images with terms linked to related images [58, 105].

Structural similarity. A main motivation for representing data as a graph is to utilize 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 using random walks. For example, in a citation network, which includes co-citation links, applying random graph walks should identify clusters of papers that are mutually related; in a social network, graph walks can reveal sub-communities, and so on. The extent to which similarity gets 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 infinite (i.e., shorter than mixing time) to find clusters in data [129, 131]. The *Personalized PageRank* graph walk variant maintains high weight around the query nodes and decays fast thereafter. It reflects global structural similarity, however, in the sense that the graph topology affects the resultant similarity score distribution.

Evidence integration. As discussed above, graph walks accumulate evidence of similarity between query nodes and a target nodes, via multiple connecting paths. Thus, the probability score assigned to a target node as a consequence of the graph walk, summarizes various aspects of similarity. In the email domain, for instance, 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*).

The graph representation can readily integrate also multiple information sources. For example, organizational hierarchies can be added to email corpora, by adding links such as “reports-to” between an employee and her manager in the graph. Lexical networks are another example, in which various WordNet-based and other word-to-word relations are included in the graph [136, 33, 65].

Finally, the proposed framework offers the advantage of *Generality.* In the past, Personalized PageRank graph walks have been applied to carefully engineered graph schemas, constructed using information that is partial or pre-processed, with the goal of optimizing a particular task. For example, Toutanova et al [136] applied graph walks on a graph engineered to improve on the task of prepositional phrase attachment; and special graphs have been engineered for image captioning [58, 105]. In this thesis, we assume a graph schema that describes a given domain in a general and straight-forward fashion. i.e., avoiding addition or omission of certain entities or relations from the relational data structure. We claim that in a general graph, multiple tasks can be phrased in terms of inter-entity similarity in the graph. That is, we argue that graph walks can be applied as a *general-purpose* tool. Indeed, a general graph scheme may be sub-optimal for some tasks. Therefore, we consider learning, to optimize the graph-walk based similarity measure per task. In our study of the email problem (Chapter 4), we will show that graph walks yield good performance for a variety of email-related problems using default parameters, and improved results with learning.

2.4 Related Research

There are many research areas that are related to the framework that we apply in this thesis. In this section we give a short overview of some of these main areas, and try to point out the links between previous algorithms and observations made and this work. We first give a review of similarity measures prevalent in the area of graph theory (Section 2.4.1), including basic measures of graph similarity such the shortest-path or maximum-flow criteria, and also similarity metrics that are based on node immediate neighborhoods. Section 2.4.2 focuses on more recent work in graph theory that are closely related to random graph walks in general, and the Personalized PageRank graph walk variant in particular. Several of the algorithms are based on the electrical current analogue of random walks. Also included in Section 2.4.2 are works that generate a subgraph as response to a query, and graph-walk based algorithms for this purpose. In Section 2.4.3 we review previous research that is concerned with similarity in relational data, represented as a graph, where edges denoted entity relations. Several researchers have previously applied graph walks in these settings. We discuss the differences between these works and our approach.

In Section 2.4.4 we review several works in the area of learning, mostly for clustering and semi-supervised disambiguation, that apply a notion of structural similarity, and have strong connections to random graph walks.

Section 2.4.5 described the paradigm of *spreading activation*, a mechanism for propagating similarity between concepts in associative networks. We claim that the framework of this thesis (including its learning component) automates many manual design choices that are necessary in spreading activation.

Finally, in Section 2.4.6 we discuss the methodology of statistical relational learning (SRL), which has drawn much interest in the recent years, and is concerned with the modeling of structured information. As an example of SRL, we focus on Markov logic networks (MLNs). A short overview of the MLN approach is given, followed by a discussion of some of the main differences between MLNs and our framework.

2.4.1 Similarity Measures in Graph Theory

One simple graph proximity measure is the length of the shortest path connecting two nodes x and y, measured as the number of hops, or the as the sum of the edge weights along the shortest path. Another related concept from graph theory is *maximal network flow* [34]. Assigning a limited capacity to each edge (proportional to the edge's weight), this measure is defined as the maximal number of units that can be simultaneously delivered from x to

Shortest distance	(the negation of) the length of shortest path between x and y
Maximum flow	max. number of units that can be simultaneously delivered from x to y
Neighborhood measures	
Common neighbors	$\lvert\Gamma(x)\cap\Gamma(y)\rvert$
Jaccard's coefficient	$\lvert\frac{\Gamma(x)\cap\Gamma(y)}{\Gamma(x)\cup\Gamma(y)}\rvert$
Adamic Adar	$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\lvert log(\Gamma(z))\rvert}$

Table 2.1: Basic measures of node similarity in graph theory

y.

It has been argued that the shortest-path and max-flow similarity measures are not suitable for graphs representing phenomena such as social networks, for several reasons [46, 80]. First, the relationships between entities may be realized by multiple different paths; the shortest-path criterion, however, considers a single path by definition. Second, the maximum flow criterion is monotonic with the number of connecting paths, but disregards path lengths. In addition, it is desired that the proximity measure assign higher importance to edges between low-degree nodes, as these edges presumably indicate a more meaningful relationships. Both of the shortest-path and maximum-flow measures fail to capture these phenomena. Koren et-al [80] point out that maximal flow equals the capacity of the bottleneck of the flow between x and y, making such a measure less robust.

Additional measures of node proximity in graph theory, based on node neighborhoods, are included in Table 2.1. The most basic *node neighborhood* measure is computed as the overlap between node neighbors. That is, denoting the neighborhood of node x as $\Gamma(x)$, inter-node similarity is defined as $|\Gamma(x)\cap\Gamma(y)|$. The *Jaccard coefficient* [115] measures the probability that x and y have a common neighbor, for a randomly selected node from the union $|\Gamma(x)\cup\Gamma(y)|$. The related *Adamic-Adar* measure [1] considers a notion of frequency of the common neighbors, represented by neighborhood size. Liben-Nowell and Kleinberg [85] describe and empirically evaluate these and other related measures on the task of link prediction in social networks. They indicate that in the networks studied, between 71% and 83% of new edges form between pairs at distance three or greater. Since nodes at distance greater than two have no neighbors in common, this rules out the neighborhood-based methods, which are local, for link prediction tasks.

The *Katz measure* [72] is another metric, which defines node similarity as the number of their connecting paths, where path contribution is damped by length. It is calculated as follows:

$$Sim(x,y) = \sum_{\ell=1}^{\infty} \beta^{\ell} |paths^{\ell}_{x,y}|$$

where $paths^{\ell}$ is the set of connecting paths of length ℓ, and β is the damping factor.

Liben-Nowell and Kleinberg include the Katz measure in their comparative study [85], where it is shown to be among the best performing method. This measure is in fact related to random walks, which we describe next.

2.4.2 Graph-walk based similarity measures

In this section we describe proximity measures in networks that are associated with random graph walks. The methods are detailed in chronological order.

Hitting time. Consider a random walk initiated at node x, and iteratively moving to a neighbor of x chosen uniformly at random. The hitting time $H(x,y)$ is the expected number of steps required to reach y. (The corresponding similarity score is the negation of $H(x,y)$.) *Commute time* is a similar symmetric measure, defined as $C(x,y) = H(x,y) + H(y,x)$. Since the hitting time is generally small whenever y is a high-degree node, it has been suggested to modify it as follows: $H'(x,y) = H(x,y) \cdot \pi_y$, where π_y is the stationary probability of y [85].

SimRank [66] is a similarity measure adapted for directed graphs. In this model, objects are similar if they are related to similar objects, as follows:

$$Sim(x,y) = \gamma \times \frac{\sum_{a \in \Gamma(x)} \sum_{b \in \Gamma(y)} Sim(a,b)}{\Gamma(x)\Gamma(y)}$$

where $\gamma \in [0,1]$. A base case is that objects are similar to themselves (Sim(x,x)=1). 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. SimRank was shown to equal the expected value of γ^{ℓ}, where ℓ is a random variable giving the time at which two random surfers are expected to meet at the same node if they started at nodes x and y simultaneously and randomly walked the graph backwards. Hence, the SimRank measure is symmetric.

Effective conductance (EC) [104, 46]. Let G be a weighted undirected graph. The graph can be modeled as an electric circuit, where edge weights denote their conductance [41]. The proposed similarity measure is generated by setting the voltage of node x to 1, while grounding y (so its voltage is 0). Solving a system of linear equations gives the delivered current from x to y, called the *effective conductance*. In terms of random walks, EC is equivalent to the expected number of 'successful escapes' from x to y (*escape* is the event where y is reached by a random walk prior to re-visiting x), where the number of attempts equals the outgoing degree of x, denoted $deg(x)$. That is:

$$EC(x,y) = deg(x) \cdot P_{esc}(x \rightarrow y) = deg(y) \cdot P_{esc}(y \rightarrow x)$$

The escape probability decreases if long paths must be followed, and increases with the number of alternative paths. However, Palmer and Faloutsos [104] point out that this similarity measure is biased towards high degree nodes (as there is higher probability that a random walk will visit a high degree node at any given time). They therefore introduce a “universal sink” node that is grounded, and absorbs a positive proportion of the current that flows into any given node. This means that high degree nodes are heavily penalized, because each node is also “taxed” by its neighbors. In addition, grounding all of the graph nodes applies additional penalty on long paths, as after each step there is a certain probability that the walk will terminate in the universal sink.

Tong et-al [132] refer to the problem of high degree nodes, where they apply the Personalized PageRank graph walk paradigm. As an alternative to the universal sink, they propose to normalize the transition matrix **M**, as follows:

$$\mathbf{M}'_{x,z} = \frac{\mathbf{M}_{x,z}}{deg(x)^{\alpha}}$$

where z ranges over all of the graph nodes ($z = 1,...,N$), and the coefficient α is a free parameter. This formula applies a stronger penalty on high-degree nodes.

In a later work, Tong et-al [134] extend the electrical network similarity interpretation to directed graphs. While electric networks are inherently undirected, they suggest to generalize the effective conductance to handle directional information by using the escape probability. Escape probability can be computed as a function of the voltages at each nodes, as follows:

$$P_{esc}(x \rightarrow y) = \sum_{k=1}^{n} \mathbf{M}_{x,k} \cdot v_k(x,y)$$

where **M** is the transition matrix, n is the total number of nodes in the graph, and $v_k(x,y)$ is the generalized voltage at node k (where the voltage at x is 1, and y is grounded).

Cycle free effective conductance (CFEC). As discussed above, EC is interpreted in term of the escape probability, where the walk might backtrack and visit the same nodes many times. Koren et-al [80] point out that sending information in directions not leading to the target node y is a wasted effort, which cannot be fixed by a later backtracking. They suggest to consider instead cycle-free escape probabilities, which disallows paths from x to y where nodes are revisited. The Cycle Free Effective Conductance (CFEC) measure equals the sum of the simple (acyclic) connecting paths' probabilities, multiplied by the degree of the query node x, as follows:

$$CFEC(x,y) = deg(x) \cdot P_{cf.esc}(x \rightarrow y) = deg(y) \cdot P_{cf.esc}(y \rightarrow x)$$

where,

$$P_{cf.esc}(x \rightarrow y) = \sum_{R \in \mathcal{R}} Prob(R)$$

and $\mathcal{R}$ is the set of simple paths from x to y. The authors approximate the CFEC measure using the K most probable $x-y$ paths. They use an algorithm due to Katoh et-al [71], which generates paths of monotonically increasing length successively. Once the ratio between path probability and the probability of the most probable path falls below a threshold, further paths are discarded. Cycle-free effective conductance can naturally accommodate directed edges.

Connection Sub-graphs

In addition to evaluating node similarity in terms of probability scores, it has been suggested to present the user a small sub-graph, which explains the relationship between given nodes. Below is a short overview of recent research that extends graph walk (or, analogously, electrical flow) based similarity measures to a similarity sub-graph.

Faloutsos et-al [46] define a *connection subgraph* as a small subgraph (amenable to visual inspection) of a large graph that best captures the relationship between two nodes. They construct a subgraph that maximizes a goodness function, defined as the delivered current between the source node x to the destination node y, subject to a constraint on the number of nodes included in the subgraph. The flow captured in the subgraph equals the summation of delivered current over all the distinct 'downhill' (acyclic) paths from x to y included in the subgraph. A greedy algorithm optimizes the subgraph constructed, such that end-to-end paths are added iteratively, maximizing the ratio of flow along the path, divided by the number of new vertices that need to added to the output graph.

Tong et-al [132] defined the *center-piece subgraph (CEPS)* problem. This problem generalizes the connection subgraphs task, as it considers subgraphs that connect multiple query nodes. In addition, they allow different types of queries, including OR, AND, and K-softAND (where a sub-graph similar to at least K nodes out of the query nodes specified is searched) operations. The 'goodness criterion' of the subgraphs is based on Personalized PageRank scores in this case. In particular, the authors consider the *meeting probability*: the joint probability that random walks originating from multiple query nodes 'meet' at a target node in the steady-state. In the case of AND queries, the meeting probability considered is the multiplication of the individual PPR scores with respect to each query node. The OR operation requires the complimentary probability. For the computationally expensive case of K-softAND, a fast algorithm is provided. The subgraph produced is undirected and unweighted.

Koren et-al [80] use the cycle free effective conductance similarity measure to extract small subgraphs, which they call *proximity graphs*. Their motivation is to directly provide an explanation for a specific CFEC proximity value (see above). In particular, the *K* paths used for computing the proximity value serve as the building blocks of the connection subgraph. The subgraph extracted maximizes a ratio between the proximity value explained by the subgraph (raised to the power of a parameter α) and the number of vertices included. Solving this function is NP-hard, and heuristics (based on branch-and-bound algorithm) are suggested. The generated subgraphs can be directed.

2.4.3 Similarity in 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 first suggested to the best of our knowledge by Goldman et-al [54]. They suggested inter-object search, where the proximity used was the shortest path between objects.

BANKS [13] is a later model, suggested for keyword-based search in relational databases. In the BANKS framework, tuples are modeled as nodes in a graph, which 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. It is suggested 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 schema includes typed and weighted edges (reflecting link importance). The answer subtrees are ranked using a weighted combination of edge weights and node prestige, where prestige is defined as the node in-degree.

XRank [56] is a model that applies graph walks for keyword search queries over hyperlinked XML documents. In their model, the search can return nested XML elements that contain the desired keywords, rather than full documents. The authors' goal is to compute a measure an XML element's importance, based on the hyperlinked structure of XML documents. They suggest applying the PageRank model, where possible moves to neighboring nodes in the XML hierarchy are also considered by the random walk model (in addition to following hyperlinks between webpages).

The *ObjectRank* model [7] was the first to apply random walks – specifically, Personalized PageRank – to keyword search in relational data modeled as *typed* graphs. In ObjectRank, the graph edges are directed and typed; nodes are typed and associated with a set of keywords, derived from the attribute values of the represented tuple. As in previous works, for each edge in the graph, an inverse edge is added to the graph schema. The authors use an 'authority transfer' schema that is set manually, to determine the weight per edge type. (The schema is equivalent to the edge weight parameter Θ in our notation.) The authority transfer rate per each type is distributed uniformly among the outgoing edges of that type from each node. Given a query, Personalized PageRank graph walks are applied, where the reset operation is limited to graph nodes that include the query terms as keywords. The final node similarity scores are a combination of the latter keyword-specific scores, and global node scores, obtained using the PageRank approach. The authors evaluate ObjectRank using citation records. Our framework is very similar to ObjectRank. However, we allow querying the graph regardless of object types, whereas queries in ObjectRank (as well as XRank) are limited to terms. Accordingly, text is represented as regular nodes within the graph (see Chapter 4), rather than being processed separately. In addition, we are interested in optimizing the similarity measure induced by the graph walk for multiple different tasks.

Recently, several researchers have constructed special graphs, with typed edges and typed nodes, engineered to induce an improved similarity measure for a particular task, using graph walks. Pan et-al [105], for example, study the problem of automatic image captioning. They have applied Personalized PageRank graph walks to graphs that are undirected and unweighted, but include multiple types of nodes and several edge types. In particular, the graph constructed includes nodes representing *images*, *graphical regions* and *terms*. Nodes are linked due to structural links (image to its graphical regions of images, and image to its caption terms), or due to high graphical similarity. Others have constructed networks of word-to-word semantic relations to improve on the task of prepositional phrase attachment in natural language processing [136] and query expansion, in information retrieval [33]. In contrast to these works, the graphs that are the focus of this thesis are 'general-purpose' graphs, where data is represented as a graph with no target task pre-specified.

In another venue, it has been suggested to adapt the approach of connection graphs [46] (described earlier) to the relational domain [110]. The authors were interested in incorporating the semantics of different node and edge types in the considerations for selecting a response sub-graph. They used a hierarchical ontology of object classes and relationships, having each data object associated to relevant classes. Weights were assigned to nodes based on class specificity (where high specificity was preferred). The paths included in the sub-graph were weighted by parameters such as path rarity. In addition, paths that link instances of classes belonging to different schemas were considered more informative. This work is a nice example of a different approach for evaluating similarity in relational domains. However, the model suggested is somewhat arbitrary, while we are interested in adapting existing graph walk techniques to relational data. (In addition, sub-graphs construction as response to queries are not included in this thesis.)

2.4.4 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 random graph walks. In the semi-supervised clustering settings, 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.

Kondor and Lafferty [79] have proposed *heat diffusion kernels*, a class of kernels on graphs 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, in the sense that the continuous limit of heat diffusion kernels on a two-dimensional grid is a Gaussian kernel. It is also 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 x to point y, namely the sum of the probabilities that the lazy walk takes each path. Diffusion kernels are applicable only to undirected graphs, as a kernel function must be symmetric.

Szummer and Jaakkola [129] have applied Markov diffusion processes [131] 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 exponentiated and normalized to obtain transition probabilities.) This work applies *lazy* graph walks, as self-transitions back to each point are also included. A global similarity measure between arbitrary two points x and y is defined as the *diffusion probability* – the probability of transitioning from x to y in t time steps. (This measure is produced by using a matrix power, that is computing $\mathbf{M}^t$, where $\mathbf{M}$ is the transition matrix including self-loops.) The association of unlabelled points to the different labels is defined as the expectation over the diffusion probabilities to the labeled points. The authors discuss considerations of choosing the number of walk steps t. It is argued (similarly to Tishby and Slonim [131]) that clusters are formed for a finite walk length t, and that the clusters start dissipating as the graph walk converges. They claim that good choices of t for classification depend on the problem; for example, if labels change quickly over small distances, a smaller *t* provides a sharper representation. It is therefore proposed to choose *t* that maximizes the log likelihood of the data. It is also proposed that in case the graph has multiple connected components, individual *t*'s should be set for each component.

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 [146]. 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, this is interpreted as a walk starting from an unlabeled node, until the particle hits a labeled node. The measure used for classification is the probability that the particle, starting from node x, hits a labeled node with label 1. That is, 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 authors point out that using this formulation, there is no need to tune the walk length t (unlike Szummer and Jaakkola [129]). 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.

A regularization framework that forces the classification function to change slowly on densely linked subgraphs has also been suggested recently for *directed* graphs [144].

Zhou et-al [145] suggested to perform *ranking* using the intrinsic global manifold structure collectively revealed by a very large amount of data. They claim that 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 and normalized matrix, constructed using a local similarity metric. Positive scores are assigned to each query node. The query points then spread their score to their nearby neighbors via the weighted network. 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 equivalent to Personalized PageRank, where the ranking score of each query node is weighted according to its degree. They show that the suggested graph walks are equivalent to assembling all paths between two points, and weighting them by a decreasing factor.

2.4.5 Spreading Activation

The spreading activation (SA) Model is based on supposed mechanisms of human memory operations. Originating from psychological studies [109, 114], 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 [35]). 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* [45].

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 modeled 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:

1. **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.

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

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

4. **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 have to be often manually crafted as well, for a given domain.

The framework of Personalized PageRank and its variants, which gained much popularity in recent years (as described thus far) addresses some of these shortcomings. First, the inherent exponential decay over path length implements a soft *distance* constraint. In addition, the probabilistic graph walk limits the outgoing probability mass from a given node, such that nodes that have high out-degree distribute little probability to their individual descendants. This implements a soft version of the *fan-out* constraint. Further, path constraints and the importance (weights) of different link types in the network can be learned, rather then set manually, as will be discussed in the next chapter. Finally, while spreading activation required careful graph design, we consider using relational data, that is transformed to a graph in an automatic data-driven fashion.

2.4.6 Statistical Relational Learning

Statistical relational learning (SRL) concerns the induction of probabilistic knowledge for multi-relational structured data. Various paradigms of statistical relational learning have been proposed in recent years, including probabilistic relational models [51], Bayesian logic programs [74], relational dependency networks [99], Markov Logic Networks (MLNs) [111] and others. A general review of statistical relational learning is out of the scope of this thesis, and is available elsewhere [52].[4] In this section we give an overview of the Markov Logic Networks paradigm, an SRL model that generalizes finite first-order logic and Markov networks. We then discuss some of the differences between this paradigm and the graph walk framework.

It has been indicated that small variations in parameters can cause large variations in the models learned with ILP (e.g., Relational data min-ing with inductive logic programming for link discovery, Mooney et-al, 2002])

[4] SRL is closely related to inductive logic programming (ILP); on the application of ILP to relational learning, see for example Mooney et al [70].

Overview of Markov Logic Networks.

Markov logic combines first-order logic and Markov networks by attaching weights to first-order formulas and viewing them as templates for features of Markov networks. In general, MLNs allows softened first-order logic, where situations in which not all formulae are satisfied are considered less likely but not impossible.

Formally, a Markov network is a model for the joint distribution of a set of variables $X = (X_1, X_2, ..., X_n) \in \chi$. It is composed of an undirected graph G and a set of potential functions ϕ_k. The graph includes a node for each variable, and the model has a potential function for each clique in the graph, mapping the clique's state to a non-negative real value. The joint distribution represented by a Markov network is given by:

$$P(X = x) = \frac{1}{Z} \prod_{k} \left(\phi_k x_{\{k\}} \right)$$

where $x_{\{k\}}$ is the state of the k-th clique (comprised of the states of the variables that appear in that clique). The partition function Z, is a summation over all possible clique states.

Markov networks are often represented as log-linear models, with each clique potential replaced by an exponentiated weighted sum of features of the state. Markov logic assigns first order formulae (also called clauses or rules) as features. Let F be the set of all clauses in the MLN, w_i be the weight associated with clause $f_i \in F$, $G_{f_i} \in \{0,1\}$ be the set of possible groundings of clause f_i (1 if satisfied, and 0 otherwise), and Z be the normalizing constant. Then the probability of a particular truth assignment x to the variables in X is as follows:

$$\begin{aligned} P(X = x) &= \frac{1}{Z} \exp\left(\sum_{f_i \in F} w_i \sum_{g \in G_{f_i}} g(x) \right) \\ &= \frac{1}{Z} \exp\left(\sum_{f_i \in F} w_i n_i(x) \right) \end{aligned}$$

where $n_i(x) = \sum_{g \in G_{f_i}} g(x)$ is the number of groundings of f_i that are satisfied given the current truth assignments to the variables in X. This means that the fewer formulas a world violates, the more probable it is. The impact of each rule is determined by its associated weight.

The main inference task in MLNs involves finding the most probable state of the world given some evidence. In order to perform inference for an MLN, one needs to produce its corresponding ground Markov network. As described by Richardson and Domingos [111], this is done by including a node for every possible grounding of the predicates in the network and an edge between two nodes if they appear together in a grounding of a clause. Network grounding consumes memory exponential in the arity of the clauses. For example, Figure 2.3 (taken from [40]) shows the ground Markov network obtained by applying an MLN containing the formulas: $\forall x Smokes(x) \Rightarrow Cancer(x)$ and $\forall x \forall y Friends(x,y) \Rightarrow (Smokes(x) \Leftrightarrow Smokes(y))$ to the constants Anna(A) and Bob(B).

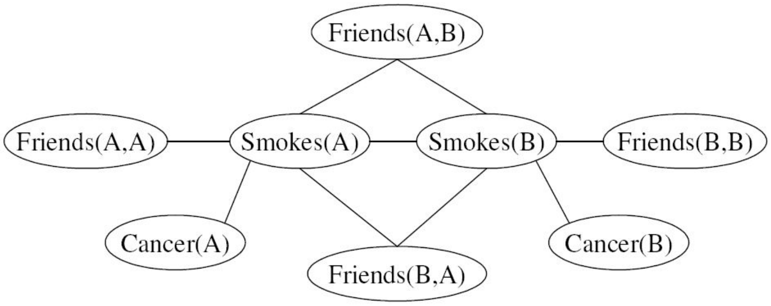


Figure 2.3: A ground Markov network obtained for two formulas of arity 2 and two constants

Exact inference in MLNs is intractable in general. Approximate inference algorithms include MCMC methods (like Gibbs sampling and simulated tempering). Weighted satisfiability solvers can also be readily applied in these settings (such as MaxWalkSAT, a weighted variant of the WalkSAT). MC-SAT [107] is a method that has been shown to outperform MCMC methods, as it employs WalkSAT to jump between regions of non-zero probability. In order to alleviate the memory requirements for propositionalizing the domain, a lazy version of WalkSAT has been suggested, which grounds atoms and clauses only as needed, taking advantage of the sparseness of the relational domain (e.g., most atoms are false) [125]. Using lazy WalkSAT has been shown to reduce memory usage by orders of magnitude. It is also possible to perform lifted first-order probabilistic inference in Markov logic [40, 15].

Learning an MLN includes two components: setting the weight of each clause, and learning the logical clauses (structure learning). There are two approaches to weight learning in MLNs: generative, and discriminative. In generative learning, the goal is to maximize the likelihood of the data. Running inference in every iteration, however, is too expensive. A more efficient approach is to maximize the pseudo-likelihood of the data, and its gradient, estimated using the Markov blanket [111]. The pseudo-likelihood parameters may, however, lead to poor results when long chains of inference are required. Discriminative learning can be applied in cases where the query predicates are pre-specified and the goal is to correctly predict the latter given the evidence variables. The voted Perceptron algorithm for discriminatively learning hidden Markov models has been adapted to Markov logic simply by replacing the Viterbi algorithm with MaxWalkSAT, for finding the MAP state (the most probable state of the query predicate given the evidence) [124]. A state-of-the-art discriminative weight learner is preconditioner scaled conjugate gradient (PSCG) [87], which uses samples from MC-SAT to approximate the expected counts of satisfied clauses for a given model, feeding them into the gradient and Hessian of the conditional log-likelihood of an MLN.

Regarding structure learning, Kok and Domingos [77] have suggested to apply beam search or shortest-first search over the set of clauses. Mihalkova and Mooney [93] suggested to first construct Markov network templates from the data and then generate candidate clauses from these network templates, adding greedily to the final MLN. Recently, Huynh and Mooney [138] proposed a discriminative approach for constructing MLNs, in cases where the target and evidence predicates are pre-specified. More specifically, they use a variant of an existing ILP system (ALEPH) to construct a large number of potential clauses and then learn their parameters by altering existing discriminative MLN weight-learning methods to utilize exact inference and L1 regularization.

Markov logic has been applied to problems in entity resolution, link prediction, information extraction and others, and is the basis of the open-source Alchemy system [78].

Discussion

The information encoded in the graph can be represented as Markov logic networks. In particular, the direct inter-entity relations represented by the graph edges correspond to evidence predicates in MLNs (e.g., *sent-from*(x,y)). Long range associations between entities can be modeled in MLNs as rules. (For example, consider the rule: $\forall x \forall y$ *has-term*(x,y)$\wedge$ *sent-to*$(y,z) \Rightarrow$ *related*(x,z). There are, however, several crucial differences between the graph walk paradigm and MLNs.

First, as mentioned above, Markov network grounding requires memory exponential in the arity of the clauses. Even with binary clauses, having a large number of constants can result in several million clauses. In the graph-walk framework, in contrast, the graph constructed is compact, as every entity (constant) corresponds to single node in the graph and the entity interactions are represented by the graph edges. The scalability challenge of network grounding is partially addressed by the inference algorithm of lazy MC-SAT;

however, its efficiency varies for different networks.

Another difference between the approaches is that MLNs require the rules relevant to the domain and problem of interest to be specified in advance. (This requirement is related to the scalability bottleneck, as specifying all possible relations is infeasible.) Currently, manually designing an MLN requires some expertise. Automatic structure learning is an active research area, which has not yet reached maturity. In contrast, learning task-specific structures (rules) is not pre-requisite in our framework. Notice also that since the relevant rules are different for each task, a different network is instantiated in MLNs in each case. The graph framework, on the other hand, does not encode task-specific information in the graph, so that the same graph is used for different tasks.

The expressive power of MLNs is larger compared with our framework, since it can also model any n-ary relations; the graph representation, on the other hand, only represents binary relations. On the other hand, the notion of structural similarity is not as well represented in MLNs. For example, high-degree nodes distribute their probability over many children nodes in the graph, thus achieving an IDF-like effect. In MLNs, this phenomenon is not inherently modeled.

An empirical evaluation of our framework and Markov logic networks for a small subset of the problems that are evaluated later on in this thesis is included in Appendix C.

2.5 Summary

In this chapter, we introduced a framework for extracting a similarity measure from structured and semi-structured data. We represent data as directed labeled graph, where nodes denote entities and typed edges represent the relations between them. We take a ranking approach, where a query includes a distribution over entities (graph nodes), and specifies the type of entity to be retrieved. A response is a list of entities of the requested type, ranked by their similarity to the query. As a mechanism for inducing the similarity measure between nodes, we adopt the Personalized PageRank algorithm, where we apply finite graph walks. The graph walk propagates similarity to a start node through edges in the graph—incidentally accumulating evidence of similarity over multiple connecting paths. While we apply Personalized PageRank, other graph-walk based variants or algorithms with similar properties could be used as well.

In this thesis, we focus on a graph that naturally models a structured dataset (like an email corpus). Representing data in full, and avoiding special data and feature engineering, allows to process many different classes of queries using the same underlying graph. We further defined a *task* as a query class as a particular inter-entity relationship in the graph. While we expect the basic similarity measure (Personalized PageRank, or another base measure) to be effective in the general case, we are interested in exploring learning to improve the results of the graph walk for specific tasks. This is the topic of Chapter 3.

Chapter 3

Learning

In the framework, as defined in Chapter 2, multiple tasks can be addressed as queries using the same graph. A *task* (defined previously) refers to 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 2.2(c). Concrete examples of tasks in this domain include *Alias finding*, where a user is interested in retrieving *email-address* nodes that are associated with (that is, belong to) a specific person. *Threading* is a different task in this domain, in which given a particular *message*, the goal is to recover messages that belong to the same thread. Another example is the *person name disambiguation* task, where given a person name mention in a message, the corresponding *person* node is sought. These and other generic tasks in the extended domain of personal information management are presented and evaluated in the next chapter. Overall, the number of different tasks possible is very large.

A general similarity measure, such as the Personal PageRank graph-walk based metric, can be applied to different tasks, and produce similarity scores that reflect structural information in the graph. It is reasonable, however, that different similarity notions imply varying importance for different link types. In other words, 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 between the query and target nodes. In this chapter, we therefore consider the problem of *learning* how to better rank graph nodes for a given task.

The learning approaches discussed include tuning of the edge weight parameters Θ where several methods exist that learn the edge weight parameters to optimize graph performance [136, 39, 4]. Following prior work, we adapt an error backpropagation approach

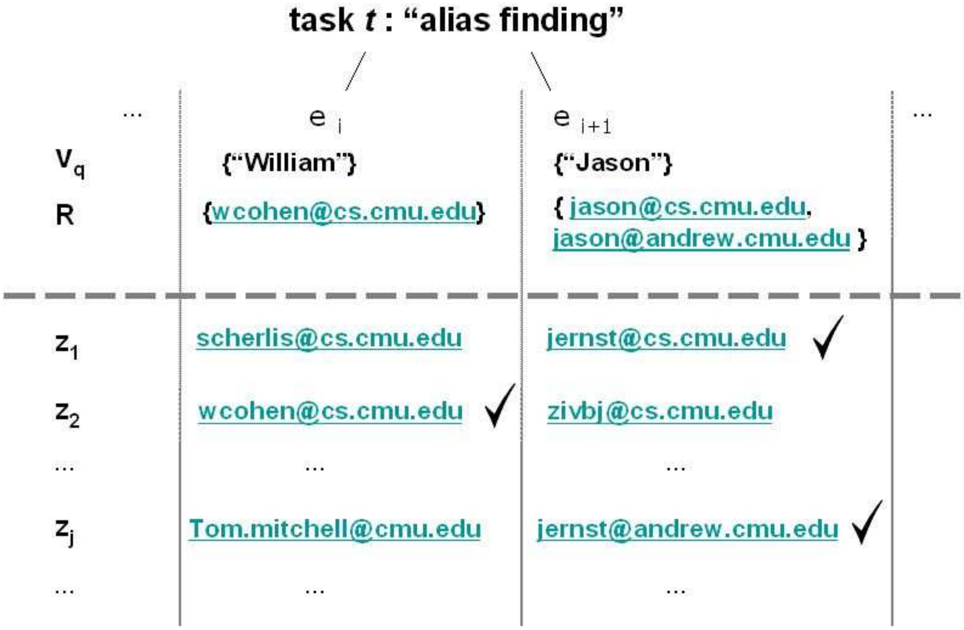
[39] to our settings. These weight tuning methods, however, are local in the sense that they decompose the graph walk into discrete time steps (i.e., they ignore the walk history). We present the concept of *global learning* to improve graph walk performance for a given task. In particular, we apply reranking in this framework. Given ranked lists generated by a base similarity metric (such as Personalized PageRank), we suggest to re-order these lists using predefined features that describe node properties. The feature set can describe global properties of the graph walk, such as the sequences of edge types traversed in the route from the source query nodes to a target node. Since reranking post-processes initially ranked lists, it can be readily combined with weight tuning in a pipeline fashion. In addition to reranking, we also suggest a *path-constrained* graph walk variant as a method for specializing the graph-based similarity measure to a task. In this graph walk variant, our goal is to incorporate global information about the graph walk into the graph walk process. Rather than model the graph walk as a Markovian process, we require that the random walkers 'remember' the paths they traversed at each step of the walk, and consider varying edge transition probabilities, depending on the walk history.

In the rest of this chapter we first describe the learning problem and settings (Section 3.1). In Section 3.2 we describe the error backpropagation algorithm for tuning the edge weights, adapting it to finite Personalized PageRank graph walks. The reranking schema, and a set of generic features used to describe graph nodes in terms of the graph walk, are described in Section 3.3.

3.1 Learning Settings

In this thesis, we consider supervised learning settings. That is, it is assumed that labeled example queries e_i are provided ($1 \leq i \leq N$) that are instances of a task t of interest. Each example query specifies a different distribution over nodes V_q^i, but the same user intention (task t) is assumed in all example queries. In case that the focus task is *alias finding*, for instance, example queries may include $V_q^i = \{term\text{=“William”}\}$, $V_q^{i+1} = \{term\text{=“Jason”}\}$ and so forth.

Example labeling schema. Several labeling schemas have been suggested in the area of learning to rank, including absolute scores, where target node probabilities are specified [137]; ordinal information, where ordinal values are assigned to nodes that represent their relative relevancy to the example query [18]; and pairwise node preferences, sampled from initially ranked lists [4]. In this thesis, we consider a binary labeling scheme, where the complete set of nodes that are considered as relevant answers to an example query e_i, denoted as R_i, is provided. (We will assume that graph nodes that are not explicitly



task *t* : “alias finding”

	…	e $_{i}$	e $_{i+1}$	…
V_q		{“William”}	{“Jason”}	
R		{wcohen@cs.cmu.edu}	{ jason@cs.cmu.edu, jason@andrew.cmu.edu }	
z_1		scherlis@cs.cmu.edu	jernst@cs.cmu.edu ✓	
z_2		wcohen@cs.cmu.edu ✓	zivbj@cs.cmu.edu	
…		…	…	
z_j		Tom.mitchell@cmu.edu	jernst@andrew.cmu.edu ✓	
…		…	…	

Figure 3.1: A dataset, generated using initial rankings per labeled examples for the task of alias finding. In this task, the queries include *term* nodes, and nodes retrieved are of type *email-address*. Relevant answers for query e_i (marked by a checkmark) are the nodes specified in R_i.

included in R_i are irrelevant to e_i.) This labeling schema is adequate for well defined problems, in which a query corresponds to a finite set of “correct answers”, and other nodes are considered irrelevant.

Initial rankings. Given are the graph G, the graph walk parameters (walk length k and reset probability γ), and some initial graph edge weight parameters Θ^0. We apply graph walks using these graph parameters to generate a ranked list of graph nodes, for every example query e_i. The corresponding output ranked list generated per example e_i is denoted as l_i^0. Henceforth, z_{ij} will denote the output node ranked at rank j in the ranked list l_i, and $p_{z_{ij}}$ will denote the score assigned to z_{ij} in the ranked list.

Learning goal. Learning is aimed at improving the initial rankings l_i^0, such that nodes known to be relevant, $z_{ij} \in R_i$, are ranked higher than the irrelevant nodes ($j_{rel} < j_{irrel}$) for every node pair in the output ranking l_i; that is, we are interested in producing modified lists l_i, in which the relevant nodes occupy the top ranks. As is the case with learning in general, it is expected that the learned models generalize and improve the rankings of unseen instances. These instances may correspond to the same graph that the labeled examples refer to, or different graphs in the same domain, i.e., other graphs that adhere to the same graph schema.

Figure 3.1 provides a graphical illustration of a dataset (a set of labeled examples) for the task of *alias finding*. Every example e_i includes a query V_q^i specifying a person's first name, represented as a *term* in the graph. The goal of the alias finding task is to retrieve email-addresses that belong to the person represented by the query. The set R_i includes the relevant answers for each query e_i. For example, the person whose name is Jason (represented in the query $V_q = \{$*term*="Jason"$\}$), uses two out of the email-addresses that are included in the corpus (graph): *jernst@cs.cmu.edu* (his department account) and *jernst@andrew.cmu.edu* (his general student account). The initial rankings generated from the example queries (shown in the bottom part of the figure) rank one of Jason's email-addresses at the top ranks in response to this query, but the other relevant email-address is ranked at the third rank. For another example query ($V_q^{i+1} = \{$*term*="William"$\}$), the initial ordering gives the correct answer in the second rank. The goal of learning is to improve the node orderings, such that the correct answers appear at the top ranks, across all the queries, to the extent possible.

3.2 Edge Weight Tuning: Error BackPropagation

As discussed earlier, the graph edge weight parameters Θ can affect the generated graph-walk based similarity scores (Section 2.2.4). Specifically, the parameters Θ, together with the graph topology, determine the transition probabilities in the graph (Equations 2.7 and 2.8). Rather than set the edge weights individually, the parametric weighting schema is based on the assumption that the relations represented by the links between entities in the graph have varying degrees of importance in evaluating inter-node relatedness. It is unlikely, however, that a single set of parameter values Θ will be best for all tasks. We are therefore interested in adapting the edge weights Θ per task.

Several methods have been developed that automatically tune the edge weight parameters in similar settings, where edge weights are parameterized by the edge type. We review these methods in Section 3.6. As an example of the weight tuning approach, we evaluate the error backpropagation algorithm due to Diligenti et-al [39], applying it to our framework for graph walks.

The algorithm applies a hill climbing approach, where the gradient of the weight of each edge type, θ_ℓ, is derived using the paradigm of error backpropagation in neural networks. The target cost function is a squared error function (typical to backpropagation

[113]), as follows:

$$E = \frac{1}{N}\sum_{z\in S} err_z = \frac{1}{S}\sum_{z\in S}\frac{1}{2}(p_z - p_z^{Opt})^2 \qquad (3.1)$$

where err_z is the error for a 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 arbitrarily set to 1 in case that the node z is known to be a correct answer or 0 otherwise. The error is averaged over a set of example nodes S. (The target nodes S can be sampled from the rankings of multiple queries, including relevant and possibly irrelevant nodes.)

The cost function is minimized by gradient descent with respect to every edge weight $\theta_{\ell'}$, as follows:

$$\theta_{\ell'} = \theta_{\ell'} - \eta\frac{\partial E}{\partial\theta_{\ell'}} = \theta_{\ell'} - \eta\frac{1}{|S|}\sum_{z\in S}\frac{\partial err_z}{\partial\theta_{\ell'}} \qquad (3.2)$$

The derivative of the error with respect to θ_ℓ is computed as the 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. Specifically, for every target node z, the full set of paths that are traversed in reaching z from the query distribution V_q can be recovered by a *path unfolding* procedure, common in neural networks (e.g., [39]). (We find the connecting paths up to length k using a concurrent walk from the query nodes and z, up to a meeting point.) Given the set of connecting paths, the derivative of the error e_z is computed as follows:

$$\frac{\partial err_z}{\partial\theta_{\ell'}} = (p_z - p_z^{opt})\sum_{t=0}^{k-1}\sum_{y\in U_z(t+1)} P(y, t+1 \rightarrow z, k)\cdot\frac{\partial p_y(t+1)}{\partial\theta_{\ell'}} \qquad (3.3)$$

where k is the number of walk steps; $U_z(t+1)$ 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 the total probability of reaching z at time k starting from y at time $t+1$.

The derivative of each intermediate node y with respect to an edge weight $\theta_{\ell'}$ is computed with respect to the probability mass attributed to y by its parents, $pa(y)$, as it is determined by the ratio of probability transferred to y, out of the total outgoing probability of each parent at time t; i.e., it is determined by the edge weight parameters, as specified in Equation 2.7.[1] That is, the derivative is as follows.

[1] In case that a different formula is used, such as Equation 2.8, it is straight-forward to update the derivatives accordingly.

$$\frac{\partial p_y(t+1)}{\partial \theta_{\ell'}} = \sum_{x \in pa(y)} p_x(t) \cdot \frac{\partial \frac{\sum_{\ell \in L_{xy}} \theta_\ell}{\sum_{y' \in ch(x)} \sum_{\ell' \in L_{xy'}} \theta_{\ell'}}}{\partial \theta_{\ell'}} \tag{3.4}$$

Finally, denoting as $C(\ell', L_{xy})$ the count of edge type ℓ' in the set of connecting paths L_{xy}, the explicit derivative is:

$$\sum_{x \in pa(y)} p_x(t) \cdot \frac{C(\ell', L_{xy}) O_x - C(\ell', L_{xy'}) \theta_\ell}{O_x^2} \tag{3.5}$$

where we use the abbreviation O_x for the total outgoing weight from node x, i.e. $O_x = \sum_{y' \in ch(x)} \sum_{\ell' \in L_{xy'}} \theta_{\ell'}$, and $ch(x)$ denotes the set of nodes that have incoming edges from x.

The target function is non-convex, and it is possible that the gradient descent procedure result in local minima [90]. 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 also possible to apply an optimization package such as LBFGS.

The hill climbing process involves re-computing the ranked list (by executing the graph walk) in every iteration. The described weight tuning procedure may therefore be time consuming. (The learning time varies across datasets; in general, iteration processing time shortens drastically with caching.) As we will see, relatively few example nodes give good performance [39]. Most importantly, however, once the set of weights is learned for a given task, it can be readily applied to new queries that are instances of that task, simply by setting the graph edge weight parameters to the learned weights Θ^* and performing the graph walk. That is, weight tuning involves no additional cost in responding to a query, compared to the basic graph walks.

3.3 Reranking

An alternative approach for improving graph walk performance that we suggest in this thesis is learning to *re-order* an initial ranking. The reranking approach has been used in the past for meta-search [30] and also for several natural-language related tasks (e.g., [32, 31]). 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 by the local model. Such high-level information is often useful in discriminating between the top ranked candidates. For example, a previous work [31] applied a

MaxEnt learner to perform named entity tagging; then, reranked high-probability annotations using features describing the entity boundaries predicted. Discriminative reranking has improved the state-of-the-art results of syntactic parsing, using sentence-level features to describe the high-probability candidate parse trees [32, 25].

In this section we suggest to apply discriminative reranking to learn to better rank graph nodes. Unlike weight tuning, reranking allows one to consider *global* properties of the graph-walk based similarity measure. In particular, we will use generic features that describe the *paths* traversed in the graph walk from the query distribution to a target node.

Next we give an overview of the reranking model. We then propose a generic set of reranking features that describe the graph walks (Section 3.3.2). The computation of these features, either throughout the execution of the graph walks, or post the initial graph walk, is discussed in Section 3.3.3.

3.3.1 Reranking Overview

The reranking model represents each output node z_{ij} through m features, which are computed by pre-defined feature functions $f_1, \ldots, f_m$. Assuming that relevant answer(s) are retrieved in the top ranks, reranking often considers only the top K candidates ($j = 1, ..., K$). The goal in reranking is to maximize the margin between the candidate that is known to be the best answer and the other candidates. The reranking problem is thus reduced to a classification problem by using pairwise samples [122]. Several algorithms have been used for reranking, including the Perceptron algorithm and its variants [32, 122] and Support Vector Machines [121]. In this section we describe in detail a boosting approach, due to Collins and Koo [32].

In this approach, the *ranking function* for node z_{ij} is linear, defined as:

$$F(z_{ij}, \bar{\alpha}) = \alpha_0 log(p_{z_{ij}}) + \sum_{k=1}^{m} \alpha_k f_k(z_{ij}) \tag{3.6}$$

where $\bar{\alpha}$ is a vector of real-valued parameters. As shown, this function considers also $p_{z_{ij}}$, the probability assigned to z_{ij} by the initial ranker. Given an initially ranked list of a new test example, it is re-ordered by $F(z_{ij}, \bar{\alpha})$.

To learn the parameter weights $\bar{\alpha}$, the boosting algorithm minimizes the following exponential loss function on the training data:

$$ExpLoss(\bar{\alpha}) = \sum_i \sum_{j=2}^{l_i} e^{-(F(z_{i1}, \bar{\alpha}) - F(z_{ij}, \bar{\alpha}))} \tag{3.7}$$

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 [118].

Other researchers have also applied the voted Perceptron algorithm [50] and other Perceptron variants to learn the weights $\bar{\alpha}$ of the linear ranking function [122, 28].

3.3.2 General Graph-based Reranking Features

We suggest several generic 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, and describe non-local properties of the graph walk. In particular, we define the following three types of feature templates:

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

- *Top edge label n-grams* - these features are similar 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 includes 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 that are reachable from multiple query source nodes are more relevant to the query.

Example. Consider the sub-graph depicted in Figure 3.2. Suppose that a task of interest is *threading*, where given a message, the goal is to retrieve other messages that are a response to the source message, or otherwise, messages that the specified message responds to. For the example query $V_q = \{msg{=}m_1\}$, the ranked list generated by graph walk is likely to include the messages m_2 and m_3 at the top ranks, as both nodes are linked to m_1 over several short connecting paths. In order to represent these nodes in term of the feature templates, we first recover the set of paths linking the query m_1 and each node. Overall, the node m_2 is reached over three paths according to Figure 3.2 (the sub-graph shown is assumed to contain all of the relevant connecting paths, up to length 2), including:

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

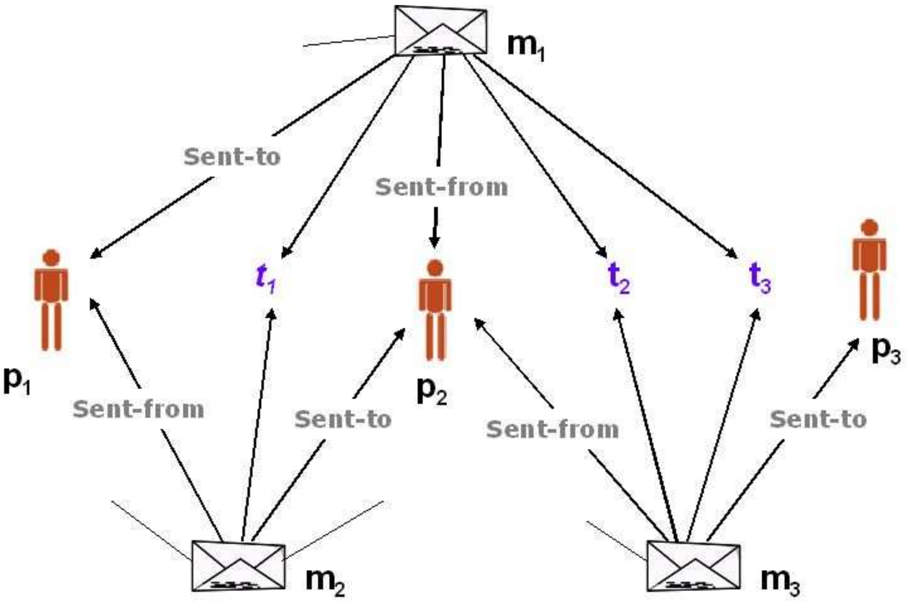


Figure 3.2: An example sub-graph, showing the connecting paths between the nodes m_1, m_2 and m_3.

$$m_1 \xrightarrow{sent-to} p_1 \xrightarrow{sent-from-inv} m_2$$
$$m_1 \xrightarrow{has-term} t_1 \xrightarrow{has-term-inv} m_2$$
$$m_1 \xrightarrow{sent-from} p_2 \xrightarrow{sent-to-inv} m_2$$

The node m_3 is connected to m_1 over three other paths:

$$m_1 \xrightarrow{sent-from} p_2 \xrightarrow{sent-from-inv} m_3$$
$$m_1 \xrightarrow{has-term} t_2 \xrightarrow{has-term-inv} m_3$$
$$m_1 \xrightarrow{has-term} t_3 \xrightarrow{has-term-inv} m_3$$

The representation of the target nodes m_2 and m_3 as features is shown in Table 3.1. As shown, the edge n-gram features represent the types of edge sequences traversed in the paths to each node. In the example, the query distribution include a single node, and the source-count feature equals 1 in both cases. The features are given in a binary form, where features that are not detailed for a given node in the table are assumed to be false for that node. It is possible to encode quantitative information by using discretized binary features (e.g., “source-count=1”,“source-count=2”). In case that real-value features are preferred,

feature type	m_2	m_3
edge unigrams	*sent-from* *sent-from-inv* *has-term* *has-term-inv* *sent-to* *sent-to-inv*	*sent-from* *sent-from-inv* *has-term* *has-term-inv*
edge bigrams	*has-term.has-term-inv* *sent-from.sent-to-inv* *sent-to.sent-from-inv*	*has-term.has-term-inv* *sent-from.sent-from-inv*
source-count	source-count=1	source-count=1

Table 3.1: Feature representation of nodes m_2 and m_3, given that the query node is m_1, the graph is as described in Figure 3.2 and walk length $k = 2$.

feature weights can denote the count of the edge n-gram sequence in the set of connecting paths; or, feature weights can also denote the probability mass that was transmitted through each edge type (unigrams) from the query nodes to the target node. Example of the latter feature weighting is included in Section 3.3.3.

Intuitively, given the features represented in Table 3.1, message m_2 is more likely to belong to the same thread as m_1, compared with m_3. Specifically, the edge sequences *sent-from.sent-to-inv* and *sent-to.sent-from-inv* are typical of a response to a message, where the sender becomes the recipient, and vice-versa. Discriminative reranking is therefore expected to assign high weight to these features. Notice that manipulating the edge weights cannot capture this long-range pehnomena. In particular, the sequence *sent-from.sent-from-inv* includes the same individual edge types, but is less indicative of a thread, or email response, at path level.

The given feature set is general, in the sense that it is applicable to any task phrased as a query in the graph. In addition to this general feature set, the design of additional task-adapted features may improve performance further. Various properties of the set of connecting to the target node can be represented as features; e.g., information about the nodes visited in the course of the graph walk may be useful for certain problems. External information, which is not included in the graph but considered relevant for the task, can also be added to a node's feature description.

In the next section, we describe a couple of approaches for computing the reranking features; either during the graph walk, or as a separate procedure.

3.3.3 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 the probability of reaching z from x over a multi-step graph walk, $V_k(z)$ (Equation 2.6). 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. 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:

$$\begin{aligned} F_0(z) &= \mathbf{0} \\ F_d(z) &= \sum_y \left(\Pr(x \xrightarrow{\ell} y) \cdot f(x \xrightarrow{\ell} y) \right) \cdot Q(y \xrightarrow{=d-1} z) \end{aligned}$$

where $Q(y \xrightarrow{=d-1} z)$ is the probability of stopping at z in graph walk originating from y of length $d-1$. Finally, we can define:

$$F_k(z) = \gamma \sum_{d=1}^{k} (1-\gamma)^d F(x \xrightarrow{=d} z) \qquad (3.8)$$

$F_k(z)$ can be computed throughout the execution of the graph walk, while computing $V_k(z)$ [28]. An algorithm for computing the graph walk and the node feature vector representation concurrently is given in Table 3.2. The algorithm computes a distribution over edge types, weighting each edge by the probability mass that was traversed via that edge en route to the target node z. This algorithm can be extended to compute edge bigrams by recording the set of edge types from x to y (that is, the union $\bigcup_x L_{xy}$) at every iteration, and using this history in walking from y in the consecutive iteration. (The bigram feature weights can be replaced by counts, or assigned similarly to edge unigrams in Table 3.2.) Similarly, n-gram edge sequences can be computed. The feature function may include additional properties of a path segment, such as the source node type $\tau(x)$ etc.

The cost involved in computing the feature function described in Table 3.2 is constant per each node visited. In our implementation, this approximately doubles the cost of the graph walk computation. Maintaining n-gram edge sequence features, however, requires memory of size $|\Theta|^n N$.

Alternatively, rather than calculate the feature vectors on-the-fly for every node visited during the graph walk, it is possible to compute the feature vectors only for the top K nodes retrieved that are to be reranked. Feature extraction in this case takes place after the graph walk is completed. Given the set of connecting paths, which can be extracted

```
1. let $V_0$ be the probability distribution $V_q$, and let $F_0$ be an empty distribution.
2. for $d = 1, \ldots, k$ do
    • let $V_d(x) = 0$ for all $x$
    • for each $x_i \in V_{d-1}$ do
        (a) $V_d(x_i) = \gamma V_{d-1}(x_i)$
        (b) $F_d(x_i) = \gamma F_{d-1}(x_i)$
        (c) for each node $y_j \in ch(x)$, $\ell \in L_{x_i y_j}$
            – increment $V_d(y_j)$ by $(1-\gamma)V_{d-1}(x_i)Pr(x_i \xrightarrow{\ell} y_j)$
            – increment $F_d(y_j)$ by $(1-\gamma)V_{d-1}(x_i)Pr(x_i \xrightarrow{\ell} y_j) \cdot f(x_i \xrightarrow{\ell} y_j)$
3. return $V_k(z), F_k(z)$.
```

Table 3.2: An algorithm for computing $V_k(z)$ and $F_k(z)$ concurrently, given transition probabilities $Pr(x_i \longrightarrow y_j)$.

via the path unfolding procedure as described above, it is straight-forward to derive the feature values (see example in Section 3.3.2). In this thesis, we take this latter approach for feature computation.

Unlike the weighted tuning approach, reranking requires 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 applying the reranking function. A main concern is that while edge label *bigrams* correspond to a relatively small space, higher order *n-grams* may translate to large feature space. Given a limited number of training examples, this is likely to lead to over-fitting. In case that high-order n-grams are incorporated, it is therefore required to apply techniques such as feature selection or regularization.

3.4 Path-Constrained Graph Walks

While node reranking allows the incorporation of high-level features that describe the traversed paths, it is desirable to utilize such information directly in the graph walk process, so the quality of the initial rankings produced is improved. 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 will depend on the random walker history. Performance-wise, in case that 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 of dynamic edge weights, given the history of a walk, based on training examples. The second component of the algorithm adapts the random walk to consider path history. The space of path histories is $|\Theta|^k$, so that a compact representation is required.

In general, the approach suggested is to model paths observed in a training dataset as a path tree, where every path is associated with the probability of reaching a relevant target node following this path, based on the labeled examples. Path probabilities are propagated in the tree to obtain estimates of the parameters Θ for various histories of graph walk. In order to perform a graph walk, which co-samples from the graph and the path tree, we will compactly represent walk histories by associating each node visited in the graph walk with the corresponding vertices of the path tree.

Next we describe in detail the process constructing the path tree, and the estimation of the path tree's vertex probabilities (Section 3.4.1). We then describe a modified algorithm of path-constrained graph walks (Section 3.4.2).

3.4.1 Path-Tree Construction

We construct a path-tree T using the training set of N labeled queries. Let a *edge sequence* 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, and extract the corresponding edge sequences. (The connecting path set per a single node may include multiple instances of an edge sequence, for different intermediate nodes visited.) Let C_p^+ be the count of an edge sequence p within the paths leading to the correct nodes, over all examples N; and similarly, let C_p^- denote its count within the paths leading to the negatively labeled nodes in the example set. The full set of

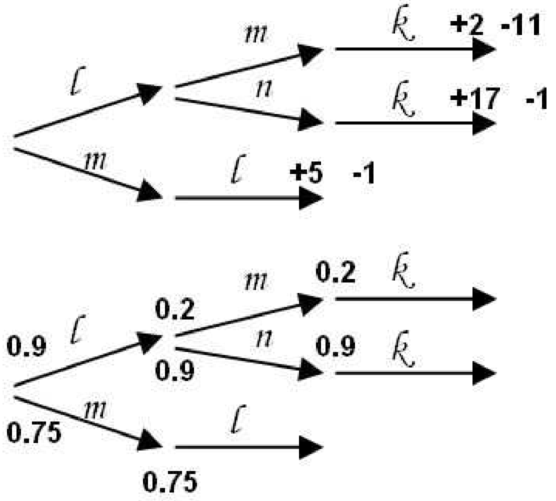


Figure 3.3: An example path-tree: path counts (top) and vertice probabilities (bottom).

edge sequences observed can be represented as a tree. The leaves of the tree, which correspond to full edge sequences traversed to a target node, 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 p, based on the observed examples. In our experiments, we found that better performance is obtained if $C(p)$ are evaluated using acyclic paths only (that is, paths where nodes are not re-visited). We therefore consider only edge sequences that are derived from the relevant acyclic paths in the graph. In the rest of this section, we refer to this set of edge sequences as *paths*, from which the tree is constructed.

Example. Consider the graph shown in Figure 3.2, where the query is $V_q = \{m_2\}$, node m_1 is considered a correct answer to the query, and node m_3 is an incorrect answer. The set of paths that lead to each of these node over a 2 step graph walk are detailed in Section 3.3.2. As described, there are three unique paths that lead to the relevant node m_2, where each path occurs once. Each path is therefore credited with a positive +1 count, as follows:

$$\xrightarrow{sent-to} \quad \xrightarrow{sent-from-inv} \quad +1$$
$$\xrightarrow{has-term} \quad \xrightarrow{has-term-inv} \quad +1$$
$$\xrightarrow{sent-from} \quad \xrightarrow{sent-to-inv} \quad +1$$

The irrelevant node m_3 is reached via two unique paths, where one of the paths repeats twice, and is therefore attributed a negative -2 count, as follows.

$$\begin{array}{ccc} \xrightarrow{sent-from} & \xrightarrow{sent-from-inv} & -1 \\ \xrightarrow{has-term} & \xrightarrow{has-term-inv} & -2 \end{array}$$

Overall path counts, based on these nodes, are:

$$\begin{array}{cccc} \xrightarrow{sent-to} & \xrightarrow{sent-from-inv} & +1 & 0 \\ \xrightarrow{has-term} & \xrightarrow{has-term-inv} & +1 & -2 \\ \xrightarrow{sent-from} & \xrightarrow{sent-to-inv} & +1 & 0 \\ \xrightarrow{sent-from} & \xrightarrow{sent-from-inv} & 0 & -1 \end{array}$$

The count statistics in this case show that while the path *has-term.has-term-inv* leads to relevant nodes, this path is relatively 'noisy', in the sense that this path also reaches incorrect responses with relatively high probability. The final path statistics per the example above correspond to a path tree that includes three branches originating from the root (representing the *sent-to*,*has-term* and *sent-from* edge types), and four leaves. The path *has-term.has-term-inv* will be associated with a leaf smoothed probability of 0.4, indicating the probability of reaching a correct target node over this path in the underlying data. Having accumulated path counts for a sufficient number of nodes, the count statistics are expected to represent general phenomena in the graph.

Further, the tree leaf probabilities are propagated backwards to the tree vertices, applying the *MAX* operator.

In our experiments, we have also considered a different *averaging* scheme for estimating vertex probability, where the positive and negative counts at the downstream paths (leaves) p_i from that node were summed (i.e., $\sum C^+_{p_i}$ and $\sum C^-_{p_i}$), and the smoothed vertex probabilities were computed using the cumulative counts. The results using the *MAX* operator were superior in most cases.[3]

Example. An example path tree is given in Figure 3.3. This path-tree includes three paths, constructed from the edge types k, l, m, n, and their observed counts. According to the stated counts, the leaf probability corresponding to the path $l.m.k$ is estimated at 0.2 (3/15), and at 0.9 per the path $l.n.k$. The bottom part of the figure gives the path-tree vertex probabilities. As shown, at the root of the tree, the probability of reaching a relevant

[3] Interestingly, in reinforcement learning an agent also selects the step that maximizes the future reward in its path to a goal.

```
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 3.4: Pseudo-code for path-constrained graph walk

answer is estimated at 0.9 (computed as $MAX(0.2, 0.9)$) if an edge of type l is followed, and at 0.75 if an edge of type m is selected. We assume that probabilities associated with edge types not included in the path-tree at a given vertex to be zero.

3.4.2 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 the path tree T. Walk histories of each node x visited in the walk are compactly represented as pairs $\langle t, x \rangle$, where t denotes the relevant vertex in the path tree. (This means that if x was reached via K different paths, it would be represented using K node pairs.) For example, according to the path-tree in Figure 3.3, suppose that after one walk step, the maintained node-history pairs include $\langle T(l), x_1 \rangle$ and $\langle T(m), x_2 \rangle$. If x_3 is reached in the next walk step from both x_1 and x_2 over paths included in the path-tree, it will be represented by multiple node pairs, e.g., $\langle T(l \rightarrow n), x_3 \rangle$ and $\langle T(m \rightarrow l, x_3 \rangle$.

The pseudo-code for the path-constrained graph walk is given in Figure 3.4. In the algorithm, the path-tree probabilities are treated as dynamic edge weights. These weights are normalized at each node (pair) traversed to generate local transition probabilities (following Equation 2.7).

Example. Consider the node pair $\langle T(\ell), x_1 \rangle$, where there are outgoing edges from x_1 of type m, n and k. The effective edge weights for this node-history pair are $\theta^*_m = 0.2$ and $\theta^*_n = 0.9$, according to the path-tree shown in Figure 3.3. The sequence of edges $\ell.k$ does not exist as a path prefix in the path-tree, and therefore $\theta^*_k = 0$. Given the modified edge weights, the graph walk can proceed according to its original schema.

Notice that the number of nodes visited in the modified graph walk increases relative to an unconstrained walk. On the other hand, paths in the graph that are not represented in the path tree are pruned. (It is possible, of course, to assign a small probability to previously unseen paths.) In addition, it is straight-forward to discard paths in T that are associated with a lower probability than some threshold. A threshold of 0.5, for example, implies that only paths that led to a majority of positively labeled nodes in the training set are followed. Path pruning has a direct effect on the time complexity of the walk, reducing the number of nodes and edges visited.

3.5 Method Comparison

So far, we presented three different approaches for learning to rank graph nodes given labeled examples and initial rankings generated using graph walk. In this section we discuss the relative strengths and weaknesses of these methods with respect to key criteria, including: the scope of information that can be considered by each approach; and the extent to which each learning method can alter the initial rankings produced by the graph walk; and, the differences between the methods in terms of their training procedures and application requirements during runtime.

Global vs. Local Information.

The graph walk process is strictly Markovian, where the random walker does not "remember" the history of the walk. In our framework, this means that edge probabilities (or, edge weights) are fixed over the course of the walk. Similarly, in learning the graph edge weights using methods like error backpropagation, 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 reranking approach allows one to exploit global properties of the walk, as it can represent information about the full paths traversed to reach a target node. In particular, we suggest features that describe *edge sequences* traversed over multiple time steps. The path-constrained graph walk method embeds high level information, considering the paths traversed, already into the graph walk.

Overall, reranking is perhaps the "most global" method, out of the approaches considered. In addition to edge sequences, reranking can incorporate features that describe properties of the *collection* of paths leading to a node. For example, the *source count* feature denotes the number of different query nodes that link to the target node. Similarly, reranking features can model the number of paths leading to a node, and other global properties pertaining to a node connectivity. The path-constrained walks, in contrast, consider individual paths and cannot model properties at the path set level.

Finally, reranking can also model arbitrary domain-specific features, incorporating additional relevant information sources that are independent of the graph walk.

Learning Impact.

Learning may alter the graph-walk based initial rankings to varying extents. Next we discuss the learning methods with respect to their ability to substantially change the initial rankings.

Weight tuning 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 2.2.4). For instance, an exponential decay over the transmitted probabilities is applied by Personalized PageRank 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. In other words, we claim that tuning the edge weight parameters Θ has a limited impact on the final rankings.

Unlike weight tuning, discriminative reranking is not constrained to the graph walk paradigm, and a ranked list can be significantly altered using the re-ranking procedure. However, for efficiently reasons, reranking is only applied to the top nodes retrieved. Performance using reranking is therefore limited, as it depends on the quality of the initially ranked lists. It is therefore desirable to apply reranking in combination with a good initial ranking function. In particular, reranking may be applied in combination with graph walks that use a learned set of edge weights Θ^*, or with path-constrained walks, in a sequential fashion.

Finally, the path-constrained graph walk variant can affect the output rankings to a large extent, as it incorporates a bias towards specific paths during the graph walk. Thus, nodes that are close to the query nodes, but are so related over a relation that is not meaningful, for example, will be excluded for the ranked list. Nevertheless, the path-constrained graph walks reflect the graph topology. The deviation of the path-constrained graph walk result from the initial rankings can vary, depending on the applied threshold.

Method Applicability.

The methods differ in their training requirements, the result of learning, and its application to unseen instances.

The error backpropagation weight tuning approach requires re-computing graph walk rankings in each learning iteration. In addition, several independent learning sessions are recommended, in order to avoid local minima. Weight tuning therefore needs to be run offline, and may require a relatively long time to train. On the other hand, this method yields a set of edge weights Θ that is optimized for the given 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, with no overhead during runtime.

The reranking approach requires only a one-time execution of the graph walk. Features describing the top graph nodes retrieved are derived either during the graph walk, or as a separate step, using a path unfolding procedure. The model generated in reranking is a weighted function, which can be readily applied to feature vectors describing other instances of the learned task. In terms of training requirements, a rule of the thumb is that for discriminative methods such as reranking, the larger the feature space modeled, the larger is the training set required. This generally means that compared with weight tuning, we expect more examples to be required using the reranking approach. That is, there is a trade-off between the methods of weight tuning and reranking, where weight tuning can efficiently learn with very few examples [39], but the feature space that it can represent is much more limited, namely, individual edge types.

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. The additional processing time is affected by the types of features used, and the fashion in which they are computed.

The path-constrained graph walk approach is simple and fast to train. Like reranking, it requires a single execution of the graph walk for the given example queries, as well as path unfolding. Unlike reranking, this approach uses the full paths as features, such that the step of computing feature values is trivial. Given the path-tree learned for the task of interest, the constrained graph walks are applied to the unconstrained graph walks original schema.

Summary

We conclude that edge weight tuning is a natural learning tool, as it is derived from and applied directly to the graph walk. However, it is limited to considering local information of the graph walk. In addition, the impact of the graph's edge weights on the graph-walk based rankings may be limited in some cases. Path-constrained walks consider global information about path relevancy, and can have a great impact on the quality of the output ranked list. Finally, reranking allows the consideration of arbitrary features, and features that describe a node by properties of the set of paths that connect it to the query. Reranking, however, operates only on the top candidates retrieved by the graph walks; also in response to a query, reranking requires feature encoding overhead, unlike the other approaches.

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 reranking model. This hybrid approach has been used successfully in the past on tasks like parsing [32].

3.6 Related Work

We first review general learning techniques previously suggested to improve graph-walk based rankings 3.6.1. In Section 3.6.2, we discuss in detail algorithms that specifically adapt the edge weight set Θ. Section 3.6.3 reviews works that consider global features in combination with local search methods (such as the graph walk), as well as methods that utilize path information in graphs.

3.6.1 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 [24] applied gradient ascent on the elements of the link matrix constructed by the related HITS algorithm [75], altering rankings to more closely align with 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 results of varying quality.

It was later suggested to learn the teleport (reset) vector in the PageRank algorithm, to affect node rankings [137]. The input 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 modeled as linear constraints. In their work, the teleport vector learned reflects fixed preferences from a data administrator's point of view; adapting the reset distribution is redundant in our framework, however, as it is defined dynamically per query, including the query nodes (Equation 2.4).

Agarwal et-al [4] 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 goal is to tune the transition probabilities of the link matrix. (In their model, teleport transitions are modeled as regular transitions to a dummy node, such that tuning of the teleport vector is included within the general transition matrix.) They present NetRank, an algorithm that optimizes the transition matrix probabilities, such that the final node probabilities are similar in terms of KL divergence to the results of an initial flow, and the given pairwise ranking preferences are satisfied. NetRank does not provide generalization guarantees, and does not generalize well in the experiments. In a later work [3], a theoretical justification is given for this approach. The authors show 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 [3] draw a connection between learning to rank graph walks, where directed edges denote structural inter-entity relations, 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 [5], applied in associative networks. The authors argue that in contrast to Laplacian smoothing, which assigns arbitrary scores to nodes, thus inducing all possible node permutations, certain node orders may be impossible to achieve in graph walks over a directed graph. That is, the hypothesis space of the PageRank model 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 to Laplacian-based smoothing for directed graphs.

3.6.2 Edge Weight Tuning

Several methods have been developed that automatically tune the edge weight parameters in extended PageRank models, where edge weighted are associated with the relation type that they represent. Earlier works, including the XRank [56] and ObjectRank [7] models, experimented with assigning different edge weights, but did so manually.

Nie et-al [100] have suggested PopRank, an object-level link analysis model that ranks the objects within a specific domain, where relationships between objects are heterogeneous. They apply a simulated annealing algorithm to explore the search space of all possible edge weight assignments, with the goal of reducing the difference between partial rankings given by domain experts, and the ranking produced by the learned model. In order to make learning time manageable, they use a subgraph in the learning process, trading optimality for efficiency. The subgraph used consists of a set of concentric circles with the training objects in the center as the core.

Toutanova et-al [136] have constructed a special graph including diverse word-to-word relationships, describing WordNet relations, morphology links and word features derived from dependency relations. They applied truncated Personalized PageRank graph walks to induce smoothed word probabilities, using these probabilities for the task of predicting prepositional word attachment. 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. 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 by which the parameters were grouped. For example, parameters were binned based on the observed number of word occurrences. However, it is reported that the simplest model having a single equivalence class across all of the graph nodes performs on average as well as the more complex models.

Agarwal et-al [4] 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 learning general transition probabilities, where the edge weights are non-parametric. They find that since the approximate gradient-descent approach estimates a small number of global weights, it can generalize from training to test instances that involve completely different nodes, far away in the graph, with a much smaller number of examples, compared with the latter settings.

3.6.3 Graph Walks using Global Information

The reranking approach has been applied in the past to a variety of structure prediction tasks, including parsing [32, 31, 25], machine translation [123], semantic role labeling [135] and more. In general, structure prediction problems are usually factorized into a chain of local decisions in order to apply efficient inference algorithms, such as dynamic programming. The factorized model, however, can only consider local features, and the maximum likelihood structure predicted is often sub-optimal. In the reranking approach, rather than predict the most likely candidate, the top *K* most likely candidates are generated in the search process. These candidates are then evaluated based on global features; i.e., properties pertaining to the long range dependencies in the predicted structure. These features allow the reranking classifier to improve on the initially ranked list, by demoting candidates that violates various constraints or preferences in the subject domain. In the problem of semantic role labeling, for example, a hard constraint is that arguments cannot overlap with each other or the predicate, and a soft constraint is that a predicate have no more than one AGENT argument [135].

To the best of our knowledge, we are the first to consider global features in graph-walk based induced similarity measures in general. In particular, we are the first to suggest reranking to improve rankings of graph nodes, using features that describe global properties of the paths traversed.

Researchers have pointed out that the performance of the reranking approach is bounded by the quality of the top candidates reranked. Rather than apply high-level constraints to the results of a localized search, it is therefore desired to consider such constraints earlier, in the inference process. Several previous works suggested models that incorporate high-level constraints in the inference procedure, tailored for specific problems. For example, Punyakanok et-al [108] apply an inference procedure based on integer linear programming that supports the incorporation of structural constraints for the semantic role labeling task. Instead of predicted a structure, they predict the local components of the structure (verb arguments), using classifiers that emphasize high recall. The inference procedure then takes confidence scores assigned to each individual component as input, and outputs the best global assignment that satisfies the high-level constraints. Specifically, they apply integer linear programming to reason about the global assignments. In another recent work, Huang [64] proposes *forest reranking*, a method that reranks a packed forest of exponentially many parse trees. Since exact inference is intractable with non-local features, he presents an approximate algorithm inspired by forest rescoring. In the proposed approach, non-local features are computed incrementally from bottom up, that is, as early as possible (‘on-the-fly’). The decoder can then consider this information at internal nodes of the parse tree generated. The path-constrained graph walk variant suggested in this chapter applies a similar idea. Rather than compute the graph walk as a Markovian process, the algorithm allows the edge weights parameters to be determined by path information, built incrementally as the graph and the path-tree are traversed concurrently. To our knowledge, the approach of using path information as guidance within the framework of random graph walks is novel.

Other researchers have considered path information in classifying relations between pairs of objects connected over individual structures, such as entities that co-appear in a sentence dependency tree [127]. In particular, rich features sets were proposed that describe these paths [36, 17]. The approach of discriminative reranking similarly incorporates path information as well as arbitrary feature sets. Given that individual structures are represented within a combined graph, the graph walk framework allows to *retrieve* the most related entities over relevant paths, rather than evaluate a large space of entity pairs.

Finally, path constraints are often used in the spreading activation paradigm (see Section 2.4.5) in order to eliminate probability propagation to irrelevant areas in the graph. In spreading activation, path preferences are coded manually, and enforced deterministically. We use learning to obtain path information, and apply path features probabilistically.

3.7 Summary

We presented three approaches for learning to adapt the graph walk based similarity measure for a given task. The first approach is weight tuning, where we adapted an error backpropagation hill climbing method to our framework of finite graph walks. The second approach is reranking. We presented a set of general features that encode high-level properties of the paths traversed in the graph walk. While additional specialized features can be designed per task, the basic set of features proposed can be applied to any task, as it represents long-range relationships between entities in the graph. In practical settings, however, only the top nodes retrieved by the initial graph walks can be reranked. It is therefore desirable to incorporate global features in the initial ranking process. We therefore suggested a novel graph walk variant, in which edge probabilities depend on the history of the walk. The proposed algorithm represents walk history efficiently using a compact path-tree, in which edge weights are derived based on path information. Both reranking and the path constrained graph walk methods can be combined with weight tuning. In the future, we would like to enhance this model by learning the edge probabilities using a richer set of features.

Chapter 4

Case Study: Personal Information Management (PIM)

In this chapter, we evaluate personal information management as a case study of 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 [63, 62]. In addition, they can be related via common key words that appear in the relevant correspondence in the email corpus [89]. Such interpersonal 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 information source for related applications and tasks. Using graph walks, these multiple email-related aspects of information 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 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 evaluate the extent to which learning can further enhance the graph-walk generated similarity

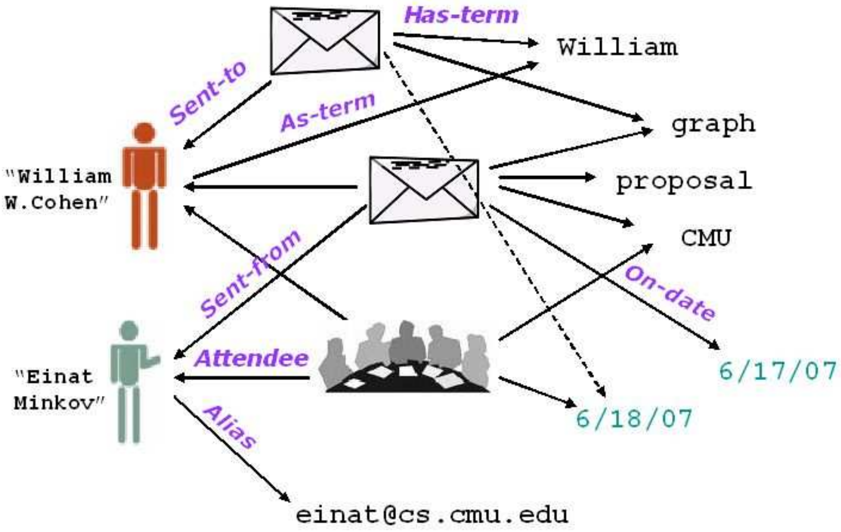


Figure 4.1: A joint graph representation of email and meetings data

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. Experimental results are then given for all tasks, where both base graph walks and the learning techniques are evaluated. For each task, the graph-based results are also compared against relevant baselines. The chapter concludes with a discussion of related works, and a summary.

4.1 Email and Meetings Graph Representation

A graph example including both email and meeting objects is given in Figure 4.1. The corresponding graph *schema* is detailed in Table 4.1. 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 well as *email addresses*, *persons* and *dates*. Directed graph edges represent relations like *sent-from*, *sent-to* and *on-date*. As shown, we distinguish between *has-term* and *has-subject-term* relations. In addition, in the suggested schema, a person node is linked to its constituent token values with an "as-term" edge. Similarly, terms that are

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*
	mtg-on-date	*date*
	mtg-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*
	mtg-has-term^{-1}	*meeting*
	is-email	*email-address*
	as-term^{-1}	*person*
date	on-date^{-1}	*message*
	mtg-on-date^{-1}	*meeting*

Table 4.1: Email and meetings node and relation types. (Inverse edge types are denoted by a superscript.)

identified as email-addresses 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 for which the evaluated string similarity score is higher than a threshold. It is straightforward to add other information types available; e.g., organizational hierarchy inter-personal relations, if given, etc.

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 longer texts, 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 *message* files, for example. Many tasks can benefit from the combined representation of messages and meetings. For instance, relevant messages (or other potentially included entities, like papers and presentations), can be retrieved as related background material for a meeting in this framework. Similarly, the social network information embedded in emails may be enhanced given 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 *activities* in the graph. While user activities are often implicit, they can be represented in the graph as explicit entities. Many user-created *folders* focus messages related to a project activity, for example. Folder structure can be extracted from the corpus, where *folder-message* associations are given. Recently, there is an increasing interest in activity-based interfaces for managing information at the desktop [12]. Such interfaces may provide additional evidence regarding activities and their relations with other entities.

4.2 PIM Tasks as Queries

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 information is available for retrieval and analysis.

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 4.2: Query realizations of the considered tasks

The framework is general, and many query and search types are possible. One can search for similar or related items to a set of objects of interest using this the framework (e.g., "show persons names that are related to person *P*"); alternatively, a user can search for a specific item, using loose associations (e.g., "show those email-messages that are related to 'Jenny', around 'March 1' ").[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 previously defined (Section 4.2), a *task* is a *query class*, for which 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, learning can be applied to adapt the graph-based similarity measure for each task.

Following is a description of the tasks evaluated in our case study. Table 4.2 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 recognition for tasks like social network analysis or studies of social interaction in email. However, while the referent of a name mention 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

[1] The latter query can be supported if links are drawn between proximate dates.

to “Andrew Y. Ng” and not “Andrew McCallum” 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 *term* that is identified as a name-mention in an email message *m*, retrieve a ranked list of *person* nodes. Assuming that the identity of the message *m* is available, one a contextual query can be phrased, 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 [84], 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 that once obtained, thread information can improve message categorization into topical folders [76].

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 often needs to retrieve email-addresses from his or her address book. In some cases, this is done by searching for a message 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.

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 associating 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, such as a project or a recurrent activity (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, with the goal of classifying an email message to a single relevant folder [119, 9, 61]. We are interested in a scenario where a user may be interested in associating a message to *multiple* relevant folders. (Multi-tagging is supported, for example, by 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 many previous works, which classified email messages to a single 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.

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 to get involved in the future in an ongoing project activity, represented by a folder in an email corpus. While finding experts [8, 106] and recommending recipients [22] relies on evidence observed in the past, the prediction of future involvement 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 for an activity may be valuable to an organization, as it may promote early involvement of relevant individuals in a project. We phrase this task as a query that includes a *folder* representing an ongoing activity, where the entities retrieved are of type *email-address*.

4.3 Experimental Corpora

We experiment with the following corpora.

Management game. This corpus contains email messages collected from a management course conducted at Carnegie Mellon University in 1997 [96]. 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.

Enron. The Enron corpus is a collection of email from the Enron corpus that has been made available to the research community [76]. This corpus can be easily segmented by user: in the experiments, we used the saved email of several different Enron 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.

Meetings. This 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.

Personal. This is a collection 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, the Management game and the 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, 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 author's home page. Unfortunately, due to privacy issues, the Management game, Meetings and Personal corpora can not be distributed.

4.4 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 is 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 as Gw:Uniform), and also for graph walks where the edge weights have been tuned (Gw:Learned). 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.[2] Further, in all experiments we applied reranking on top of the uniform-weighted graph walk results. For every example, the top 50 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-constrained graph walk variant, path trees were learned using the train and development sets, where the top positively and negatively ranked labeled nodes were considered. In general, the number of negative examples was limited to the number of positive example available, so that the constructed path trees are balanced.

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

Statistical significance in comparing performance of the various methods was obtained using a two-sided Wilcoxon test [82], at significance level of 95%.

[2] We found that the error function and MAP are well-correlated.

	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 4.3: Person disambiguation corpora and dataset details.

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 experiment regarding the framework.

4.4.1 Person Name Disambiguation

As described in Section 4.2, 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 $\tau =$*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 evaluate this task using three labeled datasets, as detailed in Table 4.3.

The Management game corpus has been manually annotated with personal names [96]. Along with the corpus, which contains correspondence between teams of students participating in a management game, 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 labeled 106 cases in which single-token names were mentioned in the the body of a message that did not match any person name included in the header. 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 fact 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 common in a message sign-off;

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

Table 4.4: Example person name type distribution per dataset.

- *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 labeled 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 4.4 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 4.3).

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 are, 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 [29]. 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 [29] between the term and any single token of the personal name (ranging between 0 to 1). In addition, we incorporated a nickname dictionary,[3] such that if the name term is a known nickname of the person name, the similarity score of that pair is set to 1.

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

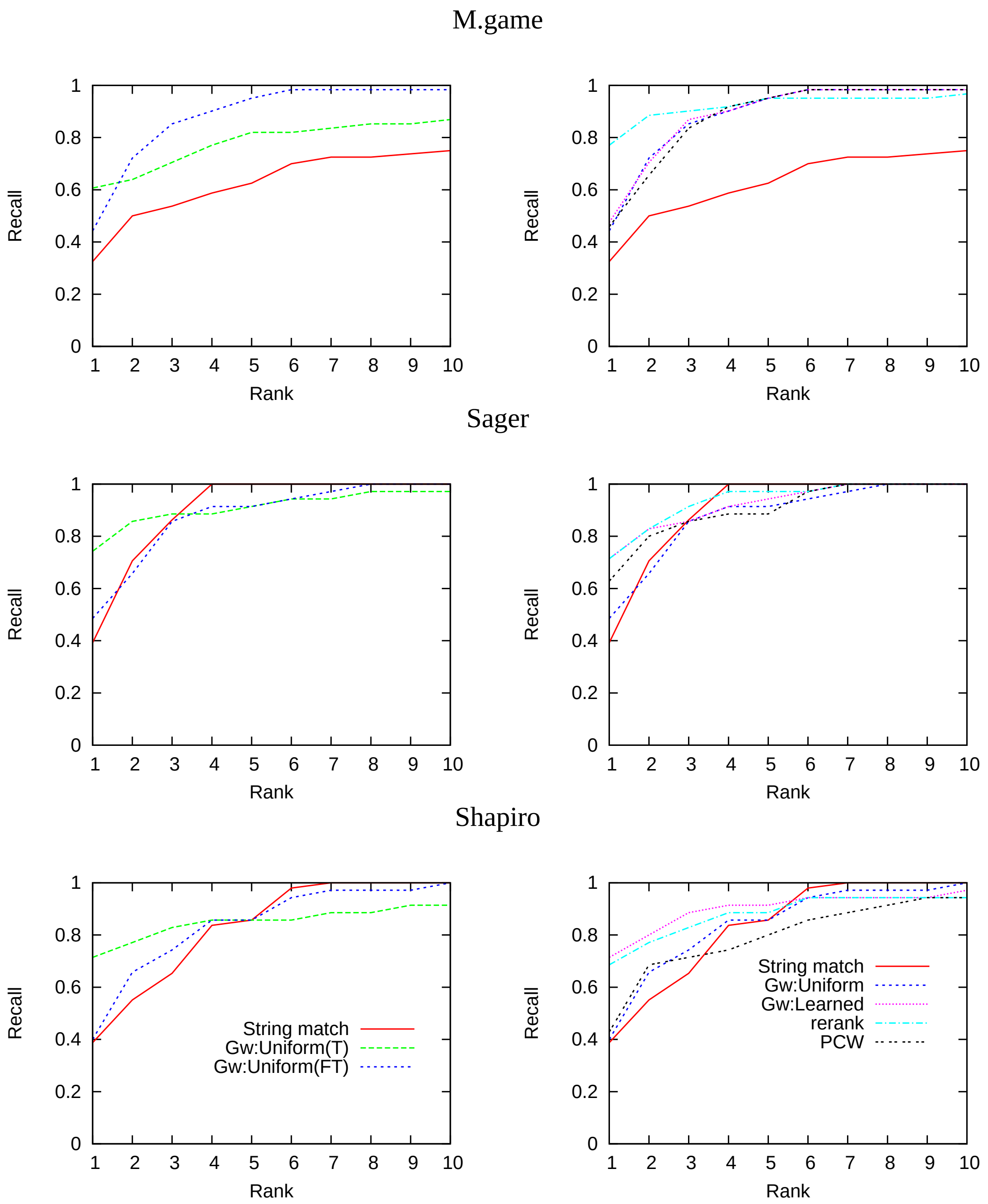


Figure 4.2: Person name disambiguation test results: Recall at the top 10 ranks, for baseline and plain graph walk, where the query includes a term only (Gw:Uniform(T)), or term and file (denoted as Gw:Uniform(T+F)) (left); and for all methods using contextual queries (T+F) (right).

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

Table 4.5: Person name disambiguation results: MAP and accuracy. The columns denoted as “T” give results for queries including the relevant *term* node, and the “T+F” columns refer to queries that include both *term* and *file* information; the $*$ sign denotes results that are statistically significantly better (in MAP) than the baseline (String sim.), and the $+$ sign marks results that are significantly better than graph walk using uniform weights (Gw: Uniform).

The results are given in Table 4.5, listing MAP and accuracy results. (These and other evaluation measures are described in Appendix B.) In addition, Figure 4.2 shows the average recall at every rank down to rank 10. As shown, 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 performs 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 similarity matches 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. 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 these files. Note that in our graph scheme there is a direct path between terms to person names (via the *as-term* relation), so that person nodes may receive weight via this path as well. The column labeled “T” in Table 4.5 gives the results of the graph walk from the *term* probability vector, and Figure 4.2 (left column, Gw:Uniform(T)) shows recall at each rank, down to rank 10. As can be seen in the results, the graph walk performance is preferable to string matching. 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 4.2 (left part, Gw:Uniform(T+F)), 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 attribution of probability score to nodes that link to the *file* node but not to the *term* node. Adding the file node to the query therefore adds noise to the output ranking. This is reflected in the lower MAP and accuracy evaluation scores. This shortcoming is addressed in this case using learning.

Learning

Weight tuning. We learned the graph edge weights using the error backpropagation method (denoted as Gw:Learned). We applied learning to each corpus separately. Edge weight learning resulted in a comparable performance to the *term*-query graph walks using uniform weights in all cases. However, learning the weights significantly improved performance for the contextual search (T+F) for the two Enron corpora. As described earlier, the Enron datasets were created using a simpler automatic procedure. We conjecture that the difference in weight learning performance between the management game and Enron corpora is due to the difference in name mention distributions (and consequently, due to different connectivity patterns).

Reranking. For re-ranking, we applied the *edge bigram* and *source count* features, as described in Section 3.3.2. 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 4.5 and in Figure 4.2, 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 4.5, reranking gives the best results for two of the three 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 Figure 4.2 shows reranking results, compared with the other methods, for the contextual search case.

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

	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 4.6: Threading corpora and dataset details.

4.4.2 Threading

In the thread recovery task, as discussed in Section 4.2, we are interested in retrieving *messages* that are adjacent to a given *message* in a thread (i.e., either a 'parent' or immediately consecutive messages). We consider this task as a proxy to the more general task of finding 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 4.6. 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 attachments 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 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 proxy for the more general task of finding related messages.

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
Gw: Uniform weights	0.59	0.53	0.36	0.36	0.46	0.35	0.20	0.22
Gw: Learned weights	0.68^{*+}	0.59	0.44^{+}	0.43^{+}	0.59	0.47	0.31	0.37
Gw: Path constrained	0.75^{*+}	0.73^{*+}	0.52^{*+}	0.45^{+}	0.67	**0.62**	0.39	**0.41**
Gw: Reranked	**0.77**$^{*+}$	**0.73**$^{*+}$	**0.59**$^{*+}$	**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
Gw: Uniform weights	-	0.55	0.49	0.44*	-	0.39	0.34	0.27
Gw: Learned weights	-	0.55	0.51*	0.44*	-	0.39	0.37	0.27
Gw: Path constrained	-	0.65^{+}	0.53*	0.45*	-	0.46	0.37	0.22
Gw: Reranked	**-**	**0.72**$^{*+}$	**0.65**$^{*+}$	**0.64**$^{*+}$	**-**	**0.56**	**0.51**	**0.51**
Farmer-D								
TF-IDF	-	0.69	0.36	0.32	-	0.55	0.25	0.13
Gw: Uniform weights	-	0.65	0.53*	0.50*	-	0.48	0.40	0.41
Gw: Learned weights	-	0.72^{+}	0.57^{*+}	0.50*	-	0.61	0.46	0.41
Gw: Path constrained	-	0.76^{+}	0.63^{*+}	0.52^{*+}	-	0.66	0.54	0.45
Gw: Reranked	**-**	**0.83**$^{*+}$	**0.65**$^{*+}$	**0.61**$^{*+}$	**-**	**0.70**	**0.56**	**0.52**

Table 4.7: Threading Results: MAP and accuracy. The $*$ sign denotes results that are significantly better (in MAP) than the TF-IDF baseline; and the $+$ sign denotes results that are significantly better than graph walks using uniform weights (Gw:Uniform). Four configurations are included, where email components are gradually removed (as detailed in the header by the checkmarks), and the best result for each configuration is marked in boldface.

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 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 4.7), 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 that corresponds to the original message, and let $\tau_{out} = file$ (see Table 4.2). Graph walks of length 2 were applied.

The results show that the graph walk using uniform weights (Gw:Uniform) 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 4.7 (Gw:Learned). High weights were assigned to the *has-subject-term* edge type (and its inverse), where applicable; and to the edges *sent-from* and *sent-to,* in all of the experiment's configurations.

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 TF-IDF baseline, as well as than 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 [20].

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.

4.4.3 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. 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.

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

Table 4.8: Meeting attendee prediction corpus and dataset details.

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 names, 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 4.8. 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 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 a bias towards specific repetitive examples, the constructed dataset includes only 11 examples, manually selected as having distinct attendee lists[4]. 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 email-addresses refer to a group, e.g., members of the RADAR project. In addition, some addresses may be rarely used or obsolete. In the experiments conducted, we consider *all* email-addresses

[4] We also required that the meetings relate to persons that are likely to appear in the email corpus.

	MAP	**Accuracy**
Meetings		
String sim.	0.24	0.33
Gw: Uniform weights	0.58^*	0.67
Gw: Learned weights	0.65^*	0.67
Gw: Path constrained	$\mathbf{0.68}^*$	**1.00**
Gw: Reranked	0.59^*	0.67

Table 4.9: Meeting attendees finding results

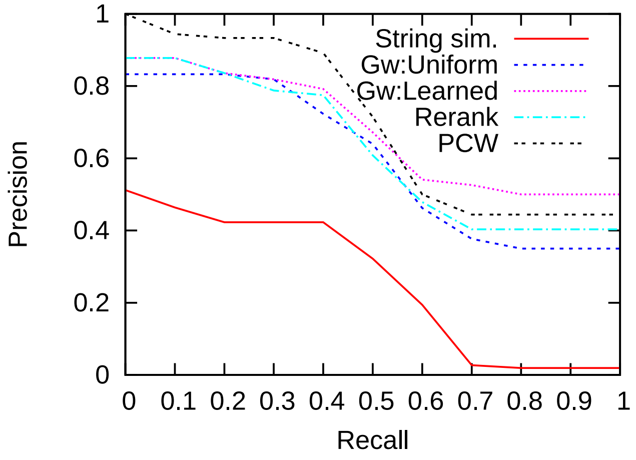


Figure 4.3: Meeting attendee prediction results: 11-point Precision-recall curve.

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 were considered per user; these gave similar results [94].

Overall, the experimental corpus and dataset are modest in size. In general, the framework should benefit from larger corpora that may be less sparse in text and having 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 these 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 [29] 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, given the meeting notes.

The results of applying the string matching approach are given in Table 4.9 in terms of MAP and accuracy. Since the number of correct answers varies to a large extent between examples we use an 11-point interpolated precision-recall curve averaged over all examples for evaluation. The precision-recall curve is shown in Figure 4.3. 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 text associated with the meeting entry 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 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 $\xrightarrow{mtg-has-term}$ term $\xrightarrow{as-term^{-1}}$ person $\xrightarrow{alias}$ email-address
- meeting $\xrightarrow{mtg-has-term}$ term $\xrightarrow{has-term^{-1}/has-subj-term^{-1}}$ message $\xrightarrow{sent-to/from-email}$ email-address
- meeting $\xrightarrow{mtg-on-date}$ date $\xrightarrow{on-date^{-1}}$ message $\xrightarrow{sent-to/from-email}$ 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.

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

Table 4.10: Alias finding corpus and dataset details.

Learning

As shown in Table 4.9 and Figure 4.3, 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. Edge types that were assigned high weights by weight tuning are, for example, *as-term-inv* and *alias*. Reranking gave similar results as 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.

4.4.4 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 consist of a *person* node, or the corresponding name *term* (see also Section 4.2). In the experiments conducted, we consider the latter settings.

Datasets

We evaluate the task of alias finding using two corpora, as detailed in Table 4.10. For both corpora, we use a manually labeled list of email-address aliases per person. All of the examples considered refer to individual users (as opposed to 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, we have shown the settings in which the query included the person's full name represented as terms to be an easier problem [94]. In addition, querying by the person's first name only 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.

	MAP	**Accuracy**
Meetings		
String similarity	0.55	0.67
Gw: Uniform weights	0.61	**0.83**
Gw: Learned weights	0.55	0.67
Gw: Path constrained	**0.68**	**0.83**
Gw: Reranked	0.59	**0.83**
Personal		
String similarity	0.54	0.69
Gw: Uniform weights	0.72	0.77
Gw: Learned weights	0.73	0.77
Gw: Path constrained	**0.74**	**0.96**
Gw: Reranked	0.63	0.85

Table 4.11: Alias Finding Results

Baseline: String matching

As a baseline, we use here the string matching approach described earlier (Section 4.4.3). The results of applying string matching are given in Table 4.11 in terms of MAP and accuracy, and in Figure 4.4, 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 (Table 4.1), 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 linked over two *string similarity* symmetrical directed edges. In general, graph walks are expected to be effective in realizing co-occurrence 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

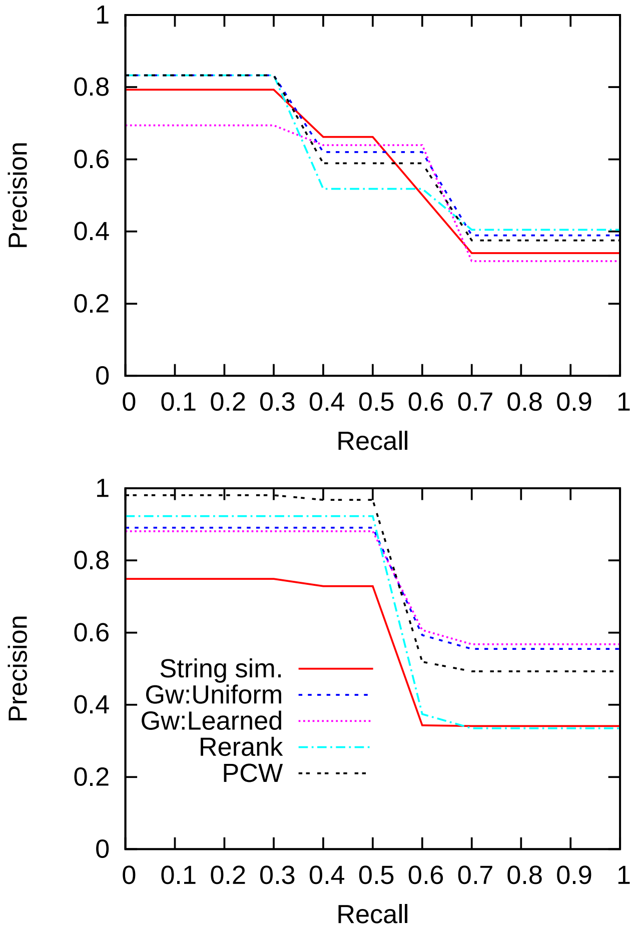


Figure 4.4: Person to email-address mapping: Precision-recall curve

corpora. It results in MAP of 0.61 and 0.72 for the Meetings and the 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 $\xrightarrow{as-term^{-1}}$ person $\xrightarrow{alias}$ email-address
- term $\xrightarrow{has-term^{-1}/has-subj-term^{-1}}$ message $\xrightarrow{sent-to/from-email}$ email-address
- term $\xrightarrow{has-term^{-1}/has-subj-term^{-1}}$ message $\xrightarrow{sent-to/from}$ person $\xrightarrow{alias}$ 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.

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 reflects the information carried in the full paths traversed.

The differences between the methods were not found to be statistically significant.

4.5 Effect of Query Length

The tasks reviewed in Section 4.4 are modeled as queries that contain a small number of term nodes, a message, or a meeting node. This section reviews a couple of additional tasks, where the query includes a *folder* node. Each folder node is associated with many messages, such that probability spreads rapidly in the network. Specifically, the tasks discussed include the prediction of persons future involvement in an ongoing project (represented by a folder), and message foldering and tracking. We first present the experiments and their results. This section concludes with a discussion.

4.5.1 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. In the experiments, 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.

	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 4.12: Activity-person prediction corpora and dataset details.

Datasets

We evaluate this task using the saved email of four different Enron employees. Each of the individual mailboxes was truncated at 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 maintained. (This allows both efficiency in maintaining 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 4.12.

The dataset consists of seven folders, drawn from the described corpora, that are associated with project activity. Table 4.12 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 for each corpus 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. The number of correct answers for each corpus is given in the table, in the ‘targets’ column.[5]

Folder	**Cosine**	**DP**	**Gw:Uniform**	**Gw:Learned**	**PCW**
London	0	0	**0.05**	-	-
Europe	0	0	**0.12**	**0.12**	0.09
Portland	0.28	0.28	0.12	0.12	**0.36**
East-power	**0.07**	0	0	0	0
Mexico	0	0	0.11	0.11	**0.22**
CES	0	0	0.08	**0.15**	0
Wellhead	**0.22**	0	0	0	0

Table 4.13: Person-activity prediction results: Recall at rank 20

Baseline: TF-IDF

A different approach previously suggested in the personal information management domain is to model the various entities using word distributions [98]. According to this approach, the word distribution assigned to an *email-address* entity, for example, reflects the word frequencies in the messages sent by, and received by that email-address. We represent *folder* (and more generally, *activity*) entities in the same fashion; that is, as an average of the word frequencies of the messages associated with the folder. Inter-entity similarity can then be estimated by the dot product or cosine similarity between pairs of word distribution vectors. In our experiments, we match the TF-IDF weighted word distribution representing the folder with the distributions that describe each of the relevant email-address entities.

The results are shown in Table 4.13. It is reasonable that the recommending system will present a relatively short ranked set of email addresses (or names) to the user. We therefore evaluate performance in terms of recall at the top ranks, considering the top 20 rated nodes. Recall is measured by the ratio of email-addresses retrieved at the top 20 ranks that indeed appeared in the folder later in time. For example, there are 25 persons (email-addresses) known to get associated with the "Portland" folder after the considered snapshot date (Table 4.12); if exactly 7 correct email-addresses are included in the top 20 email-addresses retrieved given the folder "Portland" as a query, then the recall at the top 20 ranks is 0.28 .

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

[5] The number of valid predictions for each corpus is therefore the number of 'persons', where the number of 'known' entities is subtracted.

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 4.13, the graph walk method gives better predictions for four out of the seven folders compared with cosine similarity, which is preferable for the three remaining folders.

Learning

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

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 constrained walks performance was better than its cosine similarity counterpart. Highly predictive paths in the constructed path-tree included:

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

This shows that both social network information and textual evidence were found informative for this task.

Overall, while the task of predicting future involvement of persons from the enterprise in an ongoing activity is challenging, the results indicate that several correct predictions are likely to be included in short lists introduced to users. 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

	corpora				**dataset**		
	files	folders	nodes	edges	folder	files	examples
Kaminski V.	859	33	8925	81728	Conferences	80	10
					London	80	10
					Resumes	80	10
					Stanford	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 4.14: Message foldering and tracking: corpora and dataset details.

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.

4.5.2 Message Foldering and Tracking

In the message foldering task, we are interested in ranking existing user-created *folders* by their relevancy to a given *message*. We also consider the inverse problem: ranking untagged *messages* by their similarity to a given *folder*. The folders considered denote user activities. (Other eclectic folders, such as "sent-items" folder, etc. are ignored.) A more detailed description of these tasks is given in Section 4.2.

Datasets

We consider the same corpora constructed for the task of activity-person prediction (Section 4.5.1). 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 consider folders that reflect a project or recurring activity. The modified corpora statistics for the four users are presented in Table 4.14.

Overall, the dataset includes 10 different folders for all users. For each of these folders, we consider the consequent "future" messages in each folder (right past the corpus snap-

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 4.15: Message foldering results: MAP

shot date). In the graph representation, the test messages are linked to all of the relevant entities in the graph (persons, email-addresses, terms etc.). These messages, however, are disconnected from the associated folder. Our goal is then to recover the message-folder links. 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 4.14 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. The number of test messages per folder is given in the ‘examples’ column. (In foldering, each test message is represented by a separate query; in message tagging, each query represents a folder, and the test messages are the relevant answers.)

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 associated with the folder. 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 4.15 in terms of mean average

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 4.16: Message tracking results: MAP

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 4.16. 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 long messages are ranked higher with the dot-product measure, whereas length normalization in the cosine measure accounts for this factor.

Graph walks

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

As shown, the performance of graph walks for the foldering task (Table 4.15) is comparable to cosine similarity in most cases, and somewhat worse in a few instances. The graph walk results are superior to dot-product similarity.

In the case of message tracking, however, the results are quite different (Table 4.16). The graph walk gives poor performance compared with the cosine similarity measure. Interestingly, the graph walk results are comparable to dot-product similarity for most examples.

A possible explanation for the major difference in performance of the graph walk between the two tasks is as follows. Lets assume that the folder “recruiting” 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. In the next steps of the walk, probability mass is conveyed from these 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 probability 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. In the inverse task, 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) message node, distributes its probability to the terms it contains. This means that frequent terms, that appear in a large number of messages, are likely to be assigned high weights. Similarly, the graph walk is biased in favor of long messages. This suggests that beyond stop word elimination (which was already performed in constructing the graph), downweighting high-degree nodes may be beneficial in this case.

Learning

In 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 overall minor improvements of the results for both foldering and message tracking. 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 $\xrightarrow{in-folder^{-1}}$ message $\xrightarrow{sent-to-email/sent-from-email}$ email-address $\xrightarrow{sent-from-email^{-1}}$ message

- folder $\xrightarrow{in-folder^{-1}}$ message $\xrightarrow{has-term}$ term $\xrightarrow{as-term^{-1}}$ person $\xrightarrow{sent-from^{-1}}$ message

- folder $\xrightarrow{in-folder^{-1}}$ message $\xrightarrow{sent-to-email}$ email-address $\xrightarrow{alias^{-1}}$ person $\xrightarrow{sent-from^{-1}}$ message

- folder $\xrightarrow{in-folder^{-1}}$ message $\xrightarrow{has-term}$ term $\xrightarrow{has-term^{-1}}$ message

These paths include social network connectivity (see the first three patterns), as well as shared content evidence (the last pattern above).

4.5.3 Discussion

In this section, we reviewed the graph walk performance for a set of activity centric tasks. Performance was found to be inferior to TF-IDF based entity similarity for two tasks, where the queries included *folder* nodes. Each folder in the experimental corpora is highly-connected, being linked to a large number of *message* nodes. The effective query therefore corresponds to a wide distribution over the graph nodes for these two tasks. In such circumstances, the graph walk tends to reflect the global structure of the graph, rather than local relatedness phenomenon [131, 104]. In other words, the bias towards central nodes in the graph is more prominent once probability is already spread in the graph. Thus, nodes that are highly-connected in the graph are assigned high importance, and the results get somewhat disassociated from the query. While Personalized PageRank applies exponential decay with the distance from the query nodes in order to maintain the association to the query, a large query corresponds to a relatively large subgraph as its starting point.

In general, a modified edge weighting scheme may improve performance. For example, Tong et-al [132] have proposed to normalize the graph transition matrix based on node degree, such that high-drgree nodes are more strongly penalized (see Section 2.4.1). Other weighting schemas, such as TF-IDF node weighting embedded in the transition matrix, may be useful. We leave this for future work.

4.6 Related Work

In this section we first discuss works that are related to personal information processing in general. We then proceed to reviewing previous work relevant for each of the tasks studied.

There has been an increased interest in 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 goal 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 enterprise using email resources [8, 106], recommendation of recipients for a given message [22, 103], identifying possible email leakage to wrong recipients [21], and more.

Naturally, email, as well as other entities at the work station such as documents, calendars and webpages, correspond to different facets of underlying user *activities*. Research has been conducted concerning activity-centered collaboration [53], and 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 [81, 20]. The problem of classifying email messages into activities has been studied as well [42]. Mitchell et-al [98] 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 processes personal information as semi-structured data, 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-occurrence data [10]. Another work [2] 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 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 this case study.

Person name disambiguation. The task of person name disambiguation has been studied in the field of social networks and applied also to email data (e.g., [88, 38]). Diehl et-al have suggested to perform name disambiguation in email using traffic information, as derived from the email headers [38]. 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 differs from theirs mainly in that it allows the 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 (thus bounding recall), we use the graph walk to rank all of the graph nodes using network topology.

Recently, Elsayed et-al [43] proposed a generative model for resolving name mentions in email. The model suggested can be thought of as a language model over a set of personal references. They annotated a subset of the Enron corpus with person names, mapping them to their email-addresses and nicknames (as deduced from email salutations and signatures) to learn these preferences. In the experiments conducted, persons that have a first name or nickname that exactly matches the name mention are considered as candidates for matching. 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 corresponding 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 with a similar topic that were delivered around the same time. Each name mention is then resolved based on the learned person–email-address–nickname mappings. Overall, the described approach is 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. Lewis and Knowles [84] considered email threading as a retrieval problem. They applied text matching methods to the textual portions of messages. More specifically, they suggested 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 [142] 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. In contrast with these works, the graph walk framework is generic, and does not rely on manually encoded world knowledge. Instead, learning allows to adapt the general graph walks to the special characteristics of the threading task.

In general, we note that the threading information learned in the graph can be useful in learning other related sequential information. For example, thread information was also used in the chaining of sequential speech acts [20].

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 [120, 97]. 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 [89] 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 [98] 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 relations, and optimize performance for the task. 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 [62], and attempted to combine social network information and string similarity measures for this task [63]. Our approach allows integration of header information and string similarity measures, as well as email content and time in a unified framework.

Message foldering and tracking. The foldering task [61] has been considered in the past. Previous works applied algorithms such as TF-IDF [119]. Other works have addressed foldering as a classification problem, with the goal of classifying an email message to a single relevant folder. Good classification results were obtained using Naive Bayes, MaxEnt and SVM [9]. In addition, a graph matching technique was applied for classifying incoming messages into folders [2]. Our approach in foldering is related to semi-supervised classification is graphs, where probability is distributed randomly in the graph, reaching nodes that denote classes (folder) with different intensities (see Section

2.4.4). 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. Several researchers have employed a language modeling approach for this problem [8, 106]. 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 [22]. In the settings considered, however, the 'query' pertains to a single message, and partial information about the recipients is available. In the task defined as predicting future person's 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, adapting a model for this specific problem. On the other hand, the graph framework does not apply fine text processing compared with language models.

Liben-Nowell and Kleinberg [85] investigated a generic related problem, predicting the appearance of new interactions in an evolving social structure. Given a snapshot of a social network at a given time, they were interested in accurately predicting the edges that would be added to the network during a subsequent time interval. The approaches used in this work included measures for analyzing node proximity in networks, assuming that new links are hinted at by the topology of the network. Empirical evaluation in the domain of collaboration between researchers showed that for a sufficiently narrow set of researchers considered, e.g., researchers who publish in the same conferences, almost any author can collaborate with almost any other author, and there seems to be a strong random component to new collaborations. Specifically, the link prediction methods applied could not beat random guessing by a factor of more than about seven. In our work, we consider structured networks, where relations are typed and directed. We study a different problem, and find our results to be encouraging.

4.7 Summary

We have presented a schema for representing personal information with a graph. In this schema, meta-data that is available as structured fields in the header of email messages and meeting entries is included as typed entities connected with labeled and directed edges. Associated text is represented as a bag-of-word in the graph.

We have shown that different tasks in the personal information management domain can be phrased in terms of entity similarity and addressed as queries in our framework. Graph walk performance on these tasks was evaluated using various corpora, including corpora extracted from the public Enron email collection. We have shown that graph walks yield preferable performance to alternative methods in most cases.

A major advantage of the graph walk approach is that it integrates social information, content and temporal evidence in evaluating entity relatedness. The tasks evaluated demonstrate additional strengths of the graph walk framework. In particular, the *person name disambiguation* task is an example of contextual search, where *message* information that is readily available from the user's environment is used to enhance the query. This information was shown to assist in person disambiguation. In other tasks, we have demonstrated the modularity of the graph representation, where variations in the graph layout are easily accommodated. The task of *predicting meeting attendees*, for example, utilized meeting and email messages interactions in the graph. Similarly, folders that denote user activities were included in the graph for the *foldering*, *message tracking* and *activity-person prediction* tasks. In the task of *alias finding*, we also have added string similarity edges to the graph. Nevertheless, the underlying graph was not reconstructed per task, as the graph representation is general and is independent of the tasks performed.

The basic graph walk models co occurrence and graph topology information in generating initial results. Learning, using relatively small sets of labeled examples and a set of generic features, allowed us to further optimize the graph walk results per task. We have shown that the different learning methods improve the graph walk performance in most cases. While weight tuning proved to be usually effective, reranking and the path constrained walks, which use global information about the graph walk, yielded superior results than weight tuning for some of the tasks. For instance, we found that high-level information about the edge sequences traversed in the graph walk was very informative for the *threading* task, *alias finding* etc. In addition, reranking using specialized string similarity features improved the results for the *person name disambiguation* task.

We found that 'long' queries, which refer to a large set of nodes in the graph, are challenging in this framework, as the bias towards highly-connected nodes embedded in the random graph walk paradigm is more prominent in these settings. General techniques that penalize high-degree nodes are expected to improve the graph walk performance. In addition, we are interested in exploring ways of integrating this approach with language-modeling approaches for document representation and document retrieval. Formally, this can be done straightforwardly by appropriately defining $\Pr(d \xrightarrow{\ell} t|\ell)$ for the edge type $\ell =$*has-term* to correspond to the probability for t assigned by a language model for the document d, and by defining $\Pr(t \xrightarrow{\ell} d|\ell)$ for the edge type $\ell =$*in-file* to reflect the probability of the document d given the query term t.

Another venue of future work is modeling of a timeline in this framework. It is straightforward for example to add edges between date nodes according to time proximity, thus modeling a timeline as required additional graph walk steps.

Chapter 5

Case Study: Applications of Parsed Text

In the PIM graph schema presented in Chapter 4, text was represented as a bag-of-words. It is desirable, however, 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 structures. Given this representation, we will be interested in processing tasks that involve word similarity in the graph. Previous works have applied graph walks to draw a notion of semantic similarity in graphs, which were carefully designed and manually tuned, and included WordNet [47] inter-word relations [136, 33, 65]. While these and other researchers have used lexicons such as WordNet to evaluate similarity between words, there has been much interest in extracting a word similarity measure directly from text corpora (e.g., [127, 101]). We suggest processing dependency parse trees within the general framework of directed labeled 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 measures.

The graph representation and the set of graph walk similarity measures are empirically evaluated on the task of *coordinate term* extraction,[1] from small to moderately sized corpora, where we compare them against vector-based models, including a state-of-the-art syntactic distributional similarity method [101]. 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

[1] In particular, we focus on the extraction of named entity classes.

word synonyms from a corpus, which are consistent with these findings.

Below we first outline our proposed scheme for representing a dependency-parsed text corpus as a graph (Section 5.1). We next discuss the coordinate extraction task, and how it can be processed as queries in the graph (Section 5.2). The settings and results of an empirical evaluation are detailed in Sections 5.3 and 5.4. This chapter concludes a review of related work (Section 5.5) and a summary.

5.1 Parsed Text as a Graph

In recent years, there has been an increasing interest in using dependency parses for a range of NLP tasks, including machine translation, relation extraction and question answering (e.g., [127, 139]). 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 labeled 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 5.1. 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, as well as its position within the sentence. Word mentions are 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 labeled 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 $\xrightarrow{mention}$ girls1 $\xrightarrow{nsubj}$ like1 $\xrightarrow{as-term}$ like $\xrightarrow{mention}$ like2 $\xrightarrow{nsubj-inverse}$ boys2 $\xrightarrow{as-term}$ boys , and

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

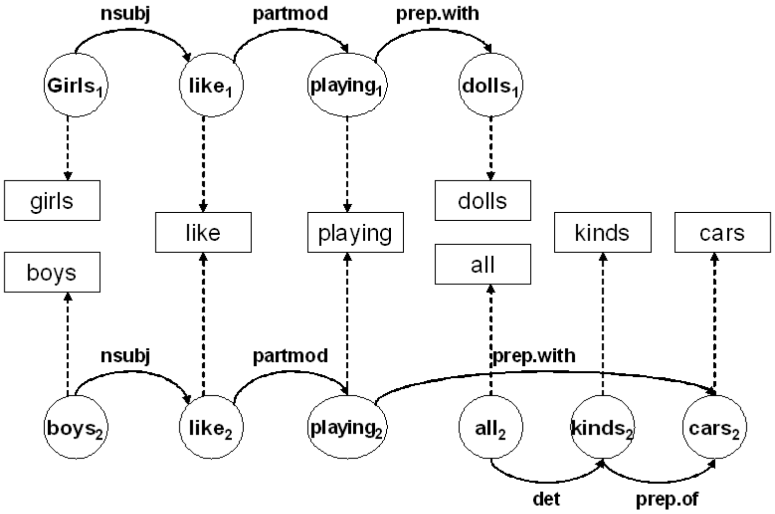


Figure 5.1: A joint graph of dependency structures

Intuitively, in a graph representing a large corpus, terms that are more semantically related will be linked by a larger number of connecting paths. In addition, shorter connecting paths may be in general more meaningful. The graph walk paradigm addresses both of these requirements. Further, different edge types, as well as the paths traversed, are expected to have varying importance in different types of word similarity (for example, verbs and nouns are associated with different connectivity patterns). These issues are addressed using learning.

5.2 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 considered to be coordinate terms. For instance, *New-York*, *Paris* and *Rome* 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 [59], 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 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 NE chunks, it is possible to filter the query results by node type $\tau =$"named entity". We apply this filter in the experiments. Otherwise, results can be filtered based on by part-of-speech tags, capitalization patterns etc.

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

General word *synonym extraction* is another 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 extract synonymous words. The query distribution V_q will consist in this case of the term of interest. The objects retrieved can be of a general type, $\tau =$*term*. It is also possible to query for only *nouns*, *verbs* etc., according to the query word and the user intentions.

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 5.1: Corpus statistics

5.3 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, and is fully annotated with named entity tags.

Associated press (AP). Another corpus used consists of articles of the Associated press, extracted from the AQUAINT corpus [14]. The AQUAINT corpus includes automatically generated, noisy, named entity tags.

British National Corpus (BNC). Finally, we use a subset of the British National Corpus [19]. 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 [37].[2]. Statistics of the experimental corpora constructed and their corresponding graph representations are given in Table 5.1. The MUC corpus is relatively small, containing about 140 thousand words. A corpus constructed that included the union 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, including a total of about 1.3 million words.

Additional details regarding the considerations in corpus construction are provided in the experimental settings description.

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

5.4 Experiments and Results

We evaluate performance using graph walks with *uniform* edge weights Θ, i.e., $\theta_\ell = \theta_{\ell'}, \forall \ell$, and also for graph walks where the edge weights have been tuned. We apply the same weight tuning procedure described in the previous chapter. Reranking is applied on top of the modified graph walk results in this case, using the learned edge weights. Path trees were learned using the top positive, and the top negatively labeled 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. Performance was evaluated on the separate test examples. In all of the experiments reported in this chapter we applied a reset 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 [55, 86, 101]. In most cases, these models construct vectors to represent each word w_i. Every element in the vector of w_i corresponds to particular "context" c, representing a numeric 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* [86] that was designed for this domain.

Recently, Padó and Lapata [101] have suggested an extended syntactic vector space model called *dependency vectors* (DV). In this model, 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 (i.e., the syntactic dependency patterns connecting the word mentions). They considered two different context-based weighting schemes: a *length* weighting scheme, assigning lower weight to word co-occurrence over longer connecting paths (computed as inverse of path length); and an *obliqueness* weighting hierarchy [73], assigning higher weight to paths that include grammatically 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 showed improvements over several previous vector space models.

In the experiments, we therefore 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 traditional approach, where text is processed as a stream of words rather than as 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 [101].

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[3], where we converted the underlying syntactic patterns to the Stanford dependency parser conventions. The parameters of the DV method were set based on a cross validation evaluation (using the city name extraction train set queries, and 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 method.

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, we present the experimental evaluation of the coordinate term extraction and the word synonyms extraction tasks in detail (Sections 5.4.1 and 5.4.2, respectively). For every task, we describe the datasets constructed, the results of the vector-space models, and the results of graph walks and learning in the graph walk framework. Finally, we draw conclusions from the experiments regarding the framework.

5.4.1 Coordinate Term Extraction

In the coordinate extraction task, queries include a small number of seed examples representing a named entity class of interest. We require the retrieved nodes to be of type $\tau =$*named entity*, using the available corpus annotations. The empirical evaluation includes the extraction of named entities that are instances of *city* and *person* names.

In what follows we describe the experimental datasets, and the experimental settings for each of the evaluated approaches. We then briefly review the models generated by the learning methods, present the results and discuss the observed trends.

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

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.[4] 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 labeled 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 corpus, by adding to the MUC-6 corpus parsed articles extracted from the AQUAINT corpus (MUC+AP). The AQUAINT corpus has been annotated with named entities automatically, so it includes noisy tags; nevertheless, we process queries in the this graph in the same manner described above. The same queries were applied to both corpora.

Experimental Setup

Details regarding the experimental settings of each of the approaches considered are given below.

Vector-Space Similarity. We evaluated the co occurrence model (CO) and the dependency vector model using the test queries on both corpora. Limiting the considered set of candidates to named entities allows us to reduce the size of the co-occurrence matrix maintained by the vector models, thus overcoming memory requirement constraints. In the larger MUC+AP corpus, however, the number of candidates that need to be evaluated is very large (Table 5.1). We therefore show the results of applying the 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, 1,000 candidates are ranked overall.)

Graph Walks. We first set the length of the graph walk k, using cross-validation over the training queries using varying walk lengths. We found that beyond $k = 6$ improvements in mean average precision were small. We therefore set $k = 6$.

Weight tuning. Weight tuning was trained using the training set and two dozens of target nodes for each task.

Reranking. In reranking, we evaluate specialized feature templates in this domain, as

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

follows.

Edge label sequences. These features indicate whether a particular sequence of edge labels ℓ_i occurred, in a particular order, within the set of paths leading to the target node z_{ij}. Here we consider full paths from query to target. (However, we removed the edges *mention* and *as-term* from the feature description, such that the remaining edge sequences are mostly bigrams.)

Lexical unigrams. The lexical unigram features indicate whether a word mention whose lexical value is t_k was traversed in the set of paths leading to z_{ij}.

For example, for the query term "girl" in the graph depicted in Figure 5.1, the target node "boys" is described by the features (denoted as *feature-name.feature-value*): *sequence.nsubj.nsubj-inv* (where *mention* and *as-term* edges are omitted) , *lexical.'"like"* etc. In addition, we applied the *Source-count* feature (see Section 3.3.2).

We set a count cutoff (of 3) to the reranking features 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. (We computed the features independently of the graph walk, using the path unfolding procedure, as described in Section 3.3.3.)

Path Constrained Walk. Finally, path-trees were constructed using the top 20 correct nodes and 20 incorrect nodes retrieved by the uniformly weighted graph walk. Labels indicating 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.

Learned Models

Following is a short description of the models learned in weight tuning, reranking and the path constrained walks.

Applying weight tuning, 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 align well with the linguistically motivated weights of the dependency vectors model.

High weights were assigned by the reranking model in the city name extraction task to lexical features such as "based" and "downtown"; and to edge bigrams such as "prep-in-Inverse→conj-and" or "nn-Inverse→nn".

In the path trees constrcuted, positive highly predictive paths included many symmetric paths. For example, in the city extraction task, predictive patterns included the following symmentric paths:

...→conj_andInverse → ...→.conj_and → ...

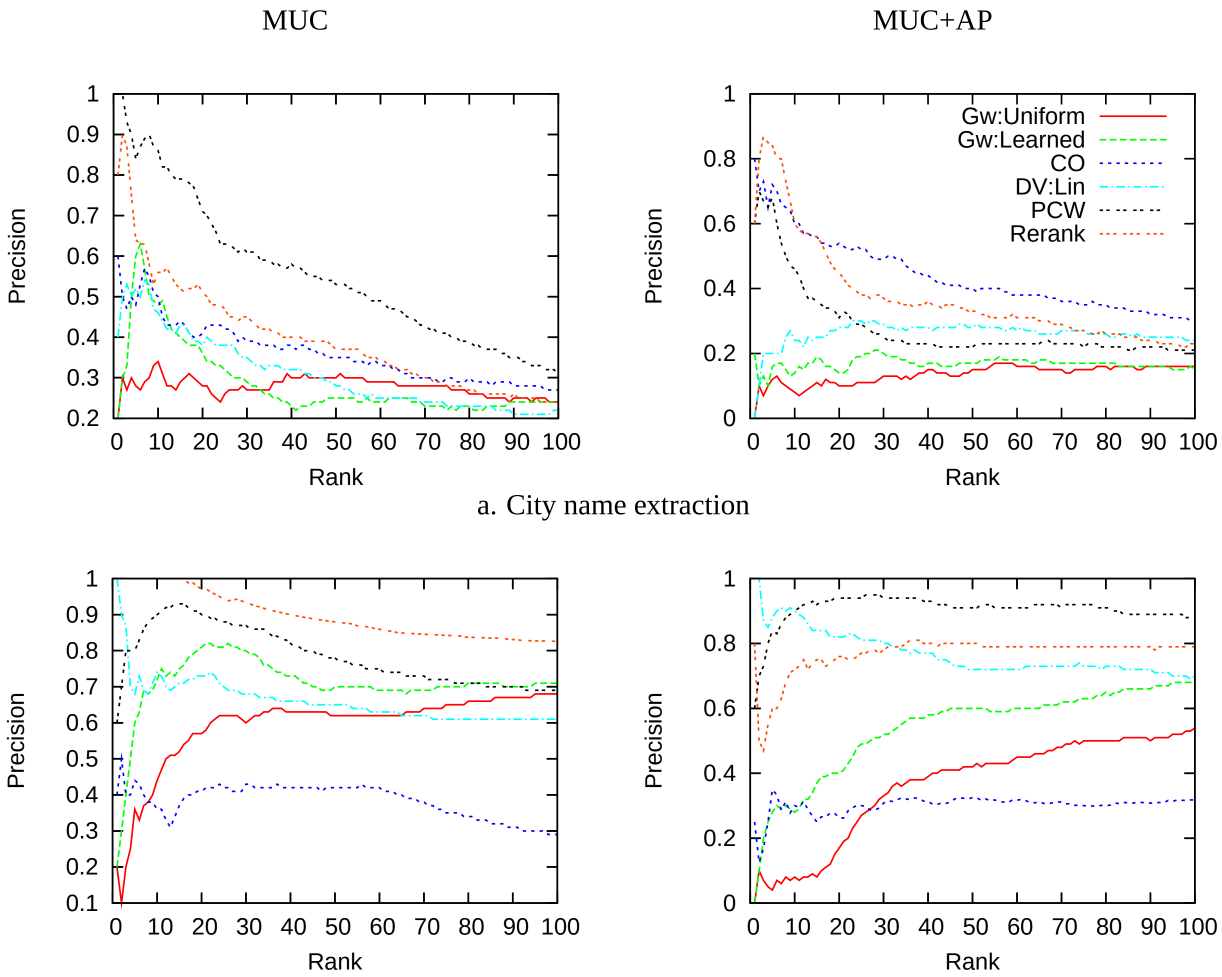
...→prep_inInverse → ...→.prep_in → ...

Results

Figure 5.2 gives results for the city name (top) and the person name (bottom) extraction tasks. The given curves show precision as a function of rank in the ranked list, down to rank 100. (We hand-labeled all the top-ranked results as to whether they are city names or person names.) The figure shows the results for the MUC corpus (left), and for the MUC+AP corpus (right).

The figures included the resulting curves using the co-occurrence model (CO), applying a cosine similarity; and using the syntactic vector-space DV model, where the Lin similarity measure was applied (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. The syntactic dependency vectors model, on the other hand, 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 in the case of city names. For example, the *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 [130]. Overall, the vector-space models demonstrate good performance for the larger MUC+AP corpus, but only mediocre performance for the smaller MUC corpus.

The results of applying graph walks with uniform weights are shown in Figure 5.2 (Gw:Uniform). 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 co occurrence model, but lower accuracy at the top ranks. Studying the results, we found that due to the exponential decay over path length embedded in the graph walk paradigm, named entities linked to the query nodes over a large number of longer paths were often ranked below other nodes connected to the query over few short paths. In the problem settings considered, however, some long connecting paths are more meaningful than shorter arbitrary connecting paths. (For example, in Figure 5.1,



a. City name extraction

b. Person name extraction

Figure 5.2: Test results: Precision at the top 100 ranks, for the city name extraction task (top) and person name extraction task (bottom).

we consider “girls” to be more similar to “boys” than to the word “playing” and the path between “girls” and “boys” to be more significant, although the path that connects “girls” to “playing” is shorter in length.)

We next discuss learning results, and show how learning allows to promote nodes that are distant, yet connected to the query over a large number of meaningful paths. Figure 5.2 shows the precision-at-rank curves for the graph walk method using the learned weights (Gw:Learned), the same graph walks with reranking (Reranked) and a path-constrained graph-walk (PCW).

Several trends can be observed from the results. First, 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 can only consider 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 thereby eliminate errors 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 the path constrained walks 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. The difference in performance between the graph-walk and vector based approaches narrows with the size of the corpus, as the vector based methods improve given more statistics.

5.4.2 General Word Similarity

In the previous section, we used the framework to extract words that are related by being coordinate terms. In particular, our experiments focused on named entities. In this section, we consider the extraction of synonyms, given general words. For example, given the query $V_q = \{term\text{=“}movie\text{”}\}$, we are interested in retrieving list of entities of type $\tau = term$ (or $\tau = noun$), and expect to get synonymous words, like *film*, appear at the top of the list retrieved.

Below we describe the experimental datasets and a set of preliminary results.

Datasets

We collected pairs of word synonyms from teaching materials for foreign students. We then constructed an experimental corpus by extracting sentences that contain these words. Specifically, we extracted all of the relevant sentences from the BNC corpus. (The number of extracted sentences was bound to 2,000 per word.) For infrequent words, we extracted additional example sentences from the AP corpus. (Sentence count was complemented to 300 per word, where applicable.) The constructed corpus, BNC+AP, is based on 1.3 million words overll. The corpus statistics are given in Table 5.1.

We distinguish between nouns, adjectives and verbs. Table 5.2 details the synonyms pairs. 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 as test queries.

Experimental Setup

We applied a simple co-occurrence model (CO) that does not consider the syntactic relations available in the parse structure, using cosine similarity and applying log-likelihood normalization. In addition, we applied the dependency 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).

For scalability reasons, rather than index all of the words in the corpus in the vector space models, we consider the top 400 words retrieved by graph walks (with uniform weights) per query. (That is, the union of the top results per all queries is indexed.)

We applied graph walks, using walk length $k = 6$ and uniform weights (Gw:Uniform). Path-trees were generated using the correct node and the top incorrect node per each of the training queries. (In this case, training data is noisy, as it is possible that the top term retrieved is semantically related to the query word.) We applied a threshold of 0.5 to the path constrained walks (PCW). Finally, reranking was applied using the features described for the coordinate term extraction task. Since the graph walks yielded poor performance in this case (a discussion of the results follows), we applied reranking on top of the results obtained using the path constrained walks in this case.

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

Table 5.2: Word synonym pairs: train and test examples

	CO	DV:Cos	DV:Cos-ll	DV:Lin	Gw:Uniform	PCW	Rerank(PCW)
Adjectives	0.07	0.18	0.21	**0.41**	0.08	0.34	0.34
Nouns	0.05	0.21	0.29	**0.55**	0.01	0.17	0.36
Verbs	0.04	0.13	0.22	**0.45**	0.01	0.40	0.27
All	0.05	0.17	0.24	**0.47**	0.04	0.31	0.33

Table 5.3: General word synonyms extraction results: MAP

Results

Results are presented in Table 5.3 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 [101]. That is, the model of dependency vectors, which considers syntactic relations (DV:Cos) in computing co occurrence weights outperforms the simple co occurrence model (CO). Adding pre-processing of log-likelihood normalization of the statistics improved the performance of the dependency vectors model further. Lastly, using the Lin measure, specialized for this model for inter-word similarity, in combination with dependency vectors reaches high levels of performance.

As shown in the table, the results of graph walk without learning are relatively poor (these results are comparable to the results of applying cosine similarity). We conjecture that as was the case in coordinate term extraction, here too the preference of the graph walks for nodes that are close to the query node is unjustified. The path constrained walks improve performance significantly, by directing the walks to follow meaningful paths. Reranking improved the path constrained walks for the noun queries, but hurt performance for the verb queries. It is possible that specially designed reranking features that are adapted to this problem can improve reranking performance. In general, we observed that the graph walks, as well as the path-constrained walks, are biased towards words that are frequent. It is therefore desired to lower the effect of high node connectivity (using manipulation of the transition matrix, for example [132]; see Section 2.4.2).

Overall, the path constrained walks and reranking give second-best performance in the reported results. The performance of learning are better than the syntactic model of dependency vectors using cosine similarity and log-likelihood normalization, for example. Notice that graph walks were in fact used in the experiments as a preliminary mechanism for retrieving relevant candidates, to be processed by the vector-space models.

5.5 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 [136] constructed a directed graph, where nodes represented words, 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 semantic similarity scores obtained were used for lexical smoothing for the task of prepositional word attachment. Recently, Hughes and Ramage [65] constructed a similar graph, representing various types of word relations from WordNet, and compared the random-walk generated similarity measure to similarity assessments from human-subject trials. Another work in the field of information retrieval 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 [33]. 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 [44, 92]. In these graphs, sentences assigned high centrality scores are considered as salient, and are used to construct an automatically generated summary. Mihalcea [91] applied graph walks also for the task of word sense disambiguation. She used a separate graph per sentence, in which nodes represented the possible WordNet synsets of the contained words, and directed edges between the synset nodes denoted their inter-similarity, weighted by the overlap in the synset definitions 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 parsed sentences have been widely used.

For example, Snow et al [127] 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-appeared, 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 indicated that due to data sparsity, the ratio of relevant sentences in the corpus was low. In contrast, we represent text corpora as a connected graph of dependency structures, where the graph walk traverses both within- and cross-sentence paths.

Dependency paths of individual sentences have been used also for general relation extraction. Culotta and Sorenson [36] 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 dependency 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 [17] 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, where 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.

The graph representation we suggest may be used for general relation extraction. The rich feature set encoded by the described kernels can be represented in the graph (for example, using part-of-speech walkable nodes and edges, WordNet links etc.). Relevant features can also be fed to 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 a direction for future exploration.

5.6 Summary

In this chapter we have explored a novel but natural representation for a corpus of dependency-parsed text, as a labeled 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 the general-purpose graph-walk based query language. Further, we have successfully applied the learning techniques available in the framework. In this domain, path information and other global features proved to be beneficial compared with the local graph walks and weight tuning.

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. We therefore find that the suggested model is suitable for deep (syntactic) processing of small specialized corpora. Preliminary evaluation of general word synonymy gave consistent results.

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.

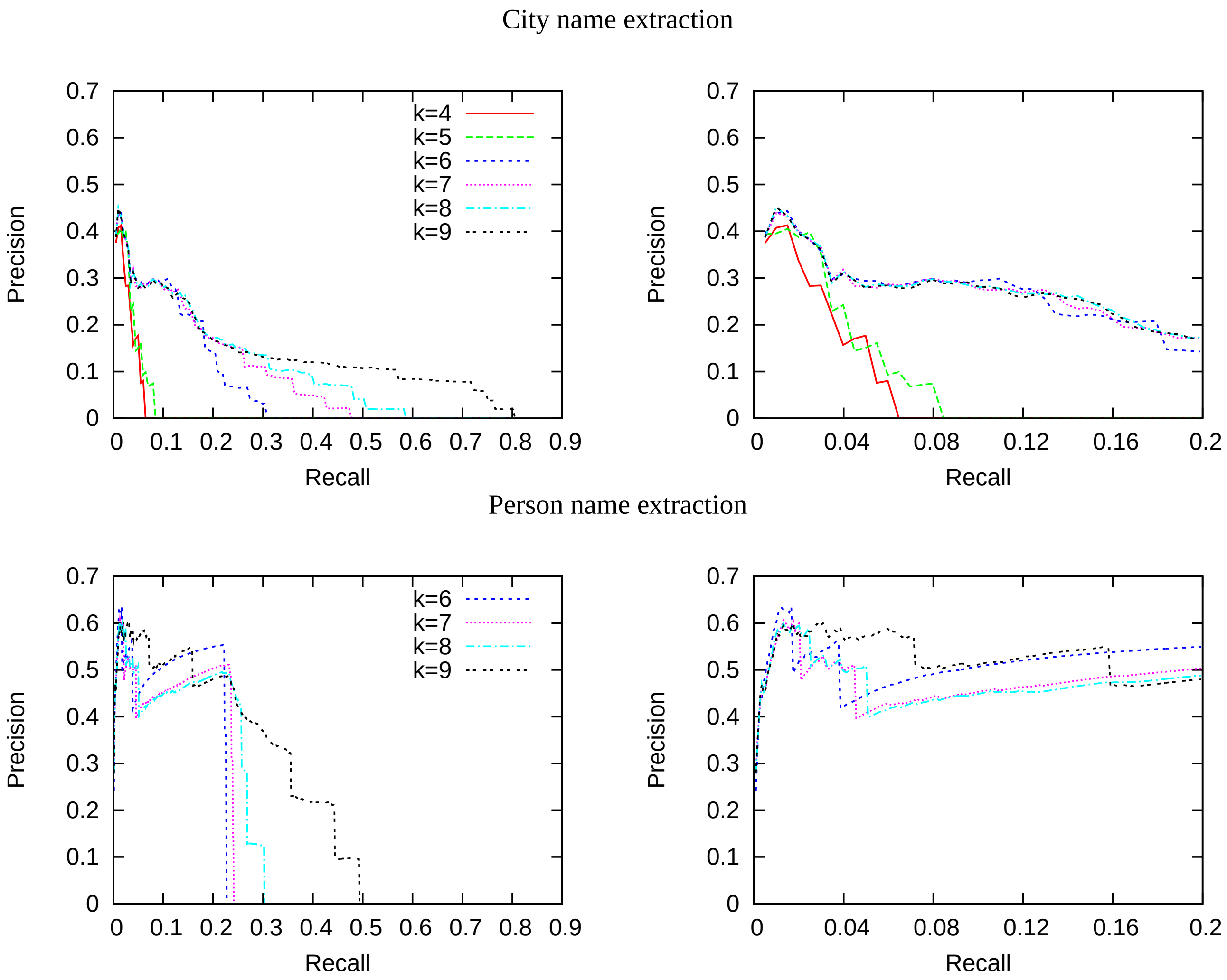
Chapter 6

Design and Scalability Considerations

The test cases studied vary in multiple dimensions, including the graph schema, the tasks considered and the relevant corpora sizes. Given this experimental setup, we are interested in drawing some conclusions regarding high-level system parameters and design choices. In the first part of this chapter, we evaluate the effect of the graph walk parameters, including the walk length, the reset probability and variants of the graph walk schema (Section 6.1). We then discuss learning, where the performance of the individual techniques, as well as their combination, are evaluated empirically (Section 6.2). In Section 6.3 we discuss scalability issues, and detail the experimental query processing times. In addition, the impact of the path constrained walks on the graph walk processing time is evaluated. Finally, a discussion of related research is included (Section 6.4) that focuses on algorithms developed for scaling up Personalized PageRank graph searches.

6.1 Graph walk parameters

The graph walk framework (as described in Chapter 2) includes two parameters: the reset probability γ; and, as we perform *finite* graph walks, another parameter is the length of the walk k. In this section we evaluate empirically the effect of these parameters on performance. In addition, several variants of the graph walk schema (Section 2.2.3) are evaluated.



City name extraction

Person name extraction

Figure 6.1: 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 6.1: Results (MAP) of applying graph walks using uniform edge weights, varying the graph walk length parameter k ($\gamma = 0.5$).

6.1.1 Walk Length

We evaluated the performance of the person name disambiguation, threading and alias finding tasks, using eight datasets in total, for varying walk length k. The results, in terms of mean average precision, are shown in Table 6.1. For every dataset, the best results are marked in bold. As shown, performance for the person name disambiguation task is similar for the management game corpus, given different values of k. In the case of the two Enron datasets, however, the performance on the person name disambiguation task is substantially better for a short walk length ($k = 2$), where it converges to an inferior result for longer walks. In 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 $k = 3$, and performance converges for longer walks. In all of the experiments, performance converged for walks longer than $k = 8$ steps.

These results support our approach of conducting finite graph walks, rather than infinite walks, in two ways. First, Personalized PageRank graph walks converge within a small number of iterations, as indeed shown by the empirical results. Walks over a small number of steps therefore provide a very good approximation of infinite walks. Secondly, and perhaps more interestingly, the results show that limited graph walks give a more accurate similarity measure in some cases. This means that given a strong local indication of inter-entity similarity (as reflected by the set of their connecting paths), propagating similarity in the graph over longer walks may introduce noise to the generated similarity metric. On the other hand, we find that the graph walk should be long enough to allow the traversal of all acyclic meaningful paths from the query to the target nodes. (Obviously, acyclic paths are the shortest way to reach a target node over a particular sequence of edges.) In other words, graph walks that are too short will hurt recall. In the alias finding task, for example, some of the target nodes can be reached in two steps from the query node. However, additional relevant nodes can be reached over paths that are three steps away overall. A walk of three steps therefore gives better performance in this case. Once all of the relevant nodes have been reached for the first time by the graph walk, the benefit in continuing the walk is marginal in the experiments reported here, or hurts performance in some of the cases, as mentioned above.

In another set of experiments, we evaluated the effect of the walk length parameter in the language domain. In this domain, every query is associated with a large number of correct answers (see Section 5.4.1). 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, the MUC corpus is fully annotated, allowing us to show results in terms of a precision-recall curve for this corpus. Figure 6.1 shows the precision-recall curve for the city name extraction task (top) and the person name extraction task (bottom) for the MUC corpus. The left part of the figure show the full curve for the two tasks; the right part focuses on the top of the lists retrieved (down to recall of 0.2).

The top left graph in Figure 6.1 demonstrates clearly that increasing the graph walk length increases recall. For both tasks, short walks where $k <= 4$ yielded poor recall. (For the person name extraction task, recall was near zero, and the corresponding curves were eliminated from the figure.) The reason for the low recall in these cases is that there are relatively few relevant nodes that can be reached over short connecting paths in this domain. For example, the short path “contains – conj-and – containsInv” models a conjunction relation between words appearing in the same sentence. This type of evidence is relatively scarce, and occurs more frequently for city names than for person names in the experimental corpora. The majority of meaningful paths are of length six in the graph. (E.g., the path that models a common direct object argument is of length six, etc.) Increasing the walk beyond $k = 6$ in this domain improves recall, as shown in the figure; however, the additional nodes reached in a longer walk are generally added at the bottom of the retrieved list, due to the exponential decay embedded in the walk.

In conclusion, the length of the graph walk k should allow the graph walk to reach graph nodes over a variety of meaningful paths. As a rule of the thumb, it is recommended that the walk length allows traversal of the full set of (acyclic) connecting paths to a target node. It is straight-forward to tune the walk length parameter using a set of tuning

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

Table 6.2: Results (MAP) of applying graph walks using uniform edge weights, varying the reset probability γ.

examples.

6.1.2 Reset Probability

In all of the experiments reported thus far, the reset probability was set to $\gamma = 0.5$. Table 6.2 shows the results of varying γ for multiple tasks and corpora in terms of mean average precision. The table shows that changing γ has negligible effect on the actual produced rankings. These results are in line with previous findings, showing that while this parameter affects the actual scores assigned to the graph nodes, it does not change the output relative rankings [102].

6.1.3 Graph walk variants

As described in Section 2.2.3, there are several variants of graph walk schemas that researchers have applied in the past in performing random graph walks to extract similarity in graphs. In the experiments conducted so far, we adopted the Personalized PageRank graph walk model, where the outgoing edge weights from each given node were normalized to form a probability distribution. In this section, we provide experimental results using closely related graph walk schemes. More specifically, we evaluate *lazy* graph walks, comparing them to Personalized PageRank graph walks. In the lazy walk schema, the

Corpus	Gw	Gw^{ts}	LGw	LGw^{ts}
Person name disambiguation				
M.Game	0.65	**0.68**	0.66	**0.68**
Sager	0.67	**0.69**	0.68	0.68
Shapiro	0.61	**0.62**	0.61	**0.62**
Threading				
M.Game	0.53	**0.62**	0.53	**0.62**
Germany	0.55	**0.56**	0.55	**0.56**
Farmer	**0.65**	**0.65**	**0.65**	**0.65**
Alias finding				
Meetings	**0.61**	0.60	0.60	0.59
Personal	**0.72**	0.71	0.69	0.70

Table 6.3: Results (MAP) of applying a lazy graph walk variant (LGw), and a different scheme for assigning the random transitions in the graph (un, in superscript).

random walker stays at the current node with probability γ at each step of the walk, or continues to the neighboring nodes with the remaining probability. We evaluate lazy walks using $\gamma = 0.5$. In addition, we evaluate two edge weighting schemas in conjunction with each of the models. In the weighting scheme used in the experiments so far, each outgoing edge from node x is assigned a typical edge weight by its type, and the walker picks an edge at random according to its weight (Formula 2.7). Another possible weighting scheme assumes that the walker first picks an edge type at random, out of the set of outgoing edge types available at each node (S(x), per section 2.2.3); given the edge type, a specific edge is then selected uniformly (Formula 2.8). We refer to the latter schema as *two-stage* graph walk.[1]

Table 6.3 gives the results for the combinations of the two walk paradigms and the two edge weighting schema. The Personalized PageRank graph walks are denoted as ‘Gw’ in the table, and the lazy graph walk variant is denoted as ‘LGw’. The ‘two-stage’ graph walk variant is marked with the superscript *ts*. The edge weights were assigned uniformly in the experiments.

Overall, the different variants have very limited effect on the results. More specifically, the lazy graph walks and Personalized PageRank are shown to generate very similar rankings (as reflected by the mean average precision measure). The two-stage weighting scheme gives noticeably better results for one of the eight datasets.

[1] In our implementation, the weights of the outgoing edge types are normalized at each node x, where the total outgoing weight from x is computed as S(x), defined in Section 2.2.3.

6.2 Learning

As discussed in Chapter 4, several researchers have suggested schemes for adjusting the set of edge weights using hill-climbing methods in the Personalized PageRank settings [39, 100, 4]. We have shown, however, that high-level information, such as the edge *sequences* encountered in traveling from the source nodes to a target node, can be useful in evaluating the node inter-relatedness. 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 3.5.)

The reranking approach parameterizes the graph walk with a set of representative features, which allows one to capture certain global properties of the graph walk. However, this representation loses some quantitative information compared with exact gradient computing. The path constrained walk approach considers global information in the form of edge sequences as well. Compared to reranking, path constrained walks have more impact on the graph walk procedure; on the other hand, a more restricted space of features is considered in this method. In Section 3.5 we discussed the qualitative differences between the learning approaches in detail. Here, we present and discuss empirical comparative results (Section 6.2.1).

It is possible to combine several learning methods. In particular, it is straight-forward 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 6.2.2, we provide empirical results for the utility of these learner combinations.

Finally, Section 6.2.3 discusses the effect of the threshold applied to the path constrained walks in terms of performance (mean average precision), based on a set of relevant empirical results.

6.2.1 Local vs. Global Learning

In this section, we compare every pair of the learning methods in terms of performance. The discussion is based on empirical results of the person name disambiguation and threading tasks, evaluated on six datasets.

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

Reranking vs. weight tuning.

We compare the gradient descent method and reranking as follows. Since the gradient descent algorithm is prone to converge to 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 considered 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 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. (Conversely, in the previous experiments reported in Chapter 4, re-ranking was given the output of the graph walk with uniform weights.) For every example, the top 50 nodes were re-ranked.

We are interested in comparing reranking with edge weight tuning as *alternative* learning methods that improve on graph walk performance. The features that we use for reranking will be therefore derived from the set of paths leading to every candidate node (that is, the same information available to the error backpropagation algorithm is used), describing non-local properties of these paths. In particular, the considered features include *edge label bigrams* and the *source count* feature. As defined in Section 3.3.2, the latter feature indicates the number of different source nodes in the set of connecting paths leading to

Corpus	Gw:Uniform	PCW	Rrk$_{Gw:U}$	Rrk$^{+}_{Gw:U}$	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 6.5: Performance comparison (MAP) of graph walks with uniform weights (Gw:Uniform), path constrained walk (PCW), reranking using edge sequence features (Rrk$_{Gw:U}$) and the combination of path constrained walks and reranking (Rrk$_{Gw:L}$). Reranking using the full set of features is denoted as Rrk^{+}.

the candidate node. The results of reranking using the full set of features designed for the considered tasks (see Chapter 4) are given as well.

Table 6.4 includes mean average precision results for the *person name disambiguation* task (applying the contextual version, where queries consist of file and term nodes) and *threading*, using the relevant corpora. The table includes the evaluation of graph walk with the baseline set of randomized weights Θ^0 (Gw:Random). It also gives the results of applying graph walks with the learned set of edge weights (Gw:Learned); reranking of the graph walk results using the initial edge weights Θ^0, where only path-describing features are used (Rrk$_{Gw:R}$). Results using the full set of reranking features are also included in the table (in the columns Rrk$^{+}_{Gw:R}$ and Rrk$^{+}_{Gw:L}$). Results that were found significantly different, using a two-sided Wilcoxon test at 95% confidence level, are marked with an asterisk, 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 for all datasets using path information only, compared with both the baseline and weight tuning.

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→ 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, yielding superior performance to weight tuning.

In addition, we refer the reader to the results reported earlier for the city name and person name extraction tasks (Figure 5.2). In the 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 all 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 (as 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 that is 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 6.5 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:Uniform). 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 graph walks with uniform weights are marked with an asterisk.

Contrasting the constrained graph walk and weight tuning results (G:Learned in Table 6.4) 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 5.2) – also in this case, the 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 6.5 details the outcome mean average precision of performing path constrained 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) are 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 accommodate 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 (where the reranking featured included both edge sequence features and lexical information) 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. The contribution of the path constrained walks to scalability is discussed later in Section 6.3.

6.2.2 Combining Learning Methods

Reranking can be affected by the quality of the input ranked lists in two ways. First, as reranking 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 where the graph walk rankings are first improved using learning, and then reranking is applied 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, Θ^{G}; then, reranking was applied given these output ranked lists. The results are given in Table 6.4. The combined approach, where reranking uses graph walk describing features, is denoted by '$\text{Rrk}_{Gw:L}$'. Results of reranking using the full feature set are given in the column named '$\text{Rrk}^{+}_{Gw:L}$. 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 6.5 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 ('$\text{Rrk}_{G:R}$ and '$\text{Rrk}^{+}_{G:R}$, in Table 6.4) or uniform weights ('$\text{Rrk}_{G:U}$ and '$\text{Rrk}^{+}_{G:U}$, in Table 6.5). 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.

We conclude that reranking, given ranked lists of sufficient quality and adequate features, is relatively insensitive to small perturbations in the initially ranked lists. However, significantly improving the initial ranking process, and then reranking the output list, is likely to boost the final result.

6.2.3 PCW thresholding

In this section we are interested in evaluating the effect of path constrained walks thresholding on performance. Table 6.6 shows the performance of the path constrained walks in terms of mean average precision for the thresholds of 0 (PCW:0), considering all the paths 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 development examples.

According to the results, a threshold of 0.5 is roughly comparable to a threshold of 0 across 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 6.2 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 a precision-recall

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

curve. 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 earlier (Section 5.4.1), the preference embedded in the graph walk paradigm for nodes that are proximate to the query, is only partially justified in this domain, and therefore incorporates noise at the top of the lists retrieved. This phenomenon accounts for the relative success of path constrained walks in this domain, where this approach directs the graph walk towards longer meaningful paths.

Comparing performance for different thresholds, as depicted in Figure 6.2 shows that applying higher threshold to the path constrained walk leads to improved performance in this case. It is also shown that the higher threshold yield lower overall recall, as expected due to narrowing path coverage.

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.

6.3 Scalability

In general, the paradigm of Personalized PageRank (and its variants) poses a major scalability challenge. While the PageRank algorithm [102] corresponds to a *single* steady-state

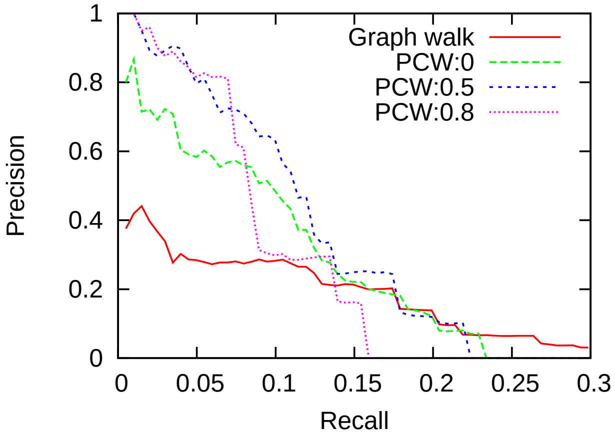


Figure 6.2: Precison-recall performance for city name extraction from the MUC corpus for path constrained walks with varying thresholds, and graph walks with uniform weights.

node distribution, which need only be updated infrequently, 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.[2] Another approach for implementing Personalized PageRank is an 'offline' computation, where personalized views are pre-processed and stored. Pre-processing of all personal views (queries) possible is infeasible due to time and space constraints, as there are $O(2^n)$ different queries possible for graphs with n vertices and the necessary index database size of a fully Personalized PageRank algorithm is $\Omega(n^2)$ [48].

A variety of techniques have been discussed in the literature that address the computational aspects of personalized PageRank. For example, several researchers have suggested efficient indexing of a reduced set of pre-computed 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 6.4.

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

[2] The complexity of online iterative graph walk is O(Ek), where E is the number of graph edges [105].

6.3.1 Implementation Details and Running Times

Table 6.7 shows the average processing time per query for the tasks of person name disambiguation, threading and alias finding, where the walk length varies from $k = 2$ to $k = 8$. The sizes of the experimental corpora range from 6K to 14K nodes, and from 60K to about 200K edges (see Chapter 4). The results were obtained using a commodity PC with 4GB of RAM, where graph information has been loaded to memory. In the experiments, we observed the processing time per query, t_i, averaged over the queries in the test set of each dataset. We obtained five such observations in repeated runs, for which we report the average: $\sum_{i=1}^{5} t_i/5$. In addition, we report the corresponding standard deviation (shown in brackets).

As shown in the table, the average processing times increase with the number of walk steps k, and with the number of graph edges. (For example, 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 equals the union of nodes visited in a separate graph walk from every individual query node. Thus, processing times are longer for the two-node person name disambiguation queries for any given k, compared with the single-node threading queries, both using the management game corpus.

The times given in Table 6.7 are satisfying for real-time applications. In particular, we have shown earlier in this chapter (Section 6.1.1) that short graph walks yield performance that is preferable or comparable to longer walks in this domain. Short walks of 2 or 3 steps require an average processing time of a small fraction of a second.

The option of online computation may be less desirable given larger graphs when real-time response is required. Table 6.8 includes the average processing time (and their standard deviation, in brackets) of graph walks of length $k = 6$ for the city name and person name extraction tasks, in the column named ‘Graph walks’. (Figure 6.3 gives a graphical display of the processing times; see the ‘Graph walk’ point set.) We remind the reader that the each of the queries constructed per these tasks include four graph nodes, compared with a single query node in threading and two query nodes in the person name disambiguation task. As discussed above, this factor adds to the spreading of the graph walk. The average processing times are stated in the table per the relatively compact MUC corpus (which includes about 80K nodes and 245K edges).

To evaluate performance on large graphs, we constructed three intermediate size corpora that include MUC, and a part of the AP corpus: one of these corpora includes a quarter of the AP corpus (MUC+1/4AP); the second corpus corresponds to about half of the AP corpus (MUC+1/2AP); and the third corpus includes MUC and about three quar-

Corpus	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
Person name disambiguation							
M.Game	0.02 (0.01)	0.15 (0.01)	0.44 (0.03)	0.74 (0.04)	1.03 (0.07)	1.38 (0.05)	1.65 (0.13)
Sager	0.08 (0.02)	0.38 (0.01)	0.94 (0.04)	1.52 (0.04)	2.12 (0.06)	2.79 (0.08)	3.29 (0.03)
Shapiro	0.11 (0.01)	0.64 (0.01)	1.56 (0.06)	2.42 (0.02)	3.43 (0.10)	4.27 (0.12)	5.22 (0.10)
Threading							
M.Game	0.03 (0.01)	0.14 (0.00)	0.43 (0.01)	0.77 (0.02)	1.12 (0.03)	1.46 (0.03)	1.94 (0.05)
Germany	0.09 (0.01)	0.33 (0.03)	0.97 (0.02)	1.97 (0.06)	2.70 (0.08)	3.56 (0.26)	4.48 (0.11)
Farmer	0.08 (0.00)	0.38 (0.01)	1.24 (0.06)	2.32 (0.06)	3.45 (0.08)	4.54 (0.20)	5.52 (0.28)
Alias finding							
Meetings	0.03 (0.01)	0.08 (0.02)	0.16 (0.01)	0.30 (0.01)	0.46 (0.06)	0.60 (0.07)	0.70 (0.01)
Personal	0.02 (0.00)	0.11 (0.02)	0.39 (0.01)	0.99 (0.02)	1.66 (0.05)	2.26 (0.10)	2.85 (0.14)

Table 6.7: Average query processing time and standard deviation [secs] per dataset and different walk length k.

ters of the AP corpus. The number of nodes and edges of each corpus are included in Table 6.8.

We limited ourselves to these moderate-sized corpus for reasons of convenience. The implementation used for the other experiments in this thesis is not optimized for memory usage; in particular, the memory required to store each edge is fairly large, including a string to label the edge type and a string identifier for the destination node. The strings used as labels are also fairly long (meaningful) labels, which is convenient for debugging and development, but expensive in memory usage. Two libraries were used for manipulating this graph, and the memory-based implementation used here stores edges in standard Java library data structures, which add an additional level of memory overhead (e.g., Java stores strings in unicode, not ascii). As a consequence, the memory-based implementation could not load the entire MUC+AP graph in the available address space of our (32-bit) machine. While more memory-efficient implementation could certainly be produced - or alternatively, the experiments could be conducted on a machine with larger address space - we leave this task as a subject for future work, and for now simply extrapolate the performance of such an implementation from moderate-sized datasets.

While the MUC corpus is larger than the email corpora, and the query distribution includes more nodes, query processing time is only 0.4 seconds on average for the person name extraction task and 0.8 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 experimental

Corpus	nodes [K]	edges [K]	Graph walk	PCW:0	PCW:0.5	PCW:0.8
City name extraction						
MUC	82	244	0.8 (0.1)	1.8 (0.1)	1.7 (0.1)	0.5 (0.1)
MUC+1/4AP	326	1,077	3.3 (0.5)	10.3 (0.4)	6.7 (0.3)	5.3 (0.4)
MUC+1/2AP	564	1,910	7.8 (0.3)	20.0 (0.7)	13.0 (0.5)	10.3 (0.7)
MUC+3/4AP	785	2,682	11.3 (0.9)	-	-	-
Person name extraction						
MUC	82	244	0.4 (0.0)	0.6 (0.1)	0.5 (0.0)	0.2 (0.1)
MUC+1/4AP	326	1,077	0.6 (0.1)	2.5 (0.5)	1.8 (0.2)	1.7 (0.3)
MUC+1/2AP	564	1,910	0.9 (0.1)	4.1 (0.6)	2.2 (0.5)	2.4 (0.1)
MUC+3/4AP	785	2,682	2.0 (0.9)	-	-	-

Table 6.8: Average query processing time and standard deviation [secs] for the named entity coordinate extraction tasks, using graph walk of $k = 6$ steps and path constrained graph walk with varying thresholds.

datasets correspond to only few mentions in the corpus, such that a smaller number of edges is traversed.)

As shown in Table 6.8 and Figure 6.3, the average query processing times for the larger corpora get substantially longer, up to 11.3 seconds on average per query for the MUC+3/4AP corpus and the city name extraction task. In general, the implementation scheme applied in our experiments can be improved by using better machinery as well as by distributed computing. We therefore expect processing times to be shorter using optimized systems, as well as using algorithms that approximate the graph walk (see Section 6.4).

Rather than process the graph walk using the machine's memory, it is also possible to store the graphs in secondary memory. We used the open-source database package Sleepycat [Sleepycat] to store the user-defined nodes and edges. This allowed us to execute the graph walk for the large MUC+AP corpus. The cumulative number of nodes visited at each step of the graph walk for MUC, MUC+1/4AP, MUC+1/2AP and the MUC+AP corpora are presented in Figure 6.4 (logarithm scale). As shown in the figure, the overall number of nodes visited increases roughly by a factor of 2, given a double sized corpus. (The MUC corpus, which is characterized by somewhat different named entity distribution, can be considered as an outlier.) The figure also shows that the graph walks for the city name queries spread more in the graph, compared with person name queries.

Next we discuss the effect of the path constrained walks on scalability.

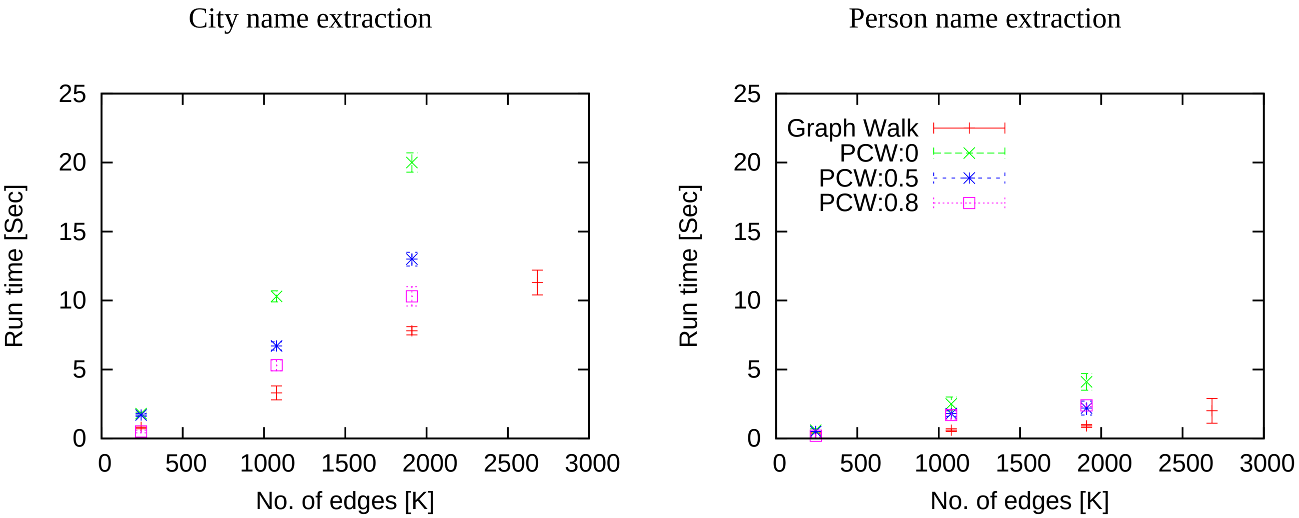


Figure 6.3: Average query processing time and standard deviation [secs] for the named entity coordinate extraction tasks, using graph walk of $k = 6$ steps and path constrained graph walk with varying thresholds. (A graphical display of Table 6.8.)

6.3.2 Impact of Path Constrained Walks on Scalability.

Table 6.8 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.[3] The average processing times required for the execution of unconstrained graph walk are given in the column named ‘Graph walk’. Figure 6.3 gives a respective graphical display of these processing times. The reported results indicate that longer processing times are required using the path constrained walks, compared with the unconstrained graph walks. As expected, the processing times shorten as the threshold applied increases.

In Figure 6.5, on the other hand, it is shown that the number of nodes visited at each step of the walk starts dropping at $k = 4$ using the path constrained walks. (In practice, graph nodes that do not have an outgoing edge associated with a high probability edge in the path tree are discarded.) 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,

[3] Results of the path constrained walk for the MUC+3/4AP are not reported due to high memory requirements, as discussed later in this section.

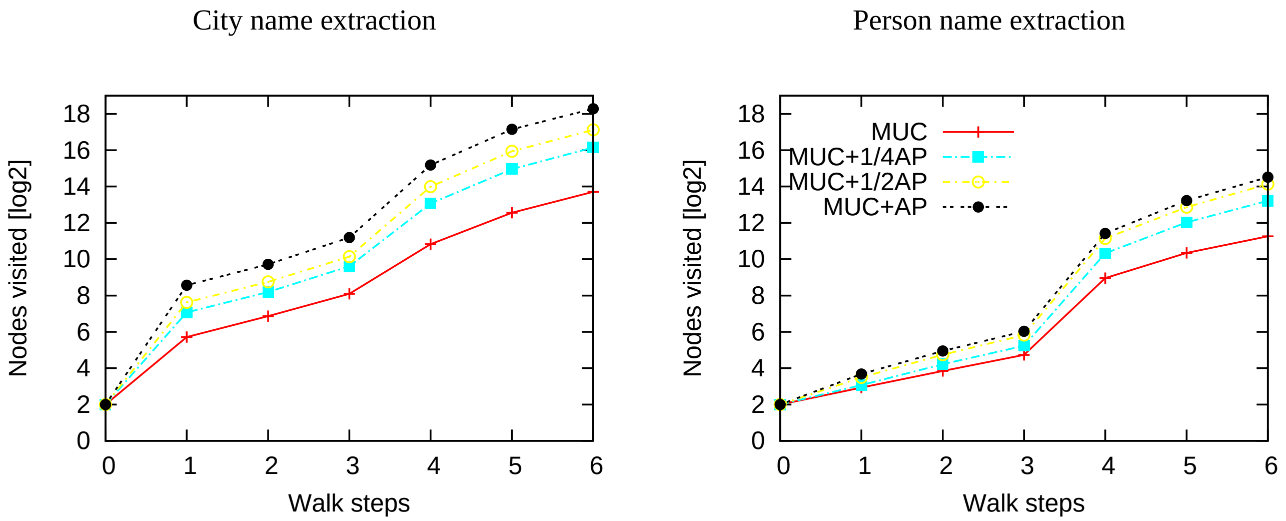


Figure 6.4: The cumulative number of nodes visited at each step of the graph walk, for the city name extraction and person name extraction datasets, for increasingly larger corpora.

the path tree represents the paths that correspond to the top nodes observed in the set of training examples, thus other arbitrarily possible paths are eliminated. Finally, applying a threshold on the probabilities associated with the path tree edges above which the graph walk is terminated, eliminates more (possibly frequent) paths.

While the path constrained walks limit the number of nodes (and edges) that are traversed in the walk, we note that the path constrained walks require the processing of all combinations of a node and its unique histories (represented as graph node and path tree node pairs, see Section 3.4). In the experiments reported in Table 6.8 and Figure 6.3, these added processing requirements overcome the savings due to node pruning in terms of running time. In addition, maintaining graph and path tree node pairs requires additional memory in comparison to the unconstrained walk. Therefore, we conclude that path constrained walks, while contributing to performance, involve in practice an additional computational cost. However, path constrained walks are expected to save on processing times in case that the graph is accessed from a secondary memory. In that case, node pruning can affect the expense involved in disk access.

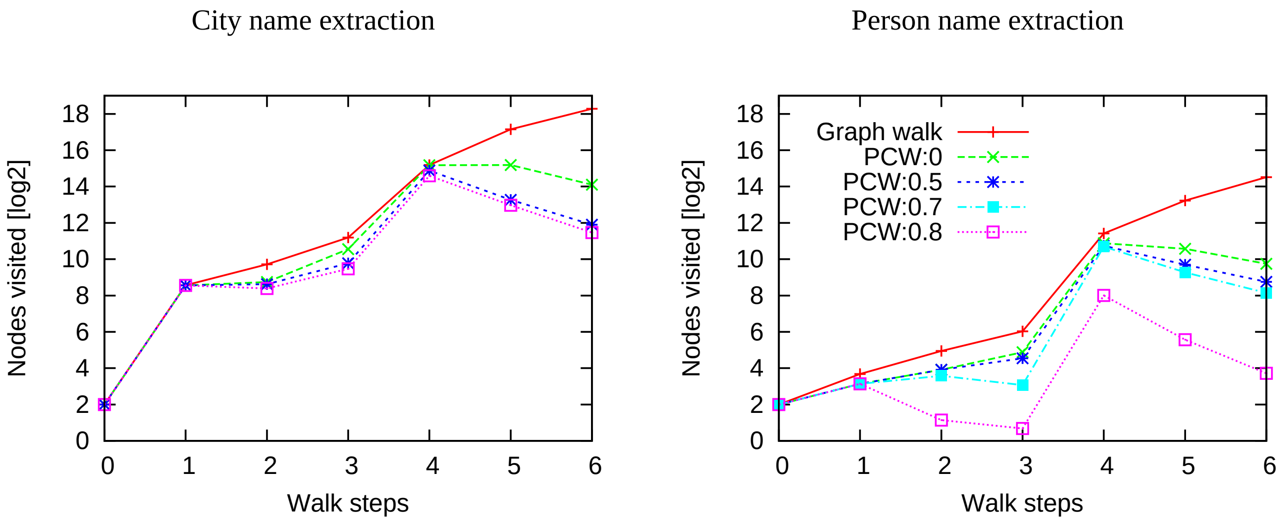


Figure 6.5: The cumulative number of nodes visited at each step of the graph walk using the MUC+AP corpus, for city name extraction and person name extraction, applying unconstrained graph walk and path constrained walk (PCW) with varying thresholds.

6.4 Related Work

As mentioned earlier, the paradigm of Personalized PageRank requires a power iteration computation given a query, which may correspond to impractical response times. Alternatively, a large set of 'personalized' distributions can be pre-processed, where computing and storing distribution vectors for all possible queries (node combinations) is infeasible. In order to alleviate the scalability problem, one must therefore either exploit special features of the web graph or relax the exact problem to an approximate one [133]. This section reviews the efforts made to address this scalability challenge. The related research reviewed includes techniques for pre-computing and storing a reduced number of Personalized PageRank vectors, sampling of the graph walk, and efficient matrix multiplication and inversion operations. Our main focus is on algorithms developed in the context of Personalized PageRank.

Scalable Storage of Personalized PageRank Vectors

Haveliwala [57] suggested the topic biased PageRank model, where his implementation 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. The final node scores were then computed using a linear combination of the topic-sensitive vectors, weighted by the similarity of the query to each topic.

Kamvar et-al [68] later suggested the *BlockRank* model. This model is adapted to the Web and restricts 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 [67] 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 recommended to restrict personalization to a set of hub nodes [67], since any node of the database may be included in a query. Instead, a Personalized PageRank vector is computed in the ObjectRank model for each word in the corpus vocabulary. A few monotone score-combining functions for multi-word queries are suggested by the authors. They also propose 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.

Sampling of the Graph Walk

Fogaras et-al [48] 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 [67], 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 $1-\gamma$ and ends with probability γ, see Equation 2.6). 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 [67]). The authors suggest also sampling 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 in the fully computed 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 [83].

Recently, Chakrabarti [23] 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, [48]) for a small fraction of nodes, chosen using query log statistics. According to 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 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 other nodes in the subgraph, using a dynamic programming computation [67]. 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 of the graph walk and pruning the resultant personalized vectors. In addition, computation is limited to a sub-graph, such that the graph walk is approximated locally. Experimental results have shown this approach to be preferable to ObjectRank [7] implementation in terms of pre-processing time, indexing space and online computation time.

Scalable Matrix Operations

The graph walk can be executed using implementations of matrix operations. The area of accelerating matrix multiplication is well studied, and there are various techniques available that reduce this opreration complexity and processing time. We review several relevant examples that pertain to the Personalized PageRank paradigm.

Kamvar et-al [69] suggested the technique of *quadratic extrapolation* that is applied periodically to enhance the convergence of PageRank using the simple power iteration method (i.e., iterative matrix multiplication). The authors claim that quadratic extrapolation eliminates the bottleneck for the power method, namely the second and third eigenvector components in every iteration, thus boosting the effectiveness of the simple power method.

Sun et-al [128] 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, they suggest to output a local estimation of the Personalized PageRank vector. Tong et-al [133] suggest an enhanced approach, which allows a global estimation of the Personalized PageRank vectors. They are interested in evaluating the stationary distribution of the Personalized PageRank graph walk process. The stationary distribution can be found by solving a linear system problem (this method is alternative to the power iteration approach), where a matrix inversion operation is required. Once the inverse matrix is computed and stored, the Personalized PageRank vectors for every given query can be efficiently computed in real-time. The matrix inversion and storing, however, requires quadratic space and cubic pre-computation. The authors alleviate this scalability problem by considering the block-wise structure property of the graph, as well as linear correlations that often exist across rows and columns of the adjacency matrix. Specifically, they suggest to partition the adjacency matrix, and pre-compute and store the inversion of each partition, rather than the full matrix. Given a low-rank approximation of the cross-partition links, a global evaluation of the Personalized PageRank distribution can be obtained. Given a query, only a few matrix-vector multiplication operations are required. Experimental results show that this approximation preserves high quality of the computed values, and achieves high speed ups in comparison with the iterative approach. The algorithm results in major savings in pre-computation and storage costs, compared with a straight-forward inversion of the full original matrix.

Finally, Cohen and Lewis [26] have 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 particularly 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.

6.4.1 Summary

In this chapter, we conducted comparative experiments for different framework parameters and design choices.

Based on empirical evidence, we find that finite (and relatively short) graph walks are preferable in some cases to infinite walks, i.e. to the stationary state probabilities. The graph walk length should be sufficient, however, to reach the relevant nodes in the graph. The edge weighting schema used also affected the graph walk performance in some cases.

The comparative evaluation of the various learning methods given in this chapter showed that global features are useful for some problems. In particular, path information was shown to be highly informative in the language domain, where local graph walks and weight tuning assigned high weights to proximate yet irrelevant nodes.

In terms of scalability, we have shown that the path constrained walks approach improves query processing times significantly, where most of the paths followed by the base (unconstrained) graph walks are effectively pruned during the walk. We have also shown that applying a threshold to the path constrained graph walk schema can improve both accuracy and scalability.

The processing times, given short walk length k and medium-sized corpora, were shown to be fast and appropriate for online settings. In addition, we have discussed related research concerned with improving the scalability of the Personalized PageRank paradigm for larger graphs. A majority of the algorithms discussed can be readily implemented within our framework.

Chapter 7

Conclusion

7.1 The Framework

This thesis presents a general framework for inducing adaptive similarity measures in heterogenous data represented as an entity-relation graph (Chapter 2). The framework builds on existing graph-walk based paradigms that generate measures of structural similarity between entities in the graph. In particular, the Personalized PageRank paradigm is used throughout this thesis; yet, other graph walk variants, e.g., lazy graph walks, can be readily applied. Previously, researchers have applied graph walks using carefully designed graphs with the goal of solving specific problems. In contrast, this thesis claims and shows that given a general representation of the data that is not engineered for a specific task, multiple tasks can be defined and performed as queries in this framework using the same underlying graph.

The graph walk paradigm has many desired properties in computing entity similarity in graphs and it is shown that finite graph walks give good performance in response to various queries in many cases. However, if labeled instances of entity relations in the graph are available, then learning can be applied to further adapt the similarity measure produced by the graph walk to the relation sought. Previously, it has been suggested to tune parametric edge weights in the graph, such the probability flow in the graph walk process is biased towards the nodes considered as correct answers to the query. In this thesis, two additional approaches were proposed: reranking, and path constrained walks (Chapter 3). Unlike weight tuning, both of these methods can consider global information about the graph walk. In reranking, discriminative learning can be applied to reorder the rankings generated by an initial graph walk using high level features; for example, these features can describe the set of paths traversed in reaching a target node from the query distribution. We proposed generic features that model global properties of the graph walk, including the sequences of edges traversed. In the path constrained walk approach, the edge weight parameters are conditioned on the history of the walk, and are updated dynamically based on edge sequence features. The learning methods of weight tuning, reranking and path constrained walks have different characteristics in terms of scope, applicability and impact (Sections 3.5 and 6.2.1). In some cases, combining local and global learning is advantageous (Section 6.2.2).

The scalability of graph walks in general, and Personalized PageRank in particular, has received much attention in recent years, with the goal of providing a fast response to a query (Section 6.4). Most of this research is orthogonal to this thesis and can be readily incorporated into the framework's implementation. Interestingly, the empirical results reported show that conducting short finite graph walks is both computationally efficient and also provides better accuracy in some cases (Section 6.1.1).

7.2 Case Studies

The thesis presents a case study in the domain of personal information management. It was shown that multiple tasks in this domain can be addressed uniformly as queries, including novel problems such as person name disambiguation, meeting attendees recommendation; as well known tasks such as threading and alias finding (Section 4.2). In most cases, learning led to improvements in performance. In particular, high level information about the graph walk was found to be useful, where reranking and the path constrained walks methods outperformed the weight tuning approach (Section 4.4).

A second domain studied in the thesis is the processing of a corpus of parsed sentences represented as a graph (Section 5.1). Applying graph walks to induce a measure of inter-word similarity gave mediocre results in this case. We found that in this domain, the assumption embedded in the graph walks, that proximate nodes are more relevant, is often false. In particular, in this domain nodes reached over specific edge sequences were more relevant than proximate nodes connected over unmeaningful relations. While graph walks are not ideal in such settings, the global learning methods, namely reranking and path constrained walks, gave excellent results (Section 5.4).

Comparing the framework of adaptive graph walks to state-of-the-art vector-space methods on the task of coordinate term extraction from parsed text, showed the framework to be preferable for small and medium text corpora (Section 5.4).

In general, tasks corresponding to “long” queries appeared to be biased in the experiments towards nodes that are highly connected in the graph in the experiments (Section 4.5). This phenomenon is known in the literature, and can be addressed to some extent by down weighting transitions in the graph towards such nodes (Section 2.4.1).

Overall, the case studies demonstrate that multiple tasks in a given domain can be successfully processed using this proposed framework. The tasks included in the thesis can be theoretically represented within related paradigms, such as statistical relational learning (Section 2.4.6). However, while statistical relational learning is more general, we find that the graph-walk based framework allows better scalability and is more suited for search settings, as of today (Appendix C).

7.3 Future Directions

There are many directions in which the framework presented can be extended; in what follows, we detail several possible venues of future research.

Framework. The graph walk representation only accommodates binary inter-entity relations. It is an open question if and how the proposed framework can account for n-ary relations. As an example, consider a graph representation of the sentences “Mary likes ice-cream in winter”, “Andy like ice-cream in winter”, “Jamie has ice-cream” and “Jamie likes tea in winter”. For a query that includes ”Mary”, the graph walk will find Jamie and Andy to be equally relevant, since the set of the corresponding connecting paths in both cases is identical. However, the combination of the arguments “likes”, “ice-cream” and “in winter” is more indicative of similarity to ”Mary” then their appearance in isolation. In the framework described in the thesis, it may be possible to model n-ary relations as features in reranking; i.e., in addition to the paths traversed, properties of the values of the nodes traversed can be considered. The features modeled will need to be general, or selected carefully, in order to avoid boosting the corresponding feature space. Another possible solution is to represent tuples (e.g., the combination of ”ice-cream” ”winter”) as nodes; this approach will involve an additional computational cost.

In Section 2.3.2, we argued that the graph representation is modular, where multiple information sources can be added to the graph. Presumably, adding nodes and edges provides more evidence for entity similarity in the graph, and thus is expected to have a positive effect on the similarity measures produced. On the other hand, it is possible that adding irrelevant or noisy information can degrade performance. An open question is in what circumstances and due to which factors adding information can *hurt* performance.

Learning. In terms of learning, we would like to modify the path constrained graph walk approach such that it can consider diverse types of features. The path constrained walks have been shown to improve performance due to the consideration of path information during the graph walk process. However, the features embedded in the path constrained walk method are limited to modeling information about the edge sequences traversed. It is therefore desired to incorporate richer types of high level features in the graph walk process (similarly to reranking); for example, it may be beneficial to consider various properties of the graph nodes traversed during the walk. A possible approach in this regard is to train a separate classifier per each of the vertices in the learned path tree. In addition, in order to account for features that are scarce, smoothing techniques that are adjusted to the graph settings may be useful.

Further, the learning settings that are assumed in this thesis can be relaxed and allow additional types of user feedback; that is, rather than consider binary signals about a node relevancy, relative node preferences or other forms of feedback may be provided. The learning procedures that are described in this thesis will need to be adjusted accordingly.

Another interesting venue for future work is learning to adapt the structure of the graph over time. Suppose that ongoing user feedback indicates that a subset of the edge types in the graph is consistently uninformative. These edges can be pruned from the graph, resulting potentially in savings in query processing times as well as reduced noise levels in the graph walk process. Similarly, one may be interested in “hard-coding” relationships between entities in the graph that are known to be closely related (for example, according to user feedback; or, based on relevant results from another domain). Adding links to the graph may improve the quality of response to future queries. A policy of modifying the underlying graph structure should be evaluated with scrutiny, where the graph should remain general in order to provide good performance given arbitrary queries.

A related question is if and how predictive models that are learned for one task can be used to leverage performance for other tasks in a given domain. Suppose that a sufficiently large set of labeled examples is available for one task, but only a few or no examples are available for another type of relation sought. It is an open question, to what extent the similarities in the graph are general and can be shared (or, *transferred*) across different tasks; for example, the edge weight parameters that are learned in one task may be helpful for other tasks. Correspondingly, relevant mechanisms for leveraging learning across tasks are required.

Applications. In terms of applications, while the evaluation of the framework focused on providing a single ranked list in response to a query, the same querying mechanism can be used in a *bootstrapping* process. In bootstrapping, the results retrieved in response to queries are used for automatically creating new queries, with the goal of machine-driven construction of a knowledge base. In order to avoid divergence in bootstrapping due to noisy responses, it may be desired to control the ratio between precision and recall of the responses to a query. The path constrained graph walk method proposed provides a coarse solution to this issue using its threshold mechanism. In addition, evaluating and controlling the reliability, or confidence, of the predicted similarity measure can be useful.

Appendix A

Symbols and Definitions

For the reader's convenience, following are the symbols used in this thesis and their definitions.[1] Table A.1 includes symbols related to the graph walk framework and Table A.2 lists the symbols that are related to the learning settings and algorithms.

Symbol	Definition
G	a graph
x, y, z	graph nodes
N	the number of graph nodes in total
$\tau(x)$	the type of entity represented by node x
ℓ	edge (relation) type
L	the set of all edge types
θ_ℓ	parametric weight of edge ℓ
Θ	the set of graph edge weight parameters
L_{xy}	the set of edge types from node x to y
S_x	the set of outgoing edge types from node x
$\mathbf{M}$	the transition matrix
$Pr(x \longrightarrow y)$	the probability of reaching node y from node x over a single time step
γ	reset probability
k	number of graph walk steps
τ_{out}	the type of nodes retrieved, as specified in a query
V_q	query distribution, as specified in a query
V_k	score distribution over graph nodes after a graph walk of k steps is performed.

Table A.1: Symbols related to the graph walk framework and their definitions.

[1] These definitions hold throughout the thesis unless stated explicitly otherwise.

Symbol	Definition
e_i	example query i
R_i	a set of relevant answers corresponding to query e_i
l_i	the output ranked list generated per example e_i
z_{ij}	the node located at rank j in list l_i
p_z	the score assigned to node z in a ranked list
p_z^{opt}	an optimal score assumed for a target node z in a ranked list
err_z	the error for a target node z in a ranked list
$U_z(t+1)$	the set of graph nodes that are traversed at step $t+1$, in route to target node z at step k
η	learning rate of gradient descent
f_k	reranking feature k
$f_k(z)$	the value of feature f_k per node z
α_k	a real-value weight associated with feature k in a reranking function
$\bar{\alpha}$	the set of feature weights in a reranking function
$F(z,\bar{\alpha})$	the value of the ranking function for node z
$Q(x \xrightarrow{=d} z)$	the probability of stopping at z in graph walk originating from X of length d
$F_d(z)$	a partial ranking function for node z, computed up to graph walk step d
T	path-tree
t_i	a node in path tree T
p	edge sequence
C_p^+	the count of p within the paths leading to correct nodes considered by T
C_p^-	the count of p within the paths leading to nodes assumed to be incorrect, considered by T
$Pr(p)$	an estimate of the probability of reaching a correct node following p

Table A.2: Symbols related to learning and their definitions.

Appendix B

Evaluation Metrics

The graph walk search framework, as well as the baseline methods that we compare it to, all generate a ranked list of entities. As in traditional document retrieval settings, every query is mapped to a set of relevant "correct" answers. In this thesis we use the following evaluation metrics:

Mean Average Precision (MAP). Consider a ranked list that has n correct entries at ranks $k_1,\ldots,k_n$, and assume that the end user will scan down the list of answers and stop at some particular "target answer" k_i that he or she finds to be of interest. One would like the density of correct answers up to rank k_i to be high: to formalize this, define the *precision at rank k*, $prec(k)$, to be the number of correct entries up to rank k, divided by k—i.e., the precision of the list up to rank k[1]. The *non-interpolated average precision* of the ranking is simply the average of $prec(k)$ for each position k_i that holds a correct entry:

$$AveragePrecision = \frac{1}{n}\sum_{i=1}^{n} prec(k_i)$$

As an example, consider a ranked list of items, where the items at ranks 1,2,5 are correct and those at ranks 3,4 are not. The precision at ranks 1 and 2 equals 1.0, and the precision at the next correct item is 0.6 (since there are 3 correct answers before rank 5). The non-interpolated average precision on this ranked list is thus $(1+1+0.6)/3 = 0.87$. The Mean Average Precision (MAP) is the average of the non-interpolated average precision scores, over multiple rankings (queries).

In our ranking systems, it may happen that some correct answers do not appear in the ranked list (i.e., they are assigned zero probability). In this case, we follow standard

[1] In case that the ranking results include blocks of items with the same score, a node's rank is counted as the average rank of the "block".

practice and define $prec(k_i)$ for that answer to be zero. (Continuing our example, if there were a fourth answer that did not appear anywhere on the list, then the average precision would be $(1+1+0.6+0)/4 = 0.65.)$ If there are no correct answers for a problem, we define the average precision of any ranking to be 1.0.

Elsewhere, it has been noted that $prec(k_i)$ can be viewed as a ratio $m_{i,actual}/m_{i,opt}$, where $m_{i,opt}$ is the number of entries the user must examine to find the *i*-th correct entry in an optimal ranked list, and $m_{i,actual}$ is the analogous number for the actual ranked list [143]. Thus, non-interpolated average precision can also be interpreted as a measure of the additional work imposed on the user by a suboptimal ranking—e.g., an average precision of 0.5 means that the user must examine twice as many list entries as needed, on average.

Mean Reciprocal Rank (MRR). Reciprocal rank is the reciprocal of the rank of the first relevant answer for each query. Thus, the MRR for the example given above is 1. The mean reciprocal rank is the average of the reciprocal ranks for all queries in the evaluation set. Note that if all queries have a single correct answer, MRR and MAP are equal.

Accuracy. This measure denotes the percentage of queries for which the top item in the ranked list contains a relevant answer.

11-point interpolated precision-recall curve. The interpolated precision at a certain recall level r is defined as the highest precision found for any recall level $r' \geq r$:

$$p_{inter}(r) = \max_{r' \geq r} p(r')$$

The 11-point curve shows the interpolated precision measured at 11 recall levels (0,0.1, ...,1.0), where it is averaged over the set of evaluated queries. While the MAP measure is a single score (which is biased towards performance at the top ranks of the retrieved lists), a precision-recall curve gives a detailed and graphical view of performance.

Appendix C

Markov Logic Networks: Empirical Comparison

An overview of the Markov Logic Networks (MLNs) paradigm is given in Section 2.4.6. In this appendix, we describe the results of applying Markov logic to the task of email threading.

We use the open-source Alchemy system [78] to conduct the experiments. Table C.1 includes an MLN model designed for the threading problem. The first part of the model details the predicates that are used in the email domain, corresponding to the graph schema described in Table 4.1 (where the relations involving meetings are excluded). The instantiated predicates encode the structure of the graph, and are detailed as evidence predicates in a separate data file. For example, an edge of type *sent-to* from *message* m_2 to *person* p_1 is represented by the evidence predicate: sent-to (m_2,p_1). Predicates that are not explicitly instantiated as evidence will be assumed to be false.

Our model of the threading problem involves to a single class predicate rule: thread(m_1,m_2). In the description of the training data, the labeled thread predicates are provided. Given the test portion of the data, the likelihood of the thread predicated being true will be evaluated.

The last part of the model shown in Table C.1 includes the rules that connect the evidence (attribute) predicates with the class predicate. For example, consider the rule $\forall x \forall y$ *has-term*(x,z)$\wedge$ *has-term*$(y,z) \Rightarrow$ *thread*(x,y).) This rule is denoted in the Markov logic model as: ‘hasTerm (m_1, t_1) $\wedge$ hasTerm (m_2, t_1) $\Rightarrow$ thread (m_1, m_2). The rules applied have been designed manually, based on the models learned by the learning methods in our framework. Automatic learning of a model given examples is supported in Alchemy; however, we have not evaluated this approach.

Predicates:

has-subject-term (msg, term)
has-term (msg, term)
sent-to-email (msg, email-address)
sent-from-email (msg, email-address)
sent-to (msg, person)
sent-from (msg, person)
alias (person, email-address)
as-term (person, term)
sent-on-date (msg, date)
thread (msg, msg)

A single predicate rule:

! thread (m1, m2)

Rules connecting attribute match predicates to class match predicates:

hasTerm (m_1, t_1)	$\wedge$ hasTerm (m_2, t_1)	$\Rightarrow$ thread (m_1, m_2)
hasSubjectTerm (m_1, t_1)	$\wedge$ hasSubjectTerm (m_2, t_1)	$\Rightarrow$ thread (m_1, m_2)
sentFromEmail (m_1, e_1)	$\wedge$ sentToEmail (m_2, e_1)	$\Rightarrow$ thread (m_1, m_2)
sentFromEmail (m_1, e_1)	$\wedge$ sentFromEmail (m_2, e_1)	$\Rightarrow$ thread (m_1, m_2)
sentToEmail (m_1, e_1)	$\wedge$ sentFromEmail (m_2, e_1)	$\Rightarrow$ thread (m_1, m_2)
sentToEmail (m_1, e_1)	$\wedge$ sentToEmail (m_2,e_1)	$\Rightarrow$ thread (m_1, m_2)
sentFrom (m_1, p_1)	$\wedge$ sentTo (m_2, p_1)	$\Rightarrow$ thread (m_1, m_2)
sentTo (m_1, p_1)	$\wedge$ sentTo (m_2, p_1)	$\Rightarrow$ thread (m_1, m_2)
sentOnDate (m_1, d_1)	$\wedge$ sentOnDate (m_2, d_1)	$\Rightarrow$ thread (m_1, m_2)

Table C.1: A Markov Logic Network suggested that models the message threading problem.

Experiments were conducted using the management game email corpus (where reply lines have been discarded). First, rule weights were learned. Then, in the inference step, real-value scores were assigned to all of the class predicate pairs, designating the confidence of the model in each predicate being true. A a ranked list was constructed based on these scores for every query in the test set.

The yielded test set result for this corpus was 0.69 in mean average precision. In comparison, graph walks using uniform weights gave MAP of 0.53; graph walks using the learned set of edge weights resulted in MAP of 0.59; and, both reranking and the path constrained walks resulted in MAP of 0.73 (as detailed in Table 4.7). Since information about relevant rules was manually predefined in the MLN schema, and learning was applied to tune the weights of these rules, the results produced using MLNs should be compared against the latter learning approaches. Overall, performance is comparable between the two paradigms for this task and corpus.

Learning the rules weights required about 5 minutes in Alchemy, using a commodity PC. We applied the lazy SAT algorithm for inference. Inference over all message pairs in the corpus (corresponding to a total of 817^2 predicates, as there are 817 distinct messages in the management game corpus) required about 5 hours and 30 minutes overall. This corresponds to about 0.03 seconds per individual query, with the graph walk process. The query response times for the threading queries using the management game corpus using our framework were similar (see Table 6.7).

In another set of experiments, we were not able to apply learning in Alchemy to the threading model using the larger Enron corpora (see Table **??**) because of memory constraints in the phase of network grounding.

Based on this limited set of experiments, we find that the graph-based framework demonstrates higher scalability on the evaluated corpora.

As discussed earlier, an advantage of the graph-walk framework is that while learning improves results for pre-defined tasks, the framework can generate results also for ad-hoc queries, applying graph walks with no learning. The Markov logic networks paradigm on the other hand requires rule learning (or expert knowledge) as a pre-requisite, since different network structures are defined per task (i.e., for every model); in the graph-walk framework, a fixed graph is used for multiple tasks.

Overall, we believe that the proposed framework is more appropriate in search settings, where arbitrary queries are possible. Our experiments suggest that the graph walk framework is more scalable as it can handle larger corpora.

Bibliography

[1] Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25(3), 2003. 2.4.1

[2] Manu Aery and Sharma Chakravarthy. emailsift: Email classification based on structure and content. In *ICDM*, 2005. 4.6

[3] Alekh Agarwal and Soumen Chakrabarti. Learning random walks to rank nodes in graphs. In *ICML*, 2007. 3.6.1

[4] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. Learning to rank networked entities. In *KDD*, 2006. 1.2, 3, 3.1, 3.6.1, 3.6.2, 6.2

[5] Shivani Agarwal. Ranking on graph data. In *ICML*, 2006. 3.6.1

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

[7] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, 2004. 2.2.3, 2.2.3, 2.4.3, 3.6.2, 6.4, 6.4

[8] Krisztian Balog, Leif Azzopardi, and Maarten de Rijke. Formal models for expert finding in enterprise corpora. In *SIGIR*, 2006. 4.2, 4.6

[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. 4.2, 4.6

[10] Ron Bekkerman, Ran El-Yaniv, and Andrew McCallum. Multi-way distributional clustering via pairwise interactions. In *ICML*, 2005. 4.6

[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. 4.6

[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. 4.1, 4.6

[13] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, 2002. 2.2.4, 2.4.3

[14] Matthew W. Bilotti, Paul Ogilvie, Jamie Callan, and Eric Nyberg. Structured retrieval for question answering. In *SIGIR*, 2007. 5.3

[15] R. Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *IJCAI*, 2005. 2.4.6

[16] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30, 1998. 2.2.1

[17] Razvan C. Bunescu and Raymond J. Mooney. A shortest path dependency kernel for relation extraction. In *HLT-EMNLP*, 2005. 3.6.3, 5.5

[18] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullende. Learning to rank using gradient descent. In *ICML*, 2005. 3.1

[19] Lou Burnard. *Users Guide for the British National Corpus*. British National Corpus Consortium, Oxford University Computing Service, Oxford, UK, 1995. 5.3

[20] Vitor R. Carvalho and William W. Cohen. On the collective classification of email ”speech acts”. In *SIGIR*, 2005. 4.4.2, 4.6

[21] Vitor R. Carvalho and William W. Cohen. Preventing information leaks in email. In *SDM*, 2007. 4.6

[22] Vitor R. Carvalho and William W. Cohen. Ranking users for intelligent message addressing. In *ECIR*, 2008. 4.2, 4.6

[23] Soumen Chakrabarti. Dynamic personalized pagerank in entityrelation graphs. In *WWW*, 2007. 6.4

[24] Huan Chang and David Cohn. Learning to create customized authority lists. In *ICML*, 2000. 3.6.1

[25] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL*, 2005. 3.3, 3.6.3

[26] Edith Cohen and David D. Lewis. Approximating matrix multiplication for pattern recognition tasks. *Journal of Algorithms*, 30(2), 1999. 6.4

[27] 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

[28] William W. Cohen and Einat Minkov. A graph-search framework for associating gene identifiers with documents. *BMC Bioinformatics*, 7(440), 2006. 2.2.1, 2.2.1, 2.2.3, 2.2.3, 3.3.1, 3.3.3

[29] William W. Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWEB*, 2003. 4.4.1, 4.4.3

[30] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research (JAIR)*, 10:243–270, 1999. 3.3

[31] Michael Collins. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *ACL*, 2002. 3.3, 3.6.3

[32] Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69, 2005. 3.3, 3.3.1, 3.5, 3.6.3

[33] Kevyn Collins-Thompson and Jamie Callan. Query expansion using random walk models. In *CIKM*, 2005. 1, 2.3.2, 2.4.3, 5, 5.5

[34] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein. *Introduction to Algorithms*. IT Press and McGraw-Hill, 1990. 2.4.1

[35] Fabio Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6), 1997. 2.4.5

[36] Aron Culotta and Jeffery Sorensen. Dependency tree kernels for relation extraction. In *ACL*, 2004. 3.6.3, 5.5

[37] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *LREC*, 2006. 5.3

[38] Christopher P. Diehl, Lise Getoor, and Galileo Namata. Name reference resolution in organizational email archives. In *SIAM*, 2006. 4.6

[39] Michelangelo Diligenti, Marco Gori, and Marco Maggini. Learning web page scores by error back-propagation. In *IJCAI*, 2005. 1.2, 3, 3.2, 3.2, 3.2, 3.5, 6.2

[40] Pedro Domingos, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. Unifying logical and statistical ai. In *AAAI*, 2006. 2.4.6, 2.4.6

[41] Peter G. Doyle and J. Laurie Snell. *Random Walks and Electrical Networks*. Mathematical Association of America, 1984. 2.4.2

[42] M. Dredze, T. Lau, and N. Kushmerick. Automatically classifying emails into activities. In *IUI*, 2006. 4.6

[43] Tamer Elsayed, Douglas W. Oard, , and Galileo Namata. Resolving personal names in email using context expansion. In *HLT-ACL*, 2008. 4.6

[44] Güneş Erkan and Dragomir R. Radev. Lexpagerank: Prestige in multi-document text summarization. In *EMNLP*, 2004. 5.5

[45] S. E. Fahlman. *NETL: A System for Representing and Using Real-World Knowledge*. MIT Press, Cambridge, MA, 1979. 2.4.5

[46] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, 2004. 2.4.1, 2.4.2, 2.4.3

[47] Christiane Fellbaum. *WordNet: An electronic lexical database*. MIT Press, 1998. 2.3.1, 2.3.2, 5

[48] D. Fogaras, B. Rácz, K. Csalogány, , and T. Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3), 2005. 2.2.1, 2.2.1, 6.3, 6.4

[49] Francois Fouss and Jean-Michel Renders. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3), 2007. 2.3.1

[50] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3), 1999. 3.3.1

[51] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, 1999. 2.4.6

[52] L. Getoor and B. Taskar. *Statistical relational learning*. MIT Press, Cambridge MA, 2007. 2.4.6

[53] W. Geyer, J. Vogel, L. Cheng, and M. Muller. Supporting activity-centric collaboration through peer-to-peer shared objects. In *ACM GROUP*, 2003. 4.6

[54] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity search in databases. In *VLDB*, 1998. 2.4.3

[55] Gregory Grefenstette. *Explorations in Automatic Thesaurus Discovery*. Kluwer Academic Publishers, Dordrecht, 1994. 5.4

[56] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD*, 2003. 1, 2.4.3, 3.6.2

[57] Taher H. Haveliwala. Topic-sensitive PageRank. In *WWW*, 2002. 1, 2.2.1, 6.4

[58] Jingrui He, Mingjing Li, Hong-Jiang Zhang, Hanghang Tong, and Changshui Zhang. Manifold-ranking based image retrieval. In *MM*, 2004. 2.3.2

[59] Marti Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, 1992. 5.2

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

[61] S. Henderson. Genre, task, topic and time: facets of personal digital document management. In *CHI*, 2005. 4.2, 4.6

[62] Ralph Holzer, Bradely Malin, and Latanya Sweeney. Email alias detection using social network analysis. In *LinkKDD*, 2005. 4, 4.6

[63] 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. 4, 4.6

[64] Liang Huang. Forest reranking: Discriminative parsing with non-local features. In *ACL*, 2008. 3.6.3

[65] Thad Hughes and Daniel Ramage. Lexical semantic relatedness with random graph walks. In *EMNLP*, 2007. 2.3.2, 5, 5.5

[66] Glen Jeh and Jennifer Widom. Simrank: A measure of structural-context similarity. In *SIGKDD*, 2002. 2.4.2

[67] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW*, 2003. 2.2.1, 6.4, 6.4

[68] 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. 6.4

[69] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating pagerank computations. In *www*, 2003. 6.4

[70] Hillol Kargupta, Anupam Joshi, Krishnamoorthy Sivakumar, and Yelena Yesha. *Data Mining: Next Generation Challenges and Future Directions*. MIT/AAAI Press, 2004. 4

[71] N. Katoh, T. Ibaraki, , and H. Mine. An efficient algorithm for k shortest simple paths. *Networks*, 12, 1982. 2.4.2

[72] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1), 1953. 2.4.1

[73] Edward Keenan and Bernard Comrie. Noun phrase accessibility and universal grammar. *Linguistic Inquiry*, 8, 1977. 5.4

[74] K. Kersting and L. De Raedt. Towards combining inductive logic programming with bayesian networks. In *ILP*, 2001. 2.4.6

[75] John Kleinberg. Authoritative sources in a hyperlinked environment. In *SODA*, 1998. 1, 3.6.1

[76] Brown Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *ECML*, 2004. 4.2, 4.3

[77] S. Kok and P. Domingos. Learning the structure of markov logic networks. In *ICML*, 2005. 2.4.6

[78] S. Kok, P. Singla, M. Richardson, and P. Domingos. The alchemy system for statistical relational ai. In *Department of Computer Science and Engineering, University of Washington, Technical Report. http://www.cs.washington.edu/ai/alchemy*, 2005. 2.4.6, C

[79] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *ICML*, 2002. 2.4.4

[80] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD*, 2006. 2.2.4, 2.4.1, 2.4.2

[81] N. Kushmerick and T. Lau. Automated email activity management: an unsupervised learning approach. In *IUI*, 2005. 4.6

[82] E.L. Lehmann. *Testing statistical hypotheses*. Wiley, 1959. 4.4

[83] 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. 6.4

[84] David E. Lewis and Kimberly A. Knowles. Threading electronic mail: A preliminary study. *Information Processing and Management*, 1997. 4.2, 4.6

[85] Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *CIKM*, 2003. 2.4.1, 2.4.1, 2.4.2, 4.6

[86] Dekang Lin. Automatic retrieval and clustering of similar words. In *COLING-ACL*, 1998. 5.4

[87] D. Lowd and P. Domingos. Efficient weight learning for markov logic networks. In *PKDD*, 2007. 2.4.6

[88] 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. 4.6

[89] Andrew McCallum, Andres Corrada-Emmanuel, and Xuerui Wang. Topic and role discovery in social networks. In *IJCAI*, 2005. 4, 4.6

[90] J.M McInerney, K. G. Haines, S. Biafore, and R. Hecht-Nielsen. Back propagation error surfaces can have local minima. In *International Joint Conference on Neural Networks (IJCNN)*, 1989. 3.2

[91] Rada Mihalcea. Unsupervised large-vocabulary word sense disambiguation with graph-based algorithms for sequence data labeling. In *HLT/EMNLP*, 2005. 5.5

[92] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. In *HLT/EMNLP*, 2004. 5.5

[93] L. Mihalkova and R. J. Mooney. Bottom-up learning of markov logic network structure. In *ICML*, 2007. 2.4.6

[94] Einat Minkov and William W. Cohen. An email and meeting assistant using graph walks. In *CEAS*, 2006. 4.4.3, 4.4.4

[95] Einat Minkov, William W. Cohen, and Andrew Y. Ng. Contextual search and name disambiguation in email using graphs. In *SIGIR*, 2006. 1, 1.1, 2.2.3, 2.2.3, 2.2.3

[96] Einat Minkov, Richard Wang, and William Cohen. Extracting personal names from emails: Applying named entity recognition to informal text. In *HLT-EMNLP*, 2005. 4.3, 4.4.1

[97] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7), 1994. 4.6

[98] Tom Mitchell, Sophie Wang, Yifen Huang, and Adam Cheyer. Extracting knowledge about users activities from raw workstation contents. In *AAAI*, 2006. 4.5.1, 4.6

[MUC6] MUC6. Proceedings of the sixth message understanding conference (muc-6). In *Morgan Kaufmann Publishers, Inc. Columbia, Maryland.*, 1995. 5.3

[99] J. Neville and D. Jensen. Dependency networks for relational data. In *ICDM*, 2004. 2.4.6

[100] Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, and Wei-Ying Ma. Object-level ranking: Bringing order to web objects. In *WWW*, 2005. 1.2, 3.6.2, 6.2

[101] Sebastian Padó and Mirella Lapata. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2), 2007. 1.3, 5, 5.4, 5.4.2

[102] 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, 2.2.1, 2.2.1, 2.3.1, 6.1.2, 6.3

[103] Chris Pal and Andrew McCallum. Cc prediction with graphical models. In *CEAS*, 2006. 4.6

[104] C. R. Palmer and C. Faloutsos. Electricity based external similarity of categorical attributes. In *PAKDD*, 2003. 2.2.4, 2.4.2, 4.5.3

[105] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, 2004. 2.3.2, 2.4.3, 2

[106] D. Petkova and W. B. Croft. Hierarchical language models for expert finding in enterprise corpora. In *ICTAI*, 2006. 4.2, 4.6

[107] H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI*, 2006. 2.4.6

[108] Vasin Punyakanok, Dan Roth, Wen tau Yih, and Dav Zimak. Semantic role labeling via integer linear programming inference. In *COLING*, 2004. 3.6.3

[109] M. R. Quillian. *Semantic memory. In M. Minsky (Ed.), Semantic information processing*. MIT Press, Cambridge, MA, 1968. 2.4.5

[110] C. Ramakrishnan, W. Milnor, M. Perry, and A. Sheth. Discovering informative connection subgraphs in multi-relational graphs. *SIGKDD Explorations Special Issue on Link Mining*, 2005. 2.4.3

[111] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62 (1-2), 2006. 2.4.6, 2.4.6, 2.4.6

[112] Matthew Richardson and Pedro Domingos. The intelligent surfer: Probabilistic combination of link and content information in PageRank. In *NIPS*, 2002. 2.2.1

[113] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996. 3.2

[114] D. Rumelhart, J. McClelland, and PDP Research Group. *Parallel Distributed Processing: exploration in the microstructure of cognition*. MIT Press, Cambridge, MA, 1986. 2.4.5

[115] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983. 2.4.1

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

[117] Jacques Savoy. Bayesian inference networks and spreading activation in hypertext systems. *Information Processing and Management*, 28(3), 1992. 2.3.1

[118] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999. 3.3.1

[119] R. Segal and J. Kephart. Incremental learning in swiftfile. In *ICML*, 2000. 4.2, 4.6

[120] Sandip Sen. Developing an automated distributed meeting scheduler. *IEEE Expert*, 12(4), 1997. 4.6

[121] Libin Shen and Aravind K. Joshi. An svm based voting algorithm with application to parse reranking. In *CONLL*, 2003. 3.3.1

[122] Libin Shen and Aravind K. Joshi. Ranking and reranking with perceptron. *Machine Learning*, 60(1-3), 2005. 3.3.1, 3.3.1

[123] Libin Shen, Anoop Sarkar, , and Franz Josef Och. Discriminative reranking for machine translation. In *HLT-NAACL*, 2005. 3.6.3

[124] P. Singla and P. Domingos. Discriminative training of markov logic networks. In *AAAI*, 2005. 2.4.6

[125] P. Singla and P. Domingos. Memory-efficient inference in relational domains. In *AAAI*, 2006. 2.4.6

[Sleepycat] Sleepycat. Sleepycat software. http://www.sleepycat.com. 6.3.1

[126] 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. 2.3.1

[127] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *NIPS*, 2005. 3.6.3, 5, 5.1, 5.5

[128] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, 2005. 6.4

[129] Martin Szummer and Tommi Jaakkola. Clustering and efficient use of unlabeled examples. In *NIPS*, 2001. 2.3.2, 2.4.4

[130] Egidio Terra and C. L. A. Clarke. Frequency estimates for statistical word similarity measures. In *NAACL*, 2003. 5.4.1

[131] Naftali Tishby and Noam Slonim. Data clustering by markovian relaxation and the information bottleneck method. In *NIPS*, 2000. 2.3.2, 2.4.4, 4.5.3

[132] Hanghang Tong and Christos Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. In *KDD*, 2006. 2.4.2, 4.5.3, 5.4.2

[133] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *ICDM*, 2006. 6.4, 6.4

[134] Hanghang Tong, Yehuda Koren, and Christos Faloutsos. Fast direction-aware proximity for graph mining. In *KDD*, 2007. 2.4.2

[135] Kristina Toutanova, Aria Haghighi, and Christopher D. Manning. Joint learning improves semantic role labeling. In *ACL*, 2005. 3.6.3

[136] Kristina Toutanova, Christopher D. Manning, and Andrew Y. Ng. Learning random walk models for inducing word dependency distributions. In *ICML*, 2004. 1, 2.2.1, 2.3.2, 2.4.3, 3, 3.6.2, 5, 5.5

[137] Ah Chung Tsoi, Gianni Morini, , Franco Scarselli, Markus Hagenbuchner, and Marco Maggini. Adaptive ranking of web pages. In *WWW*, 2003. 3.1, 3.6.1

[138] Raymond J. Mooney Tuyen N. Huynh. Discriminative structure and parameter learning for markov logic networks. In *ICML*, 2008. 2.4.6

[139] Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. What is the jeopardy model? a quasi-synchronous grammar for qa. In *EMNLP-CONLL*, 2007. 5.1

[140] 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

[141] 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

[142] Jen-Yuan Yeh and Aaron Harnly. Email thread reassembly using similarity matching. In *CEAS*, 2006. 4.6

[143] ChengXiang Zhai, William W. Cohen, , and John Lafferty. Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval. In *NIPS*, 2001. B

[144] Dengyong Zhou, Bernhard Scholkopf, and Thomas Hofmann. Semi-supervised learning on directed graphs. In *NIPS*, 2005. 2.4.4

[145] Dengyong Zhou, Jason Weston, Arthur Gretton, Olivier Bousquet, and Bernhard Schölkopf. Ranking on data manifolds. In *NIPS*, 2004. 2.4.4

[146] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, 2003. 2.4.4