Improving Graph-Walk Based Similarity with Reranking: Case Studies for Personal Information Management

EINAT MINKOV
University of Haifa
and
WILLIAM W. COHEN
Carnegie Mellon University

Relational or semi-structured data is naturally represented by a graph, 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 links. We represent personal information as a graph that includes *messages*, *terms*, *persons*, *dates* and other object types, and relations like *sent-to* and *has-term*. Given the graph, we apply finite random graph walks to induce a measure of entity similarity, which can be viewed as a *tool for performing search* in the graph. Experiments conducted using personal email collections derived from the Enron corpus and other corpora show how the different tasks of *alias finding*, *threading* and *person name disambiguation* can be all addressed as search queries in this framework, where the graph-walk based similarity metric is preferable to alternative approaches, and further improvements are achieved with learning. While researchers have suggested to tune edge weight parameters to optimize the graph walk performance per task, we apply reranking to improve the graph walk results, using features that describe high-level information such as the paths traversed in the walk. High performance, together with practical run times, suggest that the described framework is a useful search system in the PIM domain, as well as in other semi-structured domains.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**]: retrieval models, search process; I.2.6 [**Learning**]: Connectionism and neural nets

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Graph walk, learning, semi-structured data, PIM

1. INTRODUCTION

The canonical problem of *ad hoc* information retrieval (IR) is often formulated as the task of finding documents “similar to” a text query. Traditionally, this task is performed by clever application of textual similarity metrics. In modern settings,

Einat Minkov's address: Dep. of Management Information Systems, Faculty of Social Sciences, University of Haifa, Haifa, Israel, 31905.
William Cohen's address: Machine Learning department, Carnegie Mellon University, Pittsburgh PA, USA 15213.

however, documents are usually not isolated objects: instead, they are frequently connected to other objects, via hyperlinks, meta-data or relational structure. Consider a collection of email messages: while email messages are textual objects, the email headers include structural data relating to the social network involved, temporal information, etc. Other natural examples are XML documents [Guo et al. 2003] and the Semantic Web [Anyanwu et al. 2005].

Structured and semi-structured data can be naturally represented as a directed labeled graph, where nodes denote entities and typed edges represent the relations between them. While text documents may be represented by a bipartite graph that includes documents and term objects, structured data corresponds to richer, heterogeneous graphs that include multiple types of nodes and relations. For example, in the personal information management (PIM) domain, the graph nodes may represent *messages* and *terms*, as well as *persons*, *email addresses*, *dates* and other entity types. The graph edges in this domain denote inter-entity relations like *has-term*, between a *message* and the *terms* it contains; *sent-to*, between a *message* and its recipient *person* entities, and so on.

Figure 1 shows a small fraction of the email corpus of an Enron employee, represented as a graph.[1] The graph encodes the information that message "Msg.259" was sent on *date* "April 4 '01" from *email address* ginger.dernehl@enron.com to the *email addresses* of fiona.stuart@enron.com, michael.brown@enron.com and john.sherriff@enron.com; that this message includes the terms "memo", "organize" and "announce" in its subject line, and additional terms, including "review", in its content. This structured information was obtained directly from the corpus by parsing the message's header fields, as well as its textual body. In the graph, it is easy to observe that *messages* "Msg.259" and "Msg.260" have been sent on the same day, that they share the *email address* john.sherriff@enron.com as a recipient, and that they both include the *term* "memo" in their subject line. In addition, the graph includes *person* nodes, which are linked both to the relevant *email address* nodes with an *alias* relation, and to *terms* that match the person name with a relation of *as-term*.[2]

Similarly, one may add *meeting* objects to the graph. Assuming that a meeting entry typically includes a textual description and a list of attendees, it can be linked in the graph to the corresponding *terms*, and *person* or *email address* nodes.

While traditional IR considers the general task of finding documents relevant to an information need (defined as a set of terms), there are a variety of other specialized tasks possible in structured domains like PIM. Table I includes several PIM tasks that are addressed in this paper. For instance, one may be interested in finding email aliases, *i.e.*, the set of *email addresses* used by a specific person; in this case, the desired output is *email addresses*, and the person's name may be input as *terms*. Alternatively, one may wish to find *email addresses* that are relevant to a given *message* or a *meeting*, so that they can be added to the recipient

[1]The Enron corpus has been made available to the research community [Klimt and Yang 2004].
[2]In general, an email address and its person name alias may be both specified in the email's header; in the Enron corpus, however, there is a direct mapping between the email address identifier and the employee's name. In addition, while this does not apply to the data represented in the figure, it is possible that one person node is an alias of multiple email address nodes.

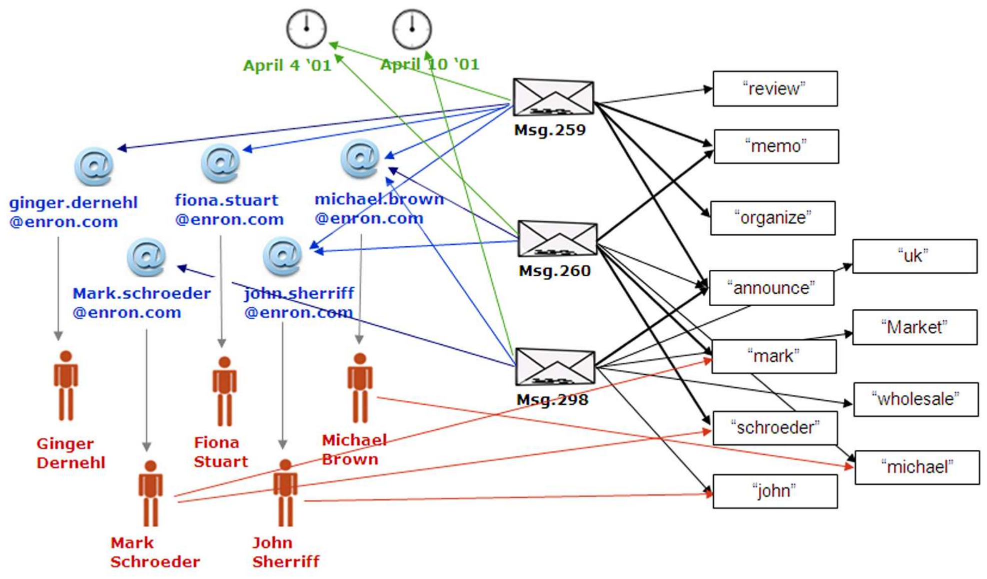

Fig. 1. A fraction of the email corpus of Richard Shapiro, an Enron employee, represented as a directed and typed graph. The graph includes entities of types *message*, *term*, *date*, *person* and *email address*. Inter-entity relations are directed and typed: edges portrayed by thin black arrows denote *has-term* relation, wide black arrows – *has-subject-term* relation, green – *on-date* relation, light blue and dark blue arrows represent *sent-to* and *sent-from* relations, respectively; grey arrows denote *alias* relation between an *email address* and the *person* it belongs to, and the red arrows denote a *as-term* relation between a *person* node and the *term* nodes that constitute the person's name. (Some of the nodes, including those denoting Richard Shapiro, have been removed from the figure for clarity.)

list of the message or meeting invitation, respectively. Another possible task is *person name disambiguation*. Assume that an incoming email message includes a common person name like "Mike", where it is not obvious who the person that this name mention refers to is; ideally, it would be possible to find the *person* in the corpus that is the correct reference, given the name *term* and the relevant *message*. Finally, one may be interested in searching for *messages* that are related to a given *message*, or more specifically, try to recover a message thread.

In order to address the specialized and typed search PIM tasks described, we will extend the standard IR model by letting a query be a set of nodes, and the answer be a set of "similar" nodes of a specified type. Traditional IR is therefore a special case in this setting, where the query only consists of *term* nodes, and the answer is always documents (*messages*). A major challenge is to generalize the similarity measure used in IR to rich typed graph schemes, such as the one portrayed in Figure 1.

We use a random-walk based similarity measure to perform search in typed graphs. In particular, we apply finite graph walks following the Personalized PageRank paradigm [Page et al. 1998; Haveliwala 2002]. This type of graph walk and its variants can be viewed as propagating "similarity" from a start node through

Task	Input	Output
Traditional IR	*terms*	*message*
Finding email aliases	*terms* (person's name)	*email address*
Finding meeting attendees	*meeting*	*email address*
Person name disambiguation	*term* (name mention) + *message*	*person*
Threading	*message*	*message*

Table I. Example PIM tasks

edges in the graph—incidentally accumulating evidence of similarity over multiple connecting paths. The resulting similarity metric is used as a *tool for performing search* across the graph nodes.

Previously, random graph walks have been used for estimating word dependency distributions [Toutanova et al. 2004]: in this case, the graph was 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 and other applications (e.g., [Xi et al. 2005; Collins-Thompson and Callan 2005]). In contrast to these works, we consider a *general-purpose* framework: rather than construct a special graph to solve a particular problem, we assume a graph representation that naturally models a semi-structured dataset, so that various types of queries are performed using the same underlying graph.

Given the rich graph representation and task diversity, we do not expect a single measure of similarity to be optimal for all query types. We therefore consider *learning* as a paradigm for improving the graph-walk based similarity metric for specific types of information needs. Towards this goal, we assume a supervised setting, where sample queries and relevance judgments are available for a task of interest. Previously, researchers have suggested improving graph walk based similarity in relational graphs by learning edge weights parameters in the graph [Diligenti et al. 2005; Nie et al. 2005; Agarwal et al. 2006]. The assumption underlying this approach is that for a particular task, some relation types should be favored by the graph walk over others. We adapt an error backpropagation gradient descent algorithm [Diligenti et al. 2005] for tuning the graph weights using finite graph walks. In addition, we suggest *reranking* as an alternative and complementary approach that learns to re-order the rankings produced by the graph walk process. While the graph walk similarity measure reflects structural information in the graph, it only considers local phenomena, being a memoryless process. Reranking, in contrast, can capture high-level properties of the graph walk. We propose a set of generic high-level features that describe a target node in terms of the paths traversed from the query distribution in reaching that node. In addition, task-specific engineered features can be accommodated in the reranking model.

Experimental results are given for three different PIM tasks— person name disambiguation, threading and alias finding—and eight corpora, including personal email collections derived from the public Enron corpus as well as other real email corpora. We show that the different tasks considered can be addressed as queries in the graph-based framework, where the graph walk based similarity gives good results and learning task-specific similarity measures lead to significant gains in performance. It is demonstrated that incorporating high-level information using reranking is highly effective. Overall, weight tuning and reranking are complementary, where the combination of these learning approaches often yields the best performance.[3]

The paper is organized as follows. In Section 2 we provide definitions of the graph and the graph walk method, as well as the corresponding query language and a user feedback mechanism. Section 3 formulates the learning settings, and outlines the weight tuning and reranking learning algorithms. In addition, a generic set of reranking features is presented, and methods for feature computation are discussed. In Section 4 we provide a detailed schema for representing email as a graph, and explain how email-related tasks can be cast as queries. The email corpora that we experiment with in this research are described in Section 5. The experimental setup and results on the set of three tasks evaluated are detailed in Section 6. Additional empirical results that evaluate the sensitivity of the framework to the parameter settings are included in Section 7. Section 8 discusses our observed running times and scalability issues. The paper concludes with a review of related work, conclusions and suggestions for future work.

2. THE FRAMEWORK

We begin with the formulation of the framework, presenting the underlying notations and definitions. We then follow with a detailed discussion of the framework's properties.

2.1 Definitions and Notation

The graph schema and user interface are specified below. The interface 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 optional component of the user-system interface is *user feedback* that includes judgments about which nodes are relevant to a query, for a given *task*.

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

[3]This article is an extension and unification of several conference and workshop publications [Minkov et al. 2006; Minkov and Cohen 2006; 2007]. The main novel contributions made in this article include: a detailed presentation of the framework, including illustrative examples and discussions (Section 2), experimental corpora analysis (Section 5), evaluation of the threading task against a TF-IDF baseline that considers meta-data (Section 6.2), evaluation of the alias finding task on two corpora, compared with a single corpus previously (Section 6.3), evaluation of the framework's sensitivity to parameter values (Section 7), detailed query processing times, along with a discussion of scalability (Section 8), and a comprehensive review of related work (Section 9).

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.

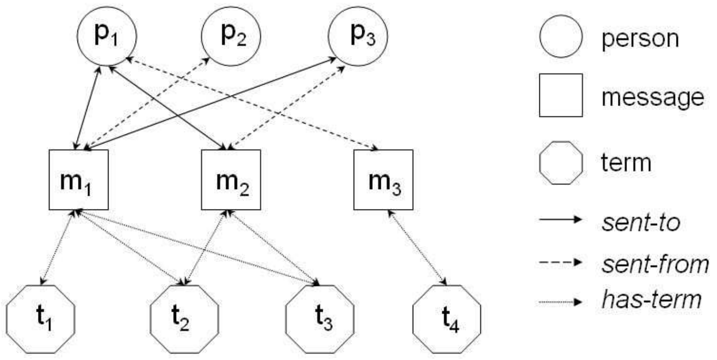

Fig. 2. A simple example of the considered graph schema

For example, consider the graph depicted in Figure 2. 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 *Query Language.* We are interested in inducing a general similarity measure between the graph nodes.[4] We take an information retrieval approach, where given a query, which is a combination of entities (nodes), a list of entities is returned to the user, ranked by their similarity to the query. Formally, we define a query language, as follows.

DEFINITION 1. *A* query $< V_q, \tau_{out} >$ *includes an initial distribution* V_q *over nodes and a desired output type* τ_{out}*, where a* response *to the query* $< V_q, \tau_{out} >$ *is a ranked list of nodes* z *of type* τ_{out}*.*

Consider the example graph in Figure 2. In the described domain, one may wish to find *persons* that are related to a particular *term*, such as "learning". The

[4]We use the words similarity and relatedness interchangeably.

relevant query in this case would be $< V_q = \{t_2\}, \tau_{out} =$'*person*'$>$ (where it is assumed that the term "learning" is represented by the graph node t_2).

2.1.3 *Tasks and Feedback.* Queries can be specified in an ad-hoc fashion by a user. However, it is reasonable that particular query types, or *tasks*, will be executed frequently in a given domain. In order to define a task, let us first define the notion of a *relation* in the graph:

Definition 2. *A* relation *denotes a distinct semantic meaning that exists between a subset of graph nodes* $V \times V$.

For example, a graph node of type *message* may be associated with several other *message* nodes in the graph, by virtue of belonging to same thread.[5] Alternatively, a subset of *message* nodes may be associated to each other semantically due to discussing the same topic, project, etc. Similarly, an association between *term* nodes in the graph and *person* nodes may describe the semantic relation of "an expert on", or some other relation. Since such relations are latent in the email corpus, they are not explicitly represented in the graph schema. In general, there is a large range of semantic relations possible, where a query is assumed to correspond to some relation of interest.

While each individual query seeks a single underlying relation, multiple queries, specifying different query distribution V_q, may share the motivation of retrieving nodes associated to the query with the same relation. This leads us to the following definition of a *task*.

Definition 3. *A* task *denotes a distinct relation r sought between a query and the nodes in the graph. Queries Q_1 and Q_2 are instances of the same task 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.*

In other words, we use the term *task* to describe search in the graph for a relation that is explicitly specified. Consider the queries $< V_q = \{t_2 =$"learning"$\}, \tau_{out} =$'*person*'$>$ and $< V_q = \{t_3 =$"recruiting"$\}, \tau_{out} =$'*person*'$>$. These queries may be instances of the same task, where the relation sought r is "an expert on". However, queries that specify *term* nodes and retrieve *person* entities may reflect a different flavor of similarity, r'; for example, a user who specifies the query $< V_q = \{t_1 =$"Bill"$\}, \tau_{out} =$'*person*'$>$, may be interested in retrieving persons whose nickname is Bill. Additional tasks are detailed in Table I. In our framework, it is possible to disambiguate the user's intention by specifying the relevant task along with a query $< V_q, \tau_{out} >$.

We distinguish between different relations in the graph, considering them as specialized cases of general inter-entity similarity. A key feature of the proposed framework is that different tasks are performed using the same underlying graph. If the underlying user intention (task) is not known, then it is expected that a 'default' similarity measure will result in useful performance for arbitrary queries. Nevertheless, we are interested in enhancing the general measure to reflect a particular similarity flavor of interest, in cases where the underlying task is known. In

[5]We consider this *same-thread* relation to be semantic rather than structural, since it may not be easily derived from the corpus; therefore, it is not represented explicitly in the graph (Figure 1).

order to support learning of specialized similarity measures per task, we next define the concept of *user feedback*.

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

Consider the query mentioned above, $< V_q = \{t =\text{“Bill”}\}, \tau_{out} =\text{‘}person\text{’}>$, where the underlying task is to find nodes in the graph that denote persons who are called “Bill”. User feedback may specify “William Scherlis” as a correct answer, and “William Cohen” (who does not use the nickname Bill), as well as other person nodes, as incorrect responses.

We use user labels, denoting node relevancy for learning task-specific similarity measures, as well as for evaluating the quality of the rankings produced in response to a query; a good response includes the relevant nodes at the top of the ranked list and the irrelevant nodes at lower ranks.

Finally, we notice that a “user” may not be a human being, but a machine application, which conducts automatic information processing. As we shall see, in some cases labels can be obtained automatically or semi-automatically.

2.2 Graph Walks

Several graph walk variants may be applied to derive entity similarity in graphs. We conduct a finite graph walk, where we follow the *Personalized PageRank* random graph walk model, applying it to entity-relation graphs.[6] We next introduce the Personalized PageRank graph walk algorithm. In addition, we define the underlying graph's edge probabilities, where edge weight parameters, together with the graph's topology, determine the probability of transitioning from a given node to its neighbors at each step of the walk. Finally, we discuss the properties of the induced graph-walk based similarity measure. Graph walk variants with similar properties can also be applied in this framework.

2.2.1 *Personalized PageRank.* We first describe the general and well-known *PageRank* model [Page et al. 1998], which derives a measure of entity “importance”, or “centrality” in a hyperlinked network. *PageRank* 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. A Web surfer's 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 will move to node j, following an outgoing link from i. That is, a random walk process is constructed as follows:

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

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

[6]Another graph walk variant that has been used in similar settings is *Lazy* graph walks [Minkov et al. 2006].

to, i.e.

$$\mathbf{M}_{ij} = \begin{cases} \frac{1}{|ch(i)|} & \text{if there is an edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $ch(i)$ is the set of nodes that have an outgoing link from i (the children of i).[7] The damping factor γ prevents the chain from getting stuck in small loops [Brin and Page 1998]. This means that Equation 1 is ergodic and has a unique stationary distribution V_* (i.e., V_d converges to V_*). 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.

The idea of biasing the PageRank computation for the purpose of personalization was first suggested in [Page et al. 1998]. 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 [Richardson and Domingos 2002], for example, suggests that the surfer only follows links to pages whose content has been deemed relevant to a given query. In the 'topic-sensitive' search [Haveliwala 2002], the surfer is biased to reset his or her search uniformly over pages pre-categorized as relevant to a given topic.

The *Personalized PageRank* is defined as follows [Page et al. 1998]:

$$V_{d+1} = \gamma V_0 + (1-\gamma)\mathbf{M}V_d \tag{3}$$

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 1), in which V_0 is uniform over all of the graph nodes. In our settings, V_0 corresponds to the query distribution V_q.

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 [Jeh and Widom 2003; Fogaras et al. 2005; Cohen and Minkov 2006]. Specifically, the Personalized PageRank probability $Q(z|x)$ of reaching z in an infinitely-long walk from x is also defined as:

$$Q(z|x) = \gamma \sum_{d=1}^{\infty} (1-\gamma)^d Q(x \xrightarrow{=d} z) \tag{4}$$

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

$$Q(x \xrightarrow{=d} z) = \sum_{y} \Pr(x \longrightarrow y) \cdot Q(y \xrightarrow{=d-1} z) \tag{5}$$

and

$$Q(x \xrightarrow{=0} z) = 1, if x = z. \tag{6}$$

[7]Pages with no outbound links are assumed to link out to all other pages in the collection.

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. As shown by Equation 4, due to the reset probability γ, the paths between x and a destination node z are weighted exponentially lower as their length increases. In practice, this means that the infinite graph walk probabilities can be effectively approximated by limiting the graph walk to a finite number of steps k [Toutanova et al. 2004; Fogaras et al. 2005; Cohen and Minkov 2006].

2.2.2 *Edge Weight Parameters.* The graph walk process (and accordingly, the similarity measure generated) is determined by the graph's topology.[8] In addition, the walk on the graph is controlled by a set of edge weight parameters Θ. This means that throughout the graph, edges of type ℓ are assigned a typical 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:

$$Pr(x \longrightarrow y) = \frac{\sum_{\ell \in L_{xy}} \theta_\ell}{\sum_{y' \in ch(x)} \sum_{\ell' \in L_{xy'}} \theta_{\ell'}} \tag{7}$$

where $ch(x)$ denotes the set of children of x (the nodes 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 the parent x.[9]

Recalling the example graph in Figure 2, 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: $\{\theta_{has-term}{=}2, \theta_{has-term-inverse} = 2, \theta_{sent-from} = 4, \theta_{sent-from-inverse}{=}3, \theta_{sent-to}{=}5, \theta_{sent-to-inverse}{=}4\}$.

Given this parameter set, the probability of reaching node t_1 from node m_1 in a single step, for example, is computed as follows:

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

The graph edge weights Θ can be set uniformly; randomly; manually, according to prior beliefs; or using a learning procedure, as discussed in Section 3.2.

2.3 The Framework's Properties

The Personalized PageRank algorithm, as described above, has several inherent preferences that determine how probability mass is distributed from a query to the graph nodes.

[8]The reset probability γ has negligible effect on the generated rankings; see a related discussion in Section 7.1.

[9]The PageRank scheme given in Formula 2 is a special case of Equation 7, where the graph includes a single edge type, and weights are distributed uniformly.

As illustrated by Equation 4, 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 described in Figure 2, 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* relation.

According to Equation 4, 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 similar to the query than nodes connected over fewer paths. For example, assume that edge weights are uniform, and a random graph walk is performed of two steps. 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:

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

whereas there are two paths leading to p_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 weights Θ provide another mechanism for affecting the probability flow in the graph. For instance, if $\theta_{sent-from} > \theta_{sent-to}$, than p_2 may be considered more similar than p_1 to the query $V_q = \{t_1, t_2\}$; otherwise, p_1 will be assigned a higher similarity score.

It is interesting to compare the node weighting scheme in Equation 7 to Inverse Document Frequency (IDF). Suppose that we restrict ourselves to a bi-partite graph that includes *terms* and *files* and allow only *has-term* (and *has-term-inverse*) edges, as is the case in traditional IR settings. Now consider an initial query distribution, which is uniform over the two terms "the aardvark". A one-step graph walk will result in a distribution V_1, which includes file nodes. The common term "the" will spread its probability mass into small fractions over many file nodes, while the unusual term "aardvark" will spread its weight over only a few files. Similarly, 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 $t_2 \overset{has-term-inverse}{\longrightarrow} m_1$.

3. LEARNING

While the graph walk similarity measure has many desired properties, we are interested in learning specialized measures to optimize performance per task. Next we outline the learning settings and describe two learning approaches: an algorithm that learns the edge weights Θ based on local information, which we adapt to finite graph walks; and reranking, using features that capture global properties of the graph walk. We propose a set of generic features and discuss their computation.

3.1 Learning Settings

We consider supervised learning settings. That is, it is assumed that labeled example queries e_i are provided ($1 \leq i \leq N$) for a task (relation) r of interest. Each example query specifies a different distribution over nodes V_q^i, and *user feedback* is available that indicates the relevancy of the graph nodes to each example query.

Example labeling scheme. Several labeling schemes have been suggested for learning to rank graph nodes, including: absolute scores, where target node probabilities are specified [Tsoi et al. 2003]; ordinal information, where ordinal values are assigned to nodes that represent their relative relevancy to the example query [Burges et al. 2005]; and pairwise node preferences, sampled from initially ranked lists [Agarwal et al. 2006]. 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 included in R_i are irrelevant to e_i.) This labeling scheme is adequate for well defined problems, in which a query corresponds to a finite set of "correct answers", and other nodes are considered irrelevant.

For instance, consider the task of *name alias finding*, where given a first name or a nickname, the goal is to retrieve the relevant person nodes. Example queries of this task are $V_q^1 = \{$*term*="Bill"$\}$, $V_q^2 = \{$*term*="Jason"$\}$ and so forth. For the first example, the user may specify the person node *person*="William Scherlis" as a correct answer; unspecified person nodes will be considered as incorrect answers, including the node *person*="William W. Cohen", implying that William Cohen is not referred to by the name Bill.

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

Learning goal. Learning is aimed at improving the initial rankings l_i^0, such that the 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 final output rankings l_i; that is, we are interested in producing modified lists l_i, in which the relevant nodes R_i 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 new (unlabeled) instances. These instances may correspond to the same graph that the labeled examples refer to, or other graphs that adhere to the same graph schema.

3.2 Edge Weight Tuning: Error BackPropagation

As discussed earlier, the graph edge weight parameters Θ, together with the graph topology, determine the transition probabilities in the graph (Equation 7), thus affecting the graph-walk generated similarity scores. The edge weight parameters reflect the assumption that the types of relations 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.

Several methods have been developed that automatically tune the edge weight parameters in similar settings, where edge weights are parameterized by edge type; we review these methods in Section 9.2.1. As an example of the weight tuning methods, we adapt an error backpropagation algorithm [Diligenti et al. 2005] to our framework of finite graph walks.

The algorithm operates via gradient descent, 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 [Ripley 1996]), as follows:

$$E = \frac{1}{|S|}\sum_{z\in S} err_z = \frac{1}{|S|}\sum_{z\in S}\frac{1}{2}(p_z - p_z^{opt})^2 \tag{8}$$

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 target nodes S. (The target nodes S can be sampled from the rankings of multiple queries, including relevant and possibly irrelevant nodes; in general, about a dozen of example nodes allow efficient learning in this paradigm for the tasks that we consider in this paper.)

The cost function is minimized by gradient descent with respect to every edge weight $\theta_{\ell'}$, using the update rule:

$$\theta_{\ell'}^{t+1} = \theta_{\ell'}^{t} - \eta\frac{\partial E^t}{\partial \theta_{\ell'}^{t}} = \theta_{\ell'}^{t} - \eta\frac{1}{|S|}\sum_{z\in S}\frac{\partial err_z^t}{\partial \theta_{\ell'}^{t}} \tag{9}$$

where t is the iteration index. The derivative of the error with respect to $\theta_{\ell'}$ 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., [Diligenti et al. 2005]). (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:

$$\frac{\partial err_z}{\partial \theta_{\ell'}} = (p_z - p_z^{opt})\sum_{d=0}^{k-1}\sum_{y\in U_z(d+1)} P(y, d+1 \to z, k)\cdot\frac{\partial p_y(d+1)}{\partial \theta_{\ell'}} \tag{10}$$

where k is the total number of walk steps, $p_y(d+1)$ is the probability assigned to node y by the graph walk after $d+1$ steps, and $U_z(d+1)$ denotes the set of graph nodes for which this probability is positive (i.e., the set of nodes that have been reached by step $d+1$); $P(y, d+1 \rightarrow z, k)$ is the total probability of reaching the target node z at the end of the walk (after k steps) starting from y at step $d+1$.

The derivative of the probability score of each intermediate node y with respect to $\theta_{\ell'}$ is computed based on the probability mass attributed to y by its parents, $pa(y)$, as specified in Equation 7. Explicitly, the derivative is as follows.

$$\begin{aligned}\frac{\partial p_y(d+1)}{\partial \theta_{\ell'}} &= \sum_{x \in pa(y)} p_x(d) \cdot \frac{\partial \frac{\sum_{\ell \in L_{xy}} \theta_\ell}{\sum_{y' \in ch(x)} \sum_{\ell' \in L_{xy'}} \theta_{\ell'}}}{\partial \theta_{\ell'}} \\ &= \sum_{x \in pa(y)} p_x(d) \cdot \frac{C(\ell', L_{xy}) O_x - C(\ell', L_{xy'}) \theta_\ell}{O_x^2} \qquad (11)\end{aligned}$$

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

The target function is not convex, and it is possible that the gradient descent procedure result in local minima [McInerney et al. 1989]. Common techniques to overcome this pitfall include executing multiple trials, using different initialization parameters (Θ^0, here), or simulated annealing. Given the cost function and the gradient, it is also possible to apply an optimization package such as L-BFGS [Nocedal and Wright 1999].

The gradient descent 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 practice, the processing time per iteration shortens drastically with caching.) As we will show, relatively few example nodes (S) give good performance [Diligenti et al. 2005]. 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 is learning to *reorder* an initial ranking. Reranking has been used in the past for meta-search [Cohen et al. 1999] and also for several natural-language related tasks (e.g., [Collins and Koo 2005; Collins 2002]). Typically, the ranked list of candidates is generated using local search methods, whereas 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.[10] For example, discriminative reranking has improved the state-of-the-art results of

[10]Due to cost considerations, reranking is typically applied to the top K candidates of the initially ranked list.

syntactic parsing, using sentence-level features to describe the high-probability candidate parse trees [Collins and Koo 2005; Charniak and Johnson 2005].

We 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 a ranked node using properties of the paths traversed to reach that node and discuss the computation of these features.

3.3.1 *Reranking Overview.* The reranking model represents each output node z_{ij} (candidate) as features, using m pre-defined feature functions $f_1, \ldots, f_m$. The goal in learning a reranking function is to maximize the margin between the candidate that is known to be the best answer and the other candidates. The reranking problem can thus be reduced to a classification problem by using pairwise samples [Shen and Joshi 2005]. Several algorithms have been used for reranking, including the Perceptron algorithm and its variants [Collins and Koo 2005; Shen and Joshi 2005] and Support Vector Machines [Shen and Joshi 2003]. We next describe a boosting approach, due to Collins and Koo [2005].

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

$$F(z_{ij}, \bar{\alpha}) = \alpha_0 log(p_{z_{ij}}) + \sum_{u=1}^{U} \alpha_u f_u(z_{ij}) \tag{12}$$

where $\bar{\alpha}$ is a vector of real-valued parameters. This linear function assigns node scores, $F(z_{ij}, \bar{\alpha})$, which are a weighted summation of the node's feature values. 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 algorithm minimizes the following exponential loss function on the training data:

$$ExpLoss(\bar{\alpha}) = \sum_{i} \sum_{j=2}^{l_i} e^{-(F(z_{i1},\bar{\alpha}) - F(z_{ij},\bar{\alpha}))} \tag{13}$$

where z_{i1} is, without loss of generality, a single correct target node.[11] The weights for the function are learned with a boosting-like method, where in each iteration the feature f_u that has the most impact on the loss function is chosen, and α_u is modified. Provided that the features are binary, closed form formulas exist for calculating the optimal additive parameter updates [Schapire and Singer 1999].[12]

Other researchers have also applied the voted Perceptron algorithm [Freund and Schapire 1999] and other Perceptron variants to learn the weights $\bar{\alpha}$ of the linear ranking function [Shen and Joshi 2005; Cohen and Minkov 2006].

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

[12]Please refer to Collins and Koo [2005], Figure 4, for a detailed description of this boosting algorithm adapted for ranking, which we apply in our experiments.

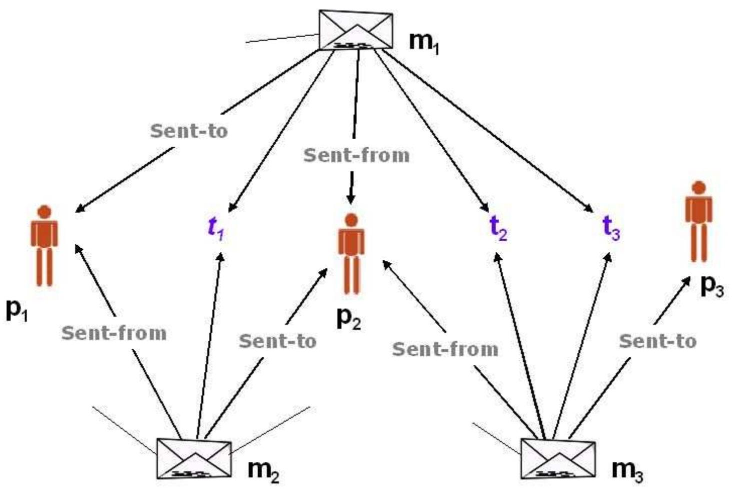

Fig. 3. An example sub-graph, showing the connecting paths between the *message* nodes m_1, m_2 and m_3. The nodes p_i and t_i in the graph represent *persons* and *terms*, respectively.

3.3.2 *General Graph-based Reranking Features.* Arbitrary features can be used in the reranking procedure. We suggest several generic, task-independent, features that describe the output nodes in terms of the paths traversed to reach these nodes. The features proposed 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 unigrams* - features indicating whether a particular edge label ℓ was included in the set of paths leading to the output nodes.

—*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 N 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 expected to be relevant to the query.

As an example, consider the sub-graph depicted in Figure 3. Suppose that a task of interest is *threading*, where given a message, the goal is to retrieve other messages that are a response to this message, or otherwise, messages that the specified message responds to. For the example query $V_q = \{msg{=}m_1\}$, the ranked list generated by a graph walk is likely to include the messages m_2 and m_3 among the top ranks, as both nodes are linked to m_1 over several short connecting paths. In order to represent these nodes in terms of the feature templates, we first recover the set of paths linking the query m_1 and each of the target nodes. Overall, the

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

Table II. 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 and walk length $k = 2$.

node m_2 is reached over three paths according to Figure 3 (the sub-graph shown is assumed to contain all of the relevant connecting paths, up to length 2), including:

$$\begin{array}{ccccc} m_1 & \xrightarrow{sent-to} & p_1 & \xrightarrow{sent-from-inv} & m_2 \\ m_1 & \xrightarrow{has-term} & t_1 & \xrightarrow{has-term-inv} & m_2 \\ m_1 & \xrightarrow{sent-from} & p_2 & \xrightarrow{sent-to-inv} & m_2 \end{array}$$

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

$$\begin{array}{ccccc} m_1 & \xrightarrow{sent-from} & p_2 & \xrightarrow{sent-from-inv} & m_3 \\ m_1 & \xrightarrow{has-term} & t_2 & \xrightarrow{has-term-inv} & m_3 \\ m_1 & \xrightarrow{has-term} & t_3 & \xrightarrow{has-term-inv} & m_3 \end{array}$$

The representation of the target nodes m_2 and m_3 as features is shown in Table II. The edge bigram features represent the types of edge sequences of length 2 traversed in the paths to each node. In the example, the query distribution includes 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 weights can denote the count of the edge n-gram sequence in the set of connecting paths; or, feature weights can denote the probability mass that was transmitted through each edge type (unigrams) from the query nodes to the target node (see Section 3.3.3).

Intuitively, given the features represented in Table II, message m_2 is more likely to belong to the same thread as m_1, compared with m_3. The reason for that is that 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. Reranking is therefore expected to assign high weights to the function parameters α_i that correspond to these features. Notice that manipulating the edge weights cannot capture this long-range phenomena. For instance, the sequences *sent-from.sent-from-inv* or *sent-to.sent-to-inv* include the same individual edge types as the sequences above, but are less indicative of a thread, or email response, at path level.

The proposed feature templates are general, in the sense that they are applicable to any task phrased as a query in the graph. In addition to this generic feature set, the design of additional task-specific features may improve performance further. In particular, other properties of the set of connecting to the target node may be represented as features; e.g., features that include information about the nodes visited in the course of the graph walk may be useful for certain problems. In addition, external information, which is not included in the graph but considered relevant for a given task, can also be encoded as features.

3.3.3 *Feature computation.* In this work, we compute the feature vectors for the top K nodes retrieved that are to be reranked. This means that feature extraction takes place after the graph walk is completed. Given the set of connecting paths to each of the top K nodes, extracted via the path unfolding procedure, it is straightforward to derive the feature values (see example in Section 3.3.2).

Alternatively, a number of features describing the set of paths from the query distribution V_q can be computed in the process of executing the graph walk. An algorithm for computing the graph walk and the node feature vector representation concurrently is given elsewhere [Cohen and Minkov 2006]. The cost involved in computing the feature function on-the-fly is constant per each node visited. This approximately doubles the cost of the graph walk computation. Maintaining n-gram edge sequence features, however, requires memory of size $|\Theta|^n T$, where T is the number of nodes traversed in the walk.

Unlike the weighted tuning approach, reranking requires some overhead over the graph walks–namely, the feature vectors for the top K nodes retrieved need to be computed as part of query execution, before the reranking function can be applied. Another concern is that while edge label *bigrams* correspond to a relatively small space, higher order *n-grams* may translate to a large feature space. Given a limited number of training examples, this may lead to over-fitting. In case that high-order n-grams are incorporated, it is therefore recommended to apply techniques such as feature selection or regularization.

4. PERSONAL INFORMATION MANAGEMENT (PIM)

There are several motivations for applying our framework to this domain. First, 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 [Hsiung et al. 2005; Holzer et al. 2005]. In addition, they can be related via common key words that appear in the relevant correspondence in the email corpus [McCallum et al. 2005]. Such inter-personal relatedness is also tied to a time dimension. It is therefore desired to utilize the multi-facet information that is included in a personal information resource for relevant applications. Using graph walks, the various aspects involved in PIM are integrated.

Another motivation for applying our framework to the PIM domain is that the underlying graph is modular and can be easily extended to include additional entity

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

types of interest. For example, the graph can represent email entities along with nodes that denote meetings, user activities, folders, and so on.

Finally, personal information involves structured meta-data and text, which are naturally cast as an entity-relation graph. A direct graph representation of the data allows performing general queries using the same underlying graph. We will show that graph walks give good performance for arbitrary queries, and that learning can further enhance the graph-based similarity measure for a specific task of interest.

In the following, we suggest a schema for representing personal information as a graph. We then present a set of PIM tasks, and illustrate how these tasks can be all processed uniformly as queries in our framework.

4.1 PIM Graph Representation

An example graph that includes a small fraction of an authentic email collection is shown in Figure 1. The corresponding graph schema is detailed in Table III. The graph representation naturally models an email corpus in the sense that it forms a direct layout of the information included within the corpus. In other words, the entities and relations represented are extracted (parsed) from the email documents, according to document structure. The graph schema described in Table III was designed manually, and other variations are possible. In our schema, the graph entities correspond to objects of types *messages* and *terms* that are traditionally indexed in information retrieval, as well as *email addresses*, *persons* and *dates*. Directed graph edges represent relations like *sent-from* and *sent-to* between a *message* and the relevant sender or recipient *person* entities, respectively; similarly, an *on-date* edge is added between a *message* and its issue *date*. As shown, we distinguish between *has-term* and *has-subject-term* relations in linking between *messages* and the *terms* they contain. In addition, in the suggested schema, a person node is linked to a relevant email address with an *alias* relation, and to its constituent token values with an *as-term* edge.[13] Similarly, terms that are identified as email addresses are linked by an *is-email* edge type to the corresponding *email address* node. In some of the experiments described in this paper, 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., an inter-person organizational hierarchy, 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, *meetings* are linked via *term* and *date* nodes to *messages*. 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 email can be enhanced with meeting information.

4.2 PIM Tasks as Queries

The suggested framework can be used as an ad-hoc search platform in the PIM domain. The data included in the graph may describe personal information, where the framework can be used to serve one's personal data search and consolidation needs; or, the graph may relate to organizational-level data, if available.

Next we present a set of PIM-related tasks. While some of these tasks have been treated in the past using different approaches, we show that all of the tasks can be addressed uniformly as queries in our framework. The query representations of the reviewed tasks are shown in Table I.

[13]For example, the header line "From: John Smith <jsmith@fake.net>" corresponds to a *person* node named "John Smith" and an *email address* node named "jsmith@fake.net"; the two nodes are linked in our schema with an *alias* relation, the *person* node "John Smith" is linked to the *terms* "john" and "smith" with an *as term* relation, etc. In the general case, however, it is possible that the sender's name is not specified.

4.2.1 *Person Name Disambiguation.* Consider an email message containing a common name like "Andrew". Ideally an intelligent automated mailer would, like a human 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 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, a contextual query can be constructed, which includes both the name mention and the *message* node, adding valuable information for name disambiguation.

4.2.2 *Threading.* Threading is the problem of retrieving messages that belong to an email thread given a single message from the thread. As has been pointed out, 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 [Lewis and Knowles 1997]. It has also been suggested that once obtained, thread information can improve message categorization into topical folders [Klimt and Yang 2004].

As threads (and more generally, similar messages) are indicated by multiple types of relations including text, social network and time 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.

4.2.3 *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 requires searching for a message with the desired information in its 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.

5. PIM CORPORA

Following is a description of the corpora that we experiment with in this paper.

Management game. This corpus contains email messages collected from a management course conducted at Carnegie Mellon University in 1997 [Minkov et al. 2005]. 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.

Corpus	**Files**	**Nodes**	**Edges**
M.Game	821	6,248	60,316
Sager	1,632	9,753	112,192
Shapiro	978	13,174	169,016
Farmer	2,642	14,082	203,086
Germany	2,651	12,730	158,484
Meetings	346	3,239	27,366
Personal	810	11,136	113,224

Table IV. Corpora statistics, including the number of email files processed, and the total number of nodes and edges in the corresponding graphs

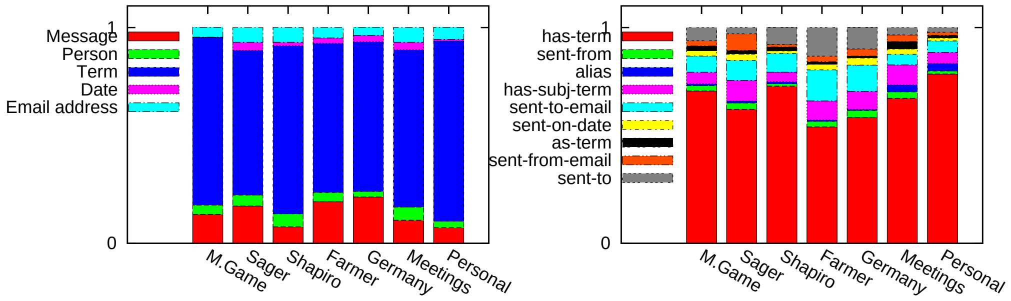

Fig. 4. Distribution of node types (left) and edge type (right) for every corpus

Enron. The Enron corpus is a collection of email messages sent and received by Enron's employees, which has been made available to the research community [Klimt and Yang 2004]. This corpus can be easily segmented by user: in the experiments, we used the saved email of several different Enron users. Overall, we consider four such corpora, of the users *E. Sager*, *R. Shapiro*, *D. Farmer* and *C. Germany*. 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. Finally, duplicate (identical) messages have been removed from the corpus.

Meetings. This corpus contains a subset of the second author'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 first author.

The statistics of corpora size and their graph representations are detailed in Table 5. 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 have a richer link structure.

Figure 5 further shows the distribution of the various node types, as well as the edge types, for all corpora. (While the inverse edges, e.g. *has-term-inverse* are not displayed in the figure, they give the same distribution, by edge symmetry.) As may be expected, the *term* nodes and *has-term* edges are dominant across corpora.

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

6. EXPERIMENTS AND RESULTS

In this section we present experimental results for the tasks of person name disambiguation, threading and alias finding. For each task we evaluate performance of an "out of the box" graph walk based similarity measure; in addition, we evaluate the performance of the specialized similarity metrics, learned using weight tuning and reranking. The graph-based results are compared against relevant baselines. A key property of the evaluation is that a non-subjective correct answer set is constructed per query.

Experimental Datasets. For every corpus and task evaluated, we created a pool of example queries, specifying the nodes which form the set of correct answers per query. In all experiments, the labeled examples available per corpus and task have been split into *training*, *development* and *test* sets. The examples included in the training set were used for learning. In contrast, the separate test examples were used for evaluating the rankings generated by the various methods, based on the annotated example labels. The *development set* was used for tuning purposes, allowing to optimize model parameters based on held-out data, where applicable. The generation process of the labeled examples and dataset statistics are described in detail later in this section.

Experiments. 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 using graph walks where the edge weights have been tuned (Gw:Learned). In all of the experiments reported we applied a reset probability $\gamma = 0.5$.

In order to avoid local minima in learning the graph edge weights using the error backpropagation 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.[14] Further, in all experiments we applied reranking on top of the *uniformly*-weighted graph walk results. That is, we evaluate reranking as alternative learning method to weight learning. For every example, the top 50 nodes have been reranked (denoted as 'Rerank'), where both train and development set examples have been utilized in training the reranking model.

Evaluation. 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"

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

answers. We evaluate the performance of the various methods in terms of Mean Average Precision (MAP). To define MAP, we first 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.[15] The *non-interpolated average precision* of the ranking is the average of *prec*(k) for each position k_i that holds a correct entry:

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

For 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 non-interpolated average precision of this ranked list is $(1+1+0.6)/3 = 0.87$. The Mean Average Precision (MAP) is the average of the non-interpolated precision scores, over multiple rankings (queries). Another evaluation measure used is *precision at rank 1*. This measure denotes the ratio of queries for which the top ranked entity is a correct answer.[16] Conceptually, this metric gauges the success of the underlying method in specifying a single answer. Finally, in tasks where a single correct answer is defined per query (aka, person name disambiguation), we also evaluate performance in terms of *mean recall at rank k*. The *non-interpolated recall at rank k* of a given ranked list is defined to be 0 for each rank $k = 0, ..., k_{i-1}$, where k_i is the rank that holds the single correct entry, and 1 for ranks $k \geq k_i$. The *(mean) recall at rank k* averages the recall scores at each rank k across the rankings of multiple queries. Thus, mean recall is in the range [0,1] at each rank k. Intuitively, this metric estimates the probability of retrieving the correct answer within the top k ranks. For example, *recall at rank 3*= 0.7, means that in 70% of the queries, the correct answer appears among the top 3 ranks of the retrieved lists.

In comparing the performance of different methods for a given dataset, we apply statistical significance tests, using a two-sided Wilcoxon test [Lehmann 1959], at significance level of 95%.

We next describe the experimental settings and datasets used for every task. The results are presented and interpreted in terms of the properties of the various approaches used.

6.1 Person Name Disambiguation

As described in Section 4.2.1, in the person name disambiguation task we are given a *term*, known to refer to a person's first name. The goal is to retrieve a ranked list of entities of type τ =*person*, where the relevant person appears at the top.

6.1.1 *Datasets.* Unfortunately, building a dataset of labeled queries for 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 (Table V).

The Management game corpus has been manually annotated with personal names [Minkov et al. 2005]. Along with the corpus, which contains correspondence between

[15] 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".
[16] It is possible that there be multiple correct answers per query.

	Train	**Dev.**	**Test**
M.Game	20	25	61
Sager	15	12	35
Shapiro	15	10	35

Table V. Person disambiguation dataset details.

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

Table VI. Example person name type distribution per dataset.

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 body of a message that did not match any person name included in the header. Overall, the types of name mentions identified include:

—*initials* – this is common in a message sign-off;

—*nicknames* – common nicknames (e.g., "Dave" for "David"); uncommon nicknames (e.g., "Kai" for "Keiko"); and names that bear no similarity to the formal full name, such as 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 VI 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 V).

6.1.2 *Baseline: string similarity.* As a baseline, we applied a reasonably sophisticated string matching method [Cohen et al. 2003]. 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 [Cohen et al. 2003] between the term and any single token of the personal name (ranging between 0 to 1). In addition, we incorporated a nickname dictionary,[17] such that if the name term is a known nickname of the person name,

[17]We used the same dictionary that was used for dataset generation.

	MAP		**Prec@1**	
	T	T+F	T	T+F
M.Game				
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: Reranked	$\mathbf{0.75}^{*+}$	$\mathbf{0.85}^{*+}$	$\mathbf{0.66}^{*+}$	$\mathbf{0.77}^{*+}$
Sager				
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^{*}	$\mathbf{0.71}^{*+}$
Gw: Reranked	$\mathbf{0.87}^{*}$	$\mathbf{0.82}^{+}$	$\mathbf{0.80}^{*}$	$\mathbf{0.71}^{*+}$
Shapiro				
String sim.	0.61	-	0.39	-
Gw: Uniform weights	0.78^{*}	0.61	$\mathbf{0.71}^{*}$	0.40
Gw: Learned weights	$\mathbf{0.78}^{*}$	$\mathbf{0.80}^{*+}$	$\mathbf{0.71}^{*}$	$\mathbf{0.71}^{*+}$
Gw: Reranked	0.76^{*}	0.78^{*+}	0.69^{*}	0.69^{*+}

Table VII. Person name disambiguation results: MAP and precision at rank 1. 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 similarity score of that pair is set to 1.

The results are given in Table VII, listing mean average precision and precision at the top rank. In addition, Figure 5 shows the average recall at every rank down to rank 10. As shown, the string similarity method is effective overall. Its performance is lower on the Management game dataset. Recall that the Management game corpus includes nicknames that have no literal resemblance to the person's name – these cases are not handled well by the string similarity approach. The Enron datasets, on the other hand, were generated automatically using lexical similarity information, and can therefore be more easily resolved using lexical similarity measures. The reason for the imperfect performance of the string similarity approach across all corpora, however, is the presence of 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.

6.1.3 *Graph walks.* We performed two variants of graph walk, corresponding to different methods of forming the query distribution V_q. In the first variant, V_q is concentrated on the *term* representing the name mention. In the other graph walk variant, V_q is a uniform distribution including the name *term* and the node denoting the *message* in which the name mention appeared. In both cases, the length of the graph walks has been set to 2.

In the first graph walk variant, where the query includes the name *term* only, the name *term* propagates its weight to the messages in which it appears; then, probability mass is further propagated to *person* nodes which co-occur with these

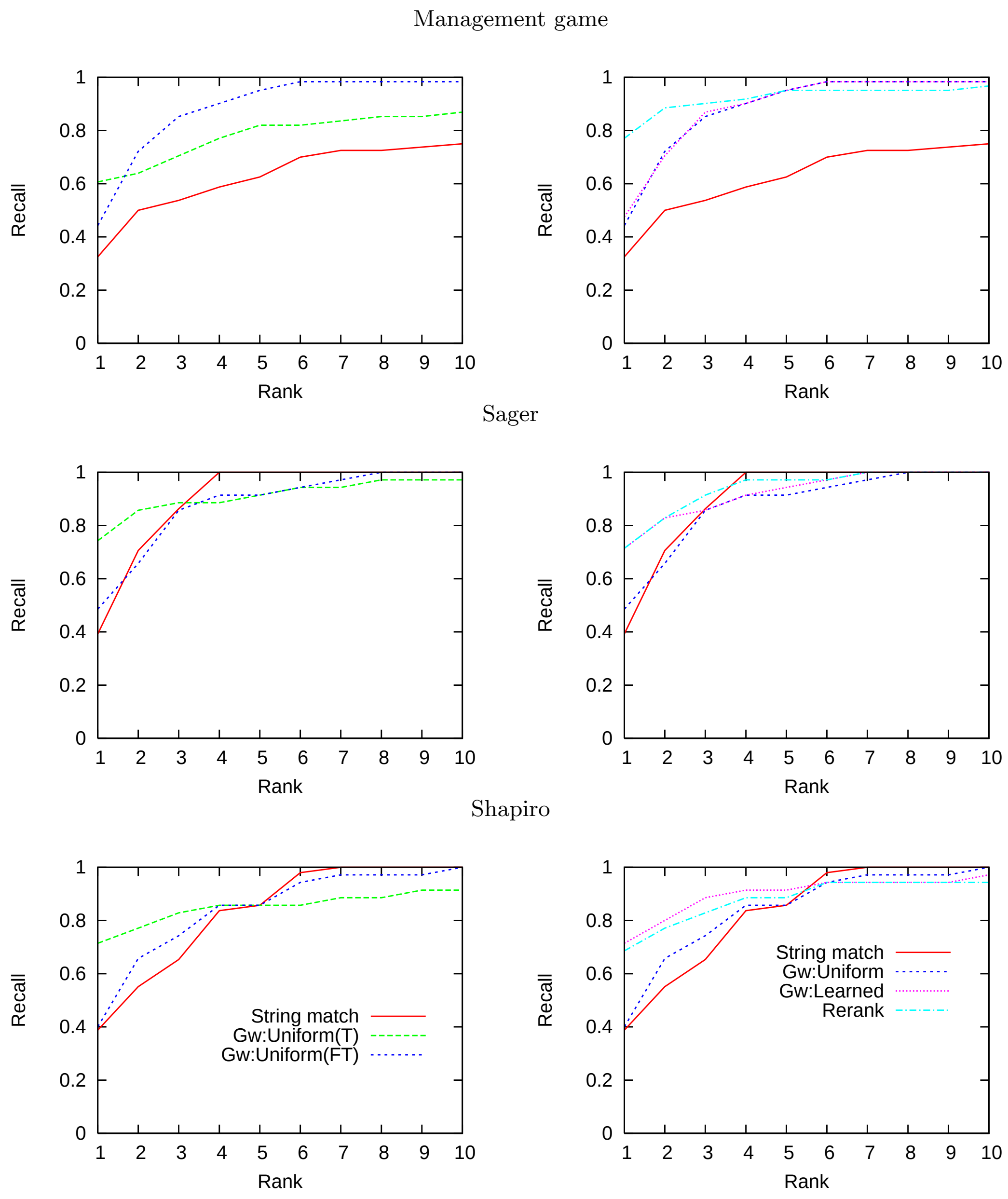

Fig. 5. Person name disambiguation test results: Recall at the top 10 ranks, for baseline and plain graph walk, where the query includes a term only (Gw:Uniform(T)), or term and file (denoted as Gw:Uniform(T+F)) (left); and for all methods using contextual queries (T+F) (right).

files. Note that in our graph schema there is a direct edge between *terms* to *person* names that include these terms (via the *as-term-inverse* relation), so that person nodes may receive weight via this path as well. The columns labeled "T" in Table VII give the results of the graph walk from the *term* probability vector in terms of mean average precision and precision at the top rank, and Figure 5 (left column, Gw:Uniform(T)) shows recall at each rank, down to rank 10. As shown by the results, the graph walk performance is preferable to string matching. For example, precision at the top rank is 52.5% and 32.5% on the management game dataset using the graph walk and string matching, respectively. More drastic improvements in accuracy are observed for the Enron corpora. The graph walks results are significantly better than string matching in terms of mean average precision. However, a graph walk originating from the *term* node only 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: this means that the top-ranked answer for ambiguous name terms (e.g., "Tom") will always be the same *person* node, where well-connected nodes are assigned higher weight by the graph walk. (For instance, "Tom Mitchell" may be ranked as the top *person* node per all of the mentions of the name "Tom" in the corpus, due to high connectivity of this node in the graph.)

We found that adding the *message* node representing the message in which the name mention appeared to V_q provides context useful for resolving ambiguous instances – this means that the correct *person* whose name is "Tom" is in general ranked higher than other persons with the same name in response to these *contextual* queries. Indeed, as shown in Figure 5 (left part, Gw:Uniform(T+F)), this search yields recall improvements compared to queries that include the name *term* only, leading to nearly perfect recall at rank 10. On the other hand, adding the *message* node results in attribution of probability score to nodes that link to the *message* 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 precision at top rank evaluation scores. This shortcoming will be addressed with learning.

6.1.4 *Learning.* We learn task-specific graph edge weights Θ^* using the error backpropagation method (denoted as Gw:Learned). Edge weight learning results in comparable performance to the graph walks using uniform weights for the queries including the *term* node only (T), across datasets. However, learning the graph edge weights significantly improves performance for the contextual search (T+F) for the two Enron corpora. As described earlier, the Enron datasets were created using an 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 lower variance in the observed connectivity patterns in the Enron datasets).

In reranking, we apply the *edge bigram* and *source count* features, as described in Section 3.3.2. In addition, we form *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 feature contains similar information to that used by the baseline ranking system.

As shown in Table VII, reranking substantially improves performance, especially

	Train	**Dev.**	**Test**
M.Game	20	25	80
Farmer	22	23	93
Germany	24	21	42

Table VIII. Threading dataset details.

for the contextual graph walk (see Figure 5). Indeed, the "noisy" nodes that were ranked at the top of the list retrieved due to their association to the *message* node but not to the *term* node included in the query, are demoted in the reranking process. In particular, high weights were assigned in the experiments to the string similarity feature and to the *source count* feature. The string similarity feature scores higher *person* nodes that resemble the name mention, and the *source count* feature favors *person* nodes that are linked both to the *term* and *message* nodes that comprise the query.

Overall, reranking gives the best results for two of the three datasets, including the challenging management game dataset. In the contextual search setting, reranking results are significantly better than the base graph walks with uniform weights for all datasets.

6.1.5 *Other Results.* Recently, specialized learning models have been suggested for the task of person name disambiguation in email collections that yield good performance. Specifically, a generative model [Elsayed et al. 2008] was reported to produce MAP of 0.91 and 0.91, and precision at the top rank of 0.86 and 0.88 on the Sager and Shapiro datasets, respectively. While these results are superior to ours, the graph walk based approach is more general. A detailed discussion of this and other work related to the person name disambiguation task is included in Section 9.3.1.

6.2 Threading

In the thread recovery task, as introduced in Section 4.2, we are interested in retrieving *messages* that are adjacent to a given *message* in a thread (i.e., both a direct 'parent' and direct 'child' messages are considered to be correct responses). We consider this task as a proxy to the task of finding generally related messages, as it is reasonable to assume that adjacent messages in a thread are most related to each other in a corpus.

6.2.1 *Datasets.* We created three datasets for the evaluation of the threading task, using the management game corpus and two Enron corpora. (Here we use the available corpora of two other Enron employees, Farmer and Germany.) Statistics about the constructed datasets are given in Table VIII. For each message including the reply prefix "Re:", its parent was identified using the quoted subject line and the time stamp.[18] We consider the immediate parent and child of the given file to be "correct" answers. About 10-20% of the query messages have both parent and child messages included in the corpus, otherwise only one message in the corpus is

[18]We manually checked about 25% of the identified message pairs, and found that in very few cases the reply mechanism was used for forwarding a message, or for changing the thread's topic to a distantly related matter; overall, less than 5% of the sampled pairs were such exceptions.

header	√	√	√	√	√	√	√	√
body	√	√	√	-	√	√	√	-
subject	√	√	-	-	√	√	-	-
reply lines	√	-	-	-	√	-	-	-
	MAP				**Prec@1**			
M.Game								
TF-IDF	0.63	0.58	0.40	0.42	0.52	0.42	0.26	0.18
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: Reranked	**0.77**$^{*+}$	**0.73**$^{*+}$	**0.59**$^{*+}$	**0.51**$^{*+}$	**0.68**$^{+}$	**0.62**$^{+}$	**0.44**$^{+}$	0.34*
Germany								
TF-IDF	-	0.56	0.40	0.42	-	0.34	0.22	0.24
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: Reranked	-	**0.72**$^{+}$	**0.65**$^{*+}$	**0.64**$^{*+}$	-	**0.56**$^{+}$	**0.51***	**0.51**$^{*+}$
Farmer								
TF-IDF	-	0.77	0.46	0.55	-	0.65	0.33	0.39
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: Reranked	-	**0.83**$^{+}$	**0.65**$^{*+}$	**0.61**$^{+}$	-	**0.70**$^{+}$	**0.56**$^{*+}$	**0.52**

Table IX. Threading Results: MAP and precision at rank 1. 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.

defined as a correct answer.

We created several versions of the data, where we varied the amount of message details that are 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.

6.2.2 *Baseline: TF-IDF.* As a baseline approach we apply a vector space model, in which a message is represented as a TF-IDF weighted vector of terms, and an inter-message similarity score is defined as the cosine similarity of their vectors. Our TF-IDF vector representation includes both message content and header. In order to represent message structure in the vector space, we represent words that appear in the header both as general and field-specific terms; for example, if the word "meet" appears in the subject line, the general term "meet" and a field-specific term "subject.meet" are added to the underlying message vector. Overall, in addition to general terms, the specialized term types represented include *subject*, *person* (corresponding to all tokens that appear in either the sender of recipient fields), and *date*.[19]

The TF-IDF weighting scheme used is the following:

$$w_{i,j} = tf_{ij} \cdot idf_i = tf_{ij} \cdot log_2(\frac{N}{df_i}) \tag{14}$$

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

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

6.2.3 *Graph walks.* To formulate this problem in the graph model, we let V_q assign probability 1 to the *message* node that corresponds to the focus message file, and let τ_{out} = "message". Graph walks of length 2 were applied.

The results show that the graph walk using uniform weights and the TF-IDF method give comparable performance. TF-IDF performance is slightly better when identical chunks of text, such as subject lines, are present in the query message and the adjacent messages in a thread (although, the difference in performance is not statistically significant). The results given header and body text information only are mixed. According to these results, processing email message similarity as semi-structured data using the graph-walk approach is comparable to using TF-IDF, where message structure is represented in the vector space.

6.2.4 *Learning.* Learning the graph edge weights results in (often significantly) improved performance across corpora, as shown in Table IX. 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 the graph walk output yields the best results out of the considered methods. In all of the experiments, the results of the graph walk with reranking are significantly better than the TF-IDF baseline, as well as better than the graph walks with uniform weights. The MAP result of the setting in which the least information is available, namely header information only, is impressive: with reranking, MAP is larger than 0.5 on all datasets.

Features that were assigned high weight by the learner included edge type bigrams such as:

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

[19]In earlier work, we used TF-IDF representation that ignored data structure [Minkov et al. 2006]; adding meta-data representation improves the TF-IDF similarity measure.

	Train	Dev.	Test
Personal	9	8	26
Meetings	8	-	6

Table X. Alias finding dataset details.

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

Note that while such sequences of relations can be readily identified as important in the graph framework, they cannot be readily modeled in the vector space representation. Sequential processes exist also for other email-related phenomena, e.g., workflows and social interaction [Carvalho and Cohen 2005]. We believe that learning using high-level features that model relation sequences will be beneficial in those settings as well.

6.3 Alias Finding

The task of alias finding is defined as the retrieval of the *full* set of *email addresses* pertaining to an individual (or a mailing-group). The query in this case may correspond to a *person* node; or, in our experiments it consists of a person's first name, represented as a *term*.[20] While a given first name may be ambiguous, we assume that the query is targeted at a specific person, and only consider the email addresses that belong to that person as correct. We assume that given a ranked list, the user can apply additional knowledge (about the person's affiliations, email domain etc.) to further select the relevant addresses. That is, we examine a setting that helps the user *recall* the set of relevant email addresses.

6.3.1 *Datasets.* We evaluate the task of alias finding using the *Meetings* and *Personal* corpora. The details of the datasets of labeled examples used are given in Table X. For each corpus a dataset has been created using manually labeled lists 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 first name.

6.3.2 *Baseline: String matching.* As a baseline, we use the string matching approach described earlier (Section 6.1.2), where similarity is computed between the query term and all of the email addresses known in the corpus. The results of applying string matching are given in Table XI in terms of MAP and precision at rank 1. String matching is successful 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 that are not similar to neither the person's first or last name. Such instances bound the recall of this approach.

6.3.3 *Graph walks.* In addition to the previously described edge types (Table III), we add here to the graph schema links that denote string similarity between

[20]Elsewhere, the settings in which the query included the person's full name represented as terms proved to be an easier problem [Minkov and Cohen 2006].

	MAP	**Prec@1**
Meetings		
String similarity	0.55	0.67
Gw: Uniform weights	**0.61**	**0.83**
Gw: Learned weights	0.55	0.67
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: Reranked	0.63	**0.85**

Table XI. Alias Finding Results

email address nodes. Specifically, email address pairs for which the Jaro similarity score is higher than a threshold of 0.8 are linked by two *string similarity* symmetrical directed edges. In general, graph walks are expected to be effective in realizing co-occurrence information and retrieving frequently used email address nodes. However, rarely used email addresses may be harder to find using graph walks, due to lack of co-occurrence information. Incorporating string matching links into the graph should therefore increase the graph walk recall.

We applied graph walks of $k = 3$ steps. As shown in Table XI, the performance of the graph walk is preferable to string matching on both 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. The graph walk gives better top rank precision as well. Unlike direct string matching, the graph walk associates the query terms with email addresses via multiple co-occurrence patterns. Some relevant paths in a 3-step walk are as follows:

$$\text{term} \xrightarrow{as-term^{-1}} \text{person} \xrightarrow{alias} \text{email address}$$
$$\text{term} \xrightarrow{has-term^{-1}/has-subj-term^{-1}} \text{message} \xrightarrow{sent-to/from-email} \text{email address}$$
$$\text{term} \xrightarrow{has-term^{-1}/has-subj-term^{-1}} \text{message} \xrightarrow{sent-to/from} \text{person} \xrightarrow{alias} \text{email address}$$

In addition, any *email address* reached via these paths may pass some probability mass to other email address nodes due to the added string similarity links (a *string similarity* link can be added as a suffix to the paths above), thus leading to increased recall.

6.3.4 *Learning.* Learning the graph edge weights resulted in comparable performance to the results of applying a graph walk using uniform weights on this task. While particular edge sequences (as detailed above) are expected to be meaningful for the alias finding task, weight tuning only uses local information. We conjecture that this limits weight tuning performance on this task.

For reranking, we used *edge bigram* features in the reported experiments. Reranking gave the best results on the two datasets in terms of top rank precision but lower MAP scores, compared with the base graph walks with uniform weights. We notice that the number of training examples available for both corpora is small, where the performance of discriminative learning usually improves with the number of training examples. Should a sufficient number of examples become available, then using edge-trigram features may be useful. (Adding feature selection to avoid overfitting is recommended in this larger feature space.)

The reported differences between the methods were not found to be statistically significant, possibly due to the limited size of the evaluation datasets.

7. SENSITIVITY TO PARAMETER AND DESIGN CHOICES

We have shown that multiple tasks can be successfully processed as queries in the general-purpose graph-based framework, and that learning often leads to significant gains in performance. In this section we further study the effect of the framework's parameters and design choices on performance. An empirical evaluation conducted shows that the length of the walk k affects performance, and that short graph walks are often preferable to longer walks. In terms of learning, it is shown that reranking gives better results than weight tuning on some tasks due to the modeling of graph-based high-level features and domain-specific features. Applying both weight learning and reranking gives the best results overall.

7.1 Graph walk parameters

The graph walk framework includes two parameters: the reset probability γ; and, as we perform *finite* graph walks, we limit the maximal length of the walk to a constant k.

7.1.1 *Reset probability.* In all of the experiments reported, the reset probability $\gamma \in (0, 1)$ was set to a default value of 0.5. The reset operation plays a crucial role in sharply (exponentially) suppressing probability distribution in the graph walk process as the distance from the query nodes grows (Equation 4); however, the magnitude of γ has negligible effect on the output relative node rankings [Page et al. 1998].[21] We have verified this empirically on all of our corpora and tasks.

7.1.2 *Walk length.* We evaluated the performance of the person name disambiguation, threading and alias finding tasks, using eight datasets in total, varying the walk length k. The results, in terms of mean average precision, are given in Table XII. The best result for every dataset is marked in boldface. As shown, the performance on the person name disambiguation task is similar for the management game corpus for varying 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 a lower MAP value for longer walks. In the threading task, performance is better for short walks of length $k = 2$ or $k = 3$ across all corpora. In all of the experiments, performance converged within $k = 8$ steps.

These results support our approach of conducting finite graph walks in two ways. First, as Personalized PageRank graph walks converge within a small number of iterations, finite graph walks over a small number of steps are shown to provide a good approximation for the infinite walks. Moreover, the results show that short

[21]The reset parameter γ must not equal 0 or 1, as both values violate the pattern of random walk with restarts defined by Equation 3.

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

graph walks give a more accurate similarity measure in some cases. This suggests that given a strong local evidence of inter-entity similarity in the graph (as reflected by the set of their connecting paths), propagating similarity over longer walks may introduce noise to the generated similarity metric. On the other hand, increasing the walk length can lead to higher recall; this is the reason that in the alias finding tasks, a three step walk gives better results than a graph walk of two steps.

How should one set the walk length k? in general, the walk length 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; due to the exponential decay over walk length, once the relevant nodes have been reached, the contribution of additional walk steps to their score would be marginal. In general, since only a few values of k need to be evaluated, it is straightforward to tune the walk length parameter empirically, using a set of tuning examples.

7.2 Impact of Learning

We have shown that both weight tuning and reranking are effective in adjusting the graph-walk based similarity measure for a given task (Section 6). However, as described earlier, these learning approaches differ in several respects. In terms of the phenomena that they model, weight tuning adjusts the graph parameters based on local information; in contrast, reranking can accommodate high-level information, such as the sequences of edge types that are traversed in the walk. On the other hand, reranking parameterizes the graph walk with a set of features, and this representation discards some quantitative information, compared with weight tuning. In addition, reranking is applied to the top K nodes in the ranked list generated by the graph walk. This means that reranking performance is bound by the quality of the initial graph walk, whereas weight tuning affects the graph walk process directly.

This section includes additional experiments that evaluate weight tuning and reranking as alternative learning methods, where we examine reranking performance using features derived from the graph walk only, comparing it directly to

Corpus	Gw:R	Gw:L	$\text{Rrk}^-_{Gw:R}$	$\text{Rrk}_{Gw:R}$	$\text{Rrk}^+_{Gw:R}$	$\text{Rrk}^-_{Gw:L}$	$\text{Rrk}_{Gw:L}$	$\text{Rrk}^+_{Gw:L}$
Person Name disambiguation								
M.Game	0.61	0.67	0.63	0.63	0.83^*	0.63	0.65	$\mathbf{0.85}^*\dagger$
Sager	0.65	0.81^*	0.48	0.72	$\mathbf{0.89}^*$	0.48	0.72	0.83^*
Shapiro	0.70	$\mathbf{0.80}^*$	0.39	0.52	0.75	0.39	0.52	0.79^*
Threading								
M.Game	0.52	0.59^*	$0.69^*\dagger$	$0.75^*\dagger$	-	$\mathbf{0.76}^*\dagger$	$0.74^*\dagger$	-
Germany	0.51	0.55^*	0.61	$0.66^*\dagger$	-	$\mathbf{0.70}^*\dagger$	$0.68^*\dagger$	-
Farmer	0.68	0.72	0.75	$0.83^*\dagger$	-	0.75	$\mathbf{0.87}^*\dagger$	-

Table XIII. Performance comparison (MAP) of graph walks using random weights (Gw:R) and learned weights (Gw:L), reranking of the graph walks with random weights using graph-based features ($\text{Rrk}_{Gw:R}$), excluding the graph walk scores as a feature ($\text{Rrk}^-_{Gw:R}$) and with additional string similarity features ($\text{Rrk}^+_{Gw:R}$). The combination of weight tuning and reranking is denoted as $\text{Rrk}_{Gw:L}$, $\text{Rrk}^-_{Gw:L}$ and $\text{Rrk}^+_{Gw:L}$ for the graph-based, excluding the graph walk scores as a feature and full feature sets, respectively. Results that are significantly different from Gw:R are marked with an asterisks, and results significantly different from Gw:L are marked with a dagger.

weight tuning. The contribution of the various types of information used in reranking is further reported and discussed. We also evaluate the utility of a combined approach: weight tuning and reranking can be applied in a pipeline fashion, where the results of the graph walk using the learned weights are input to the reranker. Our results indicate that the combined approach performs best in most cases.

7.2.1 *Reranking vs. weight tuning.* We compared weight tuning and reranking as follows. The Weight tuning algorithm was applied to each task and corpus using 5 randomly generated initial graph edge weight parameter sets. For every corpus, the parameter set for which the best end result was reached, Θ^0, was selected. Graph walks using the learned set of weights, Θ^G, were applied to evaluate the performance of weight tuning on the test set queries.

Reranking was trained separately, using both the *train* and *development* sets. For comparison reasons, the same set of initial random graph edge weights, Θ^0, was used to generate the graph walk results that are input to reranking. Thus, both methods are compared against the same baseline. (In contrast, in Section 6, reranking was applied to the output of graph walk using uniform weights.) For every example, the top $K = 50$ nodes were reranked.

Table XIII gives MAP results for the *person name disambiguation* task (applying the contextual version, where queries consist of file and term nodes) and *threading*.[22] The table includes the evaluation of graph walk using the selected initial weights Θ^0 (Gw:R) and with the tuned edge weights (Gw:L). In addition, it presents results of reranking the output of the initial graph walk (Gw:R), using features that describe the graph walk ($\text{Rrk}_{Gw:R}$). Specifically, the features applied in this reranking variant include *edge label bigrams* and the *source count* feature. Another variant of reranking evaluated is similar, except that the feature denoting the original scores of the graph walk is ommitted ($\text{Rrk}^-_{Gw:R}$) (Equation 12). Finally,

[22]The alias finding task is omitted, as the relevant datasets are smaller and do not allow significance testing.

we evaluate reranking using the full set of features available ($\mathrm{Rrk}^{+}_{Gw:R}$), including *string similarity* between the target and query nodes. As described earlier, the string similarity feature is task-specific and does not apply to threading. Results that were found significantly different from the graph walk with random weights are marked with an asterisk. Significance testing was conducted using a two-sided Wilcoxon test at 95% confidence level.

The results show different trends on the two tasks considered. Weight tuning is more effective than reranking using graph-based features on the person name disambiguation task. In fact, if the graph walk scores are excluded as a feature in reranking, then reranking performance is inferior to the graph walk. This result shows that some structural information that is modeled by the graph walk and weight tuning is not reflected in the path sequence features. However, applying reranking to this problem using the full feature set, including the string similarity feature, gives superior results. In this case, using relevant task-specific features allows reranking to eliminate noisy nodes from the ranked lists.

In the threading task, on the other hand, reranking gives significantly better results using path information only, compared with weight tuning. These results are consistent across corpora. Indeed, high-level information about the paths that connect the query to a target node is very useful in thread recovery. Specifically, an adjacent message in a thread is often a reply-to message, where a recipient becomes the sender and vice versa. This composite relation is captured by edge bigrams such as *sent-to*$\rightarrow$ *sent-from-inverse*. The weight tuning approach cannot model such multi-step dependencies, yielding smaller improvements on this task.

We conclude that reranking, while losing some structural and quantitative information that is considered by the weight tuning algorithm, can lead to preferable results to weight tuning on some tasks due to its modeling of global properties of the walk. Reranking's capacity of representing additional relevant features for a given task, leads to further gains. Importantly, the information conveyed by the graph walk is complementary to the edge sequence features, and its representation as a feature in the reranking model contributes to its performance.

7.2.2 *Combining Learning Methods.* Reranking is 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 scores assigned by the initial ranker are used as a feature by the reranker. We therefore consider the setting where the graph walk rankings are first improved using learning, and reranking is applied to the modified ranked lists.

Table XIII shows the results of reranking the lists generated by the graph walks using the learned weights Θ^{G}. The results of reranking using graph walk describing features only is denoted by '$\mathrm{Rrk}_{Gw:L}$' and by '$\mathrm{Rrk}^{-}_{Gw:L}$, if the graph walk scores are excluded from the reranking model. Results of reranking using the full feature set are denoted by '$\mathrm{Rrk}^{+}_{Gw:L}$. Results that are significantly better than weight tuning are marked with a dagger.

Overall, the combined approach gives the best performance for four out of the six datasets. Improving the graph walk's initial parameters with weight tuning prior to reranking is therefore recommended.

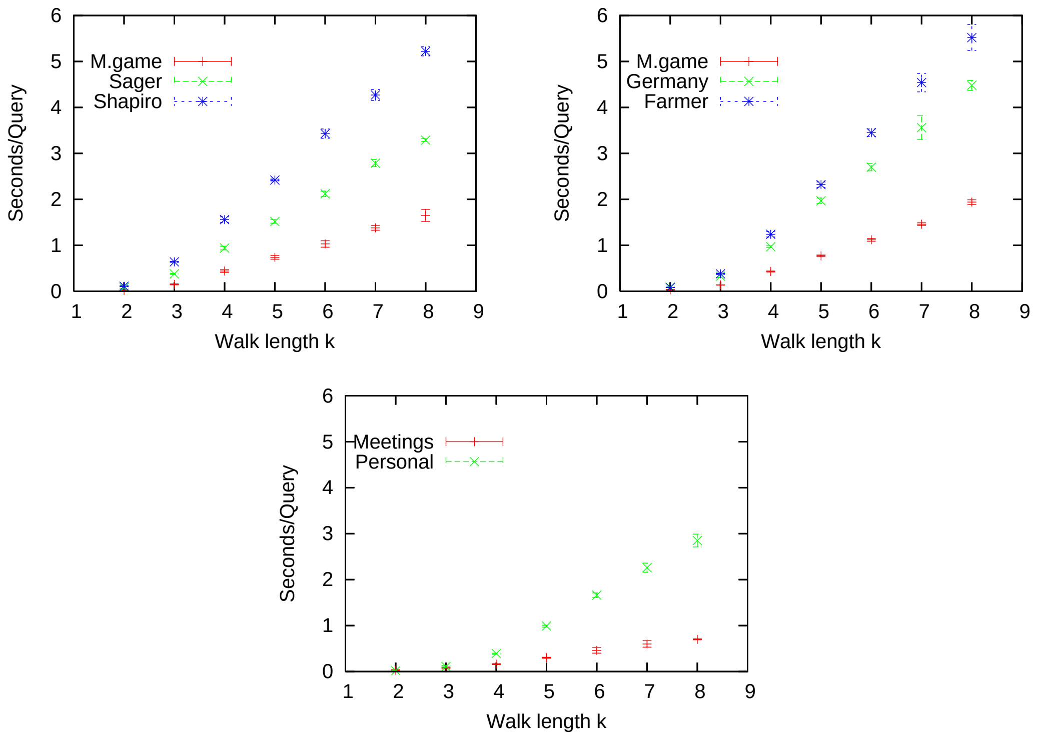

Fig. 6. Average graph walk completion time per query and one standard deviation (in error bars) for the different datasets, varying walk length k.

8. SCALABILITY

This section includes our observed query processing and discusses scalability issues.

8.1 Empirical running times

In our implementation of the graph walk, it is performed online in response to each query. Figure 6 shows the average graph walk 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 size of the experimental corpora ranges from 6K to 14K nodes, and from 60K to about 200K edges (see Table 5). 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: $\sum_{i=1}^{5} t_i/5$. The corresponding standard deviation is reported as well (shown in the error bars).

As shown in the figure, 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, for the same walk length k).

The times given in Figure 6 are satisfying for real-time applications. Earlier, we have shown (Section 7.1.2) that short graph walks yield performance that is preferable or comparable to longer walks in this domain. In addition, short walks of 2 or 3 steps are advantageous in terms of execution time, as they require an average processing time of a small fraction of a second.

On top of the graph walk, if reranking is applied, then encoding the top K nodes retrieved with graph walk describing features and scoring each node by the reranking model require additional overhead to the query response time. We sampled reranking processing times for the threading task, using the Germany corpus, over 5 runs. While executing the graph walk for $k = 2$ steps requires 90 milliseconds on average for this dataset, path unfolding and feature encoding require 2 milliseconds on average per node, and 0.6 milliseconds on average are required to score a node by the reranking model. Since $K = 50$ nodes are reranked per query, additional 130 milliseconds are required to complete query processing (*i.e.*, 220 milliseconds in total). In general, node scoring time is linear with respect to the number of features included in the reranking model (Eq. 12), whereas the complexity of the feature encoding operation is proportional to the graph walk complexity (see Section 3.3.3). Overall, the reranking overhead observed in our study is adequate for online settings. This overhead can be further controlled by altering K, the number of nodes to be reranked.

8.2 Larger Graphs

The corpora and tasks that we experiment with in this paper are reflective of personal information management, where we are given a personal corpus of email messages and meeting entries, and the user of the system is assumed to be the corpus owner or an automated personal assistant. However, one may be interested in applying the methods and tasks described to larger corpora, at company or corporate level. In general, we expect a graph that includes the email repositories of a few dozens of individuals to exceed a million nodes.[23] In addition, large graphs are expected to be generally denser compared with smaller graphs, i.e., have a higher branching factor. Thus, given a large graph, it may not be possible to accommodate it in memory and efficient processing of the graph walk is non trivial. A common solution is to store the graph in secondary memory, and cache information used frequently by the graph walk to improve response time.

Further, the scalability of applying the Personalized PageRank paradigm to very large graphs has received much attention in recent years, with the goal of providing a fast response to a query at run time. Most of that research is orthogonal to our work and can be readily incorporated into the framework's implementation.

In brief, there are two distinct approaches for applying the Personalized PageRank paradigm. The first approach is to compute the personalized views at query time. This requires an iterative computation over the graph, where response time is

[23]Interestingly, the vocabulary used in email correspondence is relatively limited, and we expect it to grow slowly with graph size; specifically, the union of the four Enron corpora used in the experiments includes roughly 24,000 unique terms. The number of *message* nodes, on the other hand, increases linearly with the size of the corpus.

linear with respect to the number of iterations and the number of edges traversed.[24] 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 $\Omega(n^2)$ [Fogaras et al. 2005]. A variety of approximation techniques have been proposed in the literature that are efficient [Jeh and Widom 2003; Balmin et al. 2004; Fogaras et al. 2005; Chakrabarti 2007].

9. RELATED WORK

There are many research areas related to the framework that we describe in this paper. In this section we discuss some of the relevant efforts, focusing on three general areas. Section 9.1 describes models for inducing a similarity measure between graph entities. In addition to graph-based methods, statistical relational learning is discussed as an alternative paradigm for evaluating inter-entity relations in structured data. Section 9.2 reviews related approaches for learning to rank graph nodes. A discussion of previous research in the PIM domain in general, and of the PIM tasks evaluated in this paper in particular, is given in Section 9.3.

9.1 Entity similarity in structured and semi-structured data

9.1.1 *Spreading activation.* A related paradigm to our framework is *spreading activation* (SA) over semantic or association networks: there, the underlying idea is to propagate *activation* from source nodes via weighted links through the network [Crestani 1997]. In order to prevent the activation signal from spreading over the whole network in a short number of steps, several heuristic constraints are used, including: *distance constraints*, which cease SA after a pre-defined number of links have been traversed; *fan-out constraints*, ceasing SA at nodes with high downstream connectivity; and *path constraints*, that divert the activation flow to particular paths in the network while stopping it from following other paths. In our framework, the graph walk scheme applies an exponential decay over path length, incorporating a soft *distance* constraint, and normalizing the graph edge weights to node probability distributions incorporates a probabilistic version of the *fan-out* constraint. We apply learning to adjust path preferences and the importance (weights) of different link types in the network, rather than set them manually.

9.1.2 *Graph walk based measures.* The idea of representing structured data as a graph is widespread in the data mining community, which is mostly concerned with relational or semi-structured data. Proximity search in databases represented as graphs has been suggested by Goldman *et al* [1998], where similarity was evaluated according to the shortest path between objects. Later models include *BANKS* [Bhalotia et al. 2002], *XRank* [Guo et al. 2003] and *SimRank* [Jeh and Widom 2002]. *SimRank* is a similarity measure adapted for graphs describing relational data. In this model, objects are recursively defined to be similar if they are related

[24]The complexity of online iterative graph walk is O(Ek), where E is the number of graph edges [Pan et al. 2004]; we take this approach in our implementation, as mentioned above.

to similar objects. 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. The SimRank measure does not consider edge weights. In addition, it is symmetric and fixed, whereas the graph walk similarity measure is query-dependent.

The *ObjectRank* model [Balmin et al. 2004] 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. The authors use an 'authority transfer' scheme that is set manually, to determine the weight per edge type. (Their scheme is equivalent to the edge weight parameters 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 only. Accordingly, we represent text as regular nodes within the graph, rather than process it separately. Importantly, we optimize 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* [2004], 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 [Toutanova et al. 2004] and query expansion in information retrieval [Collins-Thompson and Callan 2005]. In contrast to these works, we assume a more general setting, where data is represented as a graph with no target task pre-specified, and different types of queries are performed using the same underlying graph.

9.1.3 *Statistical Relational Learning.* Statistical relational learning (SRL) concerns the induction of probabilistic knowledge for multi-relational structured data. Various SRL paradigms have been proposed in recent years, including Probabilistic Relational Models [Friedman et al. 1999], Relational Dependency Networks [Neville and Jensen 2004], Markov Logic Networks [Richardson and Domingos 2006] and others. A general review of SRL is available elsewhere [Getoor and Taskar 2007].

We consider the Markov Logic Networks (MLNs) paradigm, an SRL model that generalizes finite first-order logic and Markov networks, as an alternative to the graph walk framework.

Markov logic combines Markov networks and weighted first-order logic formulae; in this paradigm, situations in which not all formulae are satisfied are considered less likely but not impossible. The information encoded in the graph can be represented as an MLN. Specifically, the inter-entity relations represented by the graph edges correspond to evidence predicates in MLNs (e.g., *sent-from*(x,y)). Long range relation between entities can be modeled in MLNs as rules. (For example, consider the rule: $\forall x \forall y \forall z$ *sent-from*(x,y)$\wedge$ *sent-to-inv*$(y,z) \Rightarrow$ *thread*(x,z).) The weights of the rules can be learned from examples. There are, however, several crucial differences between the graph walk paradigm and MLNs.

The expressive power of MLNs is larger compared with the graph framework, since it can also model any n-ary relations, whereas the graph representation only represents binary relations. However, 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. Another difference between the approaches is that MLNs require specifying (or learning) task-specific structures (rules) as a pre-requisite to network grounding. 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.

We performed an empirical evaluation of our framework and MLNs for the task of *threading* and the management game corpus.[25] The yielded test set result for this corpus using MLN was 0.69 in MAP. Since MLNs incorporate rules, this result should be compared against reranking, which gave MAP of 0.73 (Table IX). While performance is comparable between the two paradigms for this task and corpus, we were not able to apply MLNs to the larger Enron corpora because of large memory requirements due to network grounding. We therefore conclude that a main advantage of the graph walk framework over statistical relational learning methods such as MLN is better scalability.

9.2 Learning to rank graph nodes

We have applied an error backpropagation algorithm to learn the graph edge weights [Diligenti et al. 2005]. This section describes alternative weight tuning algorithms. In addition, we review previous research related to reranking in our framework.

9.2.1 *Edge Weight Tuning.* Several methods have been developed that automatically tune the edge weight parameters in extended PageRank models, where edge weights are determined by the underlying relation type. Nie *et al* [2005] have applied 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. Toutanova *et-al* [2004] have constructed

[25]We used the open source Alchemy system [Kok et al. 2005], where we applied the lazy MC-SAT algorithm for inference.

a special graph including diverse word-to-word relationships; they applied finite Personalized PageRank graph walks to induce smoothing 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.

In this work, we adapted an error backpropagation gradient ascent algorithm to learn the edge weights given labeled examples [Diligenti et al. 2005]. This method assumes labeled data in the form of correct and incorrect target nodes. Another gradient descent approximation algorithm has been presented that assumes feedback in the form of partial order preferences [Agarwal et al. 2006]. In this latter approach, the given pairwise constraints are added as a violation penalty to the cost function; the derivative with respect to the weight of each edge type is then computed by applying the chain rule, accompanying the regular PageRank iterations with gradient finding steps. The authors have shown that the time per iteration scales approximately linearly with the number of graph vertices and edges, and that the number of learning iterations grows slowly with the size of the graph. Overall, the training time is mildly superlinear to the graph scale factor.

Several authors have found that learning edge weight parameters leads to better generalization as opposed to learning general transition probabilities, where edge weights are unbounded. Agarwal *et al* [2006] experimented with maximum-entropy flow setting, with the goal of learning individual edge weights. They found that estimating a small number of global edge weights generalizes from training to test instances that involve completely different nodes, far away in the graph, using a much smaller number of examples. Toutanova *et al* [2004] experimented with tuning specialized edge weight parameters for different contexts; however, they report that assigning a fixed weight per edge type across all graph edges performs as well as more complex models.

9.2.2 *Graph Walks using Global Information.* The reranking approach has been applied in the past to a variety of structure prediction tasks, including parsing [Collins and Koo 2005; Collins 2002; Charniak and Johnson 2005], machine translation [Shen et al. 2005], semantic role labeling [Toutanova et al. 2005] and more. Structure prediction problems are usually decomposed into a chain of local decisions in order to apply efficient inference algorithms, such as dynamic programming. The resultant models, 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 violate 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 [Toutanova et al. 2005].

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 [Snow et al. 2005]. In particular, rich features sets were proposed that describe these paths [Culotta and Sorensen 2004; Bunescu and Mooney 2005]. To the best of our knowledge, we are the first to consider global features in graph-walk 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. In contrast to related works, we propose generic features that can be applied across domains.

9.3 Automated personal information management

In the last years, 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 [Bekkerman et al. 2004], automatic finding of experts at the enterprise using email resources [Balog et al. 2006; Petkova and Croft 2006], recommendation of recipients for a given message [Carvalho and Cohen 2008; Pal and McCallum 2006], identifying possible email leakage to wrong recipients [Carvalho and Cohen 2007], and more.

Unlike most previous works, the graph walk framework processes personal information as semi-structured data, where meta-data and text are all represented in a single graph. There are only a few previous works in the literature that integrate meta-data and text in email. For instance, a clustering approach has been proposed using multiple types of interactions in co-occurrence data [Bekkerman et al. 2005]. Another work [Aery and Chakravarthy 2005] proposes a graph-based approach for email classification. The authors 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.

A major 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, including ad-hoc searches. Previous works treated each task separately, optimizing the data representation and the techniques applied per task.

We next describe related works for each of the tasks considered in this paper.

9.3.1 *Person name disambiguation.* The task of person name disambiguation has been studied in the field of social networks and applied also to email data [Malin et al. 2005; Diehl et al. 2006]. Diehl *et al* [2006] have suggested to perform name disambiguation in email using traffic information, as derived from the email headers. In their approach, a candidate set is first generated, including network references with identical names to the ambiguous name mentions in a message, for which at least one email communication has been observed with the sender. They suggest a scoring formula based on the counts of message exchanges between each candidate and the sender, or between each candidate and all of the message recipients, summarizing over different ranges of history. Their approach uses social information only, as derived from email headers, while our approach integrates this information with email content and a timeline in a unified framework. In addition, we do not pre-filter a set of relevant candidates (thus bounding recall); instead, we rely on the graph walk to retrieve the relevant graph nodes based on network topology.

Recently, a generative model has been proposed for resolving name mentions in email [Elsayed et al. 2008]. The model can be thought of as a language model over a set of personal contextual references. More specifically, the model records mappings between person names, email addresses and nicknames, as deduced from email salutations and signatures. Given a new name mention, it is resolved by matching the contextual information associated with each known person and the name mention. Overall, the described approach is highly specific to the subject task. In addition, the model is feature specific; incorporating other types of evidence such as lexical similarity or information about meetings requires manual adaptation of the model. In contrast, our graph-based approach is general and modular.

9.3.2 *Threading.* Lewis and Knowles [1997] considered email threading as a retrieval problem; 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 and Harnly [2006] 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. The graph walk framework is more general, where it does not rely on task-specific features, as opposed to these previous works. Moreover, the approach of graph walk with reranking that has been proved successful for thread recovery can be applied in learning other tasks that model sequential phenomena; for example, the chaining of sequential speech acts [Carvalho and Cohen 2005].

9.3.3 *Alias finding.* The task of finding a person's set of email addresses in an email corpus given the person's *name*, as presented in this paper, is novel. This task is related, however, to the task of identifying email aliases (given an email address) in a corpus. Previously, researchers explored the information residing in social network co-occurrences on this task; this effort resulted in performance better than random [Holzer et al. 2005]. Others have attempted to combine social network information and string similarity measures [Hsiung et al. 2005]. Our approach allows integration of header information and string similarity measures, as well as email content and time information in a unified framework.

10. CONCLUSION

We described a general framework for inducing adaptive similarity measures in heterogeneous data represented as an entity-relation graph. The framework builds on existing graph-walk based paradigms that generate measures of structural similarity between entities in the graph. Previously, researchers have applied graph walks using carefully designed graphs with the goal of solving pre-defined problems. We showed 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 based similarity measure gives good performance in response to various queries. However, if labeled instances are available, then learning can be applied to generate a specialized similarity measure adapted to a specific relation sought. We suggested reranking as a method for learning to rank graph nodes that can use high-level information about the graph walk process; in particular, we proposed generic features that describe the set of paths traversed in reaching a target node from the query distribution. Reranking was shown to give better results in some cases than weight tuning, a learning approach that only considers local information about the graph walk. In addition, combining local and global learning was shown to be advantageous.

Interestingly, the empirical results that we observed in our experiments show that conducting short finite graph walks is both computationally efficient and also yields better top rank precision in some cases, compared with longer walks.

The paper presented a detailed case study where we applied the framework to the domain of personal information management. It was shown that multiple tasks in this domain can be processed uniformly as queries, as opposed to using different specialized techniques. An evaluation of the graph-walk similarity measure, augmented with learning, on the tasks of person name disambiguation, threading and alias finding was shown to give high levels of performance.

We believe that this framework is natural and adequate for performing search-related tasks in the personal information domain. A main advantage of the graph walk paradigm in this domain is that it naturally integrates the textual and the structured elements of the data, including social network and time information. In addition, the graph representation is modular, and it is straightforward to extend the graph schema to include other information sources available, such as entities that represent activities, relations that correspond to an organization chart, etc. While the graph walk and weight tuning effectively model entity associations, reranking allows further flexibility in adapting the generated similarity measure to different tasks using high-level and task-specific information. Finally, personal information corresponds to small to medium sized graphs. An average query is processed in a fraction of a second when short graph walks are applied, and applying reranking to a small subset of nodes requires another fraction of a second. Applying the framework in general, and Personalized PageRank in particular, to large corpora is possible though challenging. Improving the scalability of Personalized PageRank is an area of an active research, and ideas suggested by other researchers can be readily incorporated in an operative system.

There are many directions in which the framework presented can be extended. For example, the suggested graph schema supports binary relations, where one may wish to incorporate probabilistic relations, i.e., edges with varying levels of uncertainty. Another shortcoming of the framework is that it does not support fine text processing compared with language models. In addition, in this work we have set the graph schema manually; an interesting research direction is to construct the graph schema automatically from data. In particular, it is possible that including irrelevant or noisy information in the graph can degrade performance. If this occurs, then it is desired to eliminate such information from the graph.

In terms of learning, the learning settings that are assumed in this work can be relaxed and allow additional types of user feedback; that is, rather than consider binary signals about a node relevancy, partial node preferences or other forms of feedback may be provided. The learning procedures that are described in this work 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. Adding links to the graph may improve the quality of the responses to future queries.

A related question is whether 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. An open question is 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. Mechanisms for leveraging learning across tasks, perhaps based on recent work in transfer learning [Mihalkova and Mooney 2009; Wang et al. 2009], may be effective in these settings.

ACKNOWLEDGMENTS

The authors wish to thank Andrew Y. Ng, Christos Faloutsos, Tom Mitchell, Raymond Mooney, Hanghang Tong and Elchanan Mossel for useful discussions, and to the anonymous reviewers for their comments. This material is based upon work supported by Yahoo! Research and by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA), or the Department of Interior-National Business Center (DOI-NBC).

REFERENCES

Aery, M. and Chakravarthy, S. 2005. emailsift: Email classification based on structure and content. In *ICDM*.

Agarwal, A., Chakrabarti, S., and Aggarwal, S. 2006. Learning to rank networked entities. In *KDD*.

Anyanwu, K., Maduko, A., and Sheth, A. 2005. Semrank: Ranking complex relationship search results on the semantic web. In *WWW*.

Balmin, A., Hristidis, V., and Papakonstantinou, Y. 2004. ObjectRank: Authority-based keyword search in databases. In *VLDB*.

Balog, K., Azzopardi, L., and de Rijke, M. 2006. Formal models for expert finding in enterprise corpora. In *SIGIR*.

Bekkerman, R., El-Yaniv, R., and McCallum, A. 2005. Multi-way distributional clustering via pairwise interactions. In *ICML*.

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

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. 2002. Keyword searching and browsing in databases using banks. In *ICDE*.

Brin, S. and Page, L. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems 30*.

Bunescu, R. C. and Mooney, R. J. 2005. A shortest path dependency kernel for relation extraction. In *HLT-EMNLP*.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullende, G. 2005. Learning to rank using gradient descent. In *ICML*.

Carvalho, V. R. and Cohen, W. W. 2005. On the collective classification of email "speech acts". In *SIGIR*.

Carvalho, V. R. and Cohen, W. W. 2007. Preventing information leaks in email. In *SDM*.

Carvalho, V. R. and Cohen, W. W. 2008. Ranking users for intelligent message addressing. In *ECIR*.

Chakrabarti, S. 2007. Dynamic personalized pagerank in entityrelation graphs. In *WWW*.

Charniak, E. and Johnson, M. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL*.

Cohen, W. W. and Minkov, E. 2006. A graph-search framework for associating gene identifiers with documents. *BMC Bioinformatics 7,* 440.

Cohen, W. W., Ravikumar, P., and Fienberg, S. 2003. A comparison of string distance metrics for name-matching tasks. In *IIWEB*.

Cohen, W. W., Schapire, R. E., and Singer, Y. 1999. Learning to order things. *Journal of Artificial Intelligence Research (JAIR) 10*, 243–270.

Collins, M. 2002. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *ACL*.

Collins, M. and Koo, T. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics 31,* 1, 25–69.

Collins-Thompson, K. and Callan, J. 2005. Query expansion using random walk models. In *CIKM*.

Crestani, F. 1997. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review 11,* 6.

Culotta, A. and Sorensen, J. 2004. Dependency tree kernels for relation extraction. In *ACL*.

Diehl, C. P., Getoor, L., and Namata, G. 2006. Name reference resolution in organizational email archives. In *SIAM*.

Diligenti, M., Gori, M., and Maggini, M. 2005. Learning web page scores by error back-propagation. In *IJCAI*.

Elsayed, T., Oard, D. W., , and Namata, G. 2008. Resolving personal names in email using context expansion. In *HLT-ACL*.

Fogaras, D., Rácz, B., Csalogány, K., , and Sarlós, T. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics 2,* 3.

Freund, Y. and Schapire, R. E. 1999. Large margin classification using the perceptron algorithm. *Machine Learning 37,* 3.

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. 1999. Learning probabilistic relational models. In *IJCAI*.

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

Goldman, R., Shivakumar, N., Venkatasubramanian, S., and Garcia-Molina, H. 1998. Proximity search in databases. In *VLDB*.

Guo, L., Shao, F., Botev, C., and Shanmugasundaram, J. 2003. Xrank: Ranked keyword search over xml documents. In *SIGMOD*.

Haveliwala, T. H. 2002. Topic-sensitive PageRank. In *WWW*.

Holzer, R., Malin, B., and Sweeney, L. 2005. Email alias detection using social network analysis. In *LinkKDD*.

Hsiung, P., Moore, A., Neill, D., and Schneider, J. 2005. Alias detection in link data sets. In *Proceedings of the International Conference on Intelligence Analysis*.

Jeh, G. and Widom, J. 2002. Simrank: A measure of structural-context similarity. In *SIGKDD*.

Jeh, G. and Widom, J. 2003. Scaling personalized web search. In *WWW*.

Klimt, B. and Yang, Y. 2004. The enron corpus: A new dataset for email classification research. In *ECML*.

Kok, S., Singla, P., Richardson, M., and Domingos, P. 2005. 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.*

Lehmann, E. 1959. *Testing statistical hypotheses.* Wiley.

Lewis, D. E. and Knowles, K. A. 1997. Threading electronic mail: A preliminary study. *Information Processing and Management*.

Malin, B., Airoldi, E. M., and Carley., K. M. 2005. A social network analysis model for name disambiguation in lists. *Journal of Computational and Mathematical Organization Theory 11,* 2.

McCallum, A., Corrada-Emmanuel, A., and Wang, X. 2005. Topic and role discovery in social networks. In *IJCAI*.

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

Mihalkova, L. and Mooney, R. J. 2009. Transfer learning from minimal target data by mapping across relational domains. In *IJCAI*.

Minkov, E. and Cohen, W. W. 2006. An email and meeting assistant using graph walks. In *CEAS*.

Minkov, E. and Cohen, W. W. 2007. Learning to rank typed graph walks: Local and global approaches. In *WebKDD and SNA-KDD joint workshop*.

Minkov, E., Cohen, W. W., and Ng, A. Y. 2006. Contextual search and name disambiguation in email using graphs. In *SIGIR*.

Minkov, E., Wang, R., and Cohen, W. 2005. Extracting personal names from emails: Applying named entity recognition to informal text. In *HLT-EMNLP*.

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

Nie, Z., Zhang, Y., Wen, J.-R., and Ma, W.-Y. 2005. Object-level ranking: Bringing order to web objects. In *WWW*.

Nocedal, J. and Wright, S. J. 1999. *Numerical Optimization.* Springer Series in Operations Research.

Page, L., Brin, S., Motwani, R., and Winograd, T. 1998. The pagerank citation ranking: Bringing order to the web. In *Technical Report, Computer Science department, Stanford University.*

Pal, C. and McCallum, A. 2006. Cc prediction with graphical models. In *CEAS*.

Pan, J.-Y., Yang, H.-J., Faloutsos, C., and Duygulu, P. 2004. Automatic multimedia crossmodal correlation discovery. In *KDD*.

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

Richardson, M. and Domingos, P. 2002. The intelligent surfer: Probabilistic combination of link and content information in PageRank. In *NIPS*.

Richardson, M. and Domingos, P. 2006. Markov logic networks. *Machine Learning 62,* 1-2.

Ripley, B. 1996. *Pattern Recognition and Neural Networks.* Cambridge University Press.

Schapire, R. E. and Singer, Y. 1999. Improved boosting algorithms using confidence-rated predictions. *Machine Learning 37(3)*, 297–336.

Shen, L. and Joshi, A. K. 2003. An svm based voting algorithm with application to parse reranking. In *CONLL*.

Shen, L. and Joshi, A. K. 2005. Ranking and reranking with perceptron. *Machine Learning 60,* 1-3.

Shen, L., Sarkar, A., , and Och, F. J. 2005. Discriminative reranking for machine translation. In *HLT-NAACL*.

Snow, R., Jurafsky, D., and Ng, A. Y. 2005. Learning syntactic patterns for automatic hypernym discovery. In *NIPS*.

Toutanova, K., Haghighi, A., and Manning, C. D. 2005. Joint learning improves semantic role labeling. In *ACL*.

Toutanova, K., Manning, C. D., and Ng, A. Y. 2004. Learning random walk models for inducing word dependency distributions. In *ICML*.

Tsoi, A. C., Morini, G., , Scarselli, F., Hagenbuchner, M., and Maggini, M. 2003. Adaptive ranking of web pages. In *WWW*.

Wang, Z., Song, Y., and Zhang, C. 2009. Knowledge transfer on hybrid graph. In *IJCAI*.

Xi, W., Fox, E. A., Fan, W. P., Zhang, B., Chen, Z., Yan, J., and Zhuang, D. 2005. Simfusion: Measuring similarity using unified relationship matrix. In *SIGIR*.

Yeh, J.-Y. and Harnly, A. 2006. Email thread reassembly using similarity matching. In *CEAS*.

Received February 1986; November 1993; accepted January 1996