Learning Graph Walk Based Similarity Measures for Parsed Text

Einat Minkov[*]
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
einat@cs.cmu.edu

William W. Cohen
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
wcohen@cs.cmu.edu

Abstract

We consider a parsed text corpus as an instance of a labelled directed graph, where nodes represent words and weighted directed edges represent the syntactic relations between them. We show that graph walks, combined with existing techniques of supervised learning, can be used to derive a task-specific word similarity measure in this graph. We also propose a new *path-constrained* graph walk method, in which the graph walk process is guided by high-level knowledge about meaningful edge sequences (paths). Empirical evaluation on the task of named entity coordinate term extraction shows that this framework is preferable to vector-based models for small-sized corpora. It is also shown that the path-constrained graph walk algorithm yields both performance and scalability gains.

1 Introduction

Graph-based similarity measures have been used for a variety of language processing applications. In this paper we assume directed graphs, where typed nodes denote entities and labelled directed and weighted edges denote the relations between them. In this framework, graph walks can be applied to draw a measure of similarity between the graph nodes. Previous works have applied graph walks to draw a notion of semantic similarity over such graphs that were carefully designed and manually tuned, based on WordNet reations (Toutanova et al., 2004; Collins-Thompson and Callan, 2005; Hughes and Ramage, 2007).

While these and other researchers have used WordNet to evaluate similarity between words, there has been much interest in extracting such a measure from text corpora (e.g., (Snow et al., 2005; Padó and Lapata, 2007)). In this paper, we suggest processing dependency parse trees within the general framework of directed labelled graphs. We construct a graph that directly represents a corpus of structured (parsed) text. In the suggested graph scheme, nodes denote words and weighted edges represent the dependency relations between them. We apply graph walks to derive an inter-word similarity measure. We further apply learning techniques, adapted to this framework, to improve the derived corpus-based similarity measure.

The learning methods applied include existing learning techniques, namely edge weight tuning, where weights are associated with the edge types, and discriminative reranking of graph nodes, using features that describe the possible paths between a graph node and the initial "query nodes" (Minkov and Cohen, 2007).

In addition, we outline in this paper a novel method for learning *path-constrained* graph walks. While reranking allows use of high-level features that describe properties of the traversed paths, the suggested algorithm incorporates this information in the graph walk process. More specifically, we allow the probability flow in the graph to be conditioned on the history of the walk. We show that this method results in improved performance as it directs probability flow to meaningful paths. In addition, it leads

[*]Current address: Nokia Research Center Cambridge, Cambridge, MA 02142, USA.

to substantial gains in terms of runtime performance.

The graph representation and the set of learning techniques suggested are empirically evaluated on the task of *coordinate term* extraction[1] from small to moderately sized corpora, where we compare them against vector-based models, including a state-of-the-art syntactic distributional similarity method (Padó and Lapata, 2007). It is shown that the graph walk based approach gives preferable results for the smaller datasets (and comparable otherwise), where learning yields significant gains in accuracy.

There are several contributions of this paper. First, we represent dependency-parsed corpora within a general graph walk framework, and derive inter-word similarity measures using graph walks and learning techniques available in this framework. To our knowledge, the application of graph walks to parsed text in general, and to the extraction of coordinate terms in particular, is novel. Another main contribution of this paper is the path-constrained graph walk variant, which is a general learning technique for calculating the similarity between graph nodes in directed and labelled graphs.

Below we first outline our proposed scheme for representing a dependency-parsed text corpus as a graph, and provide some intuitions about the associated similarity metric (Section 2). We then give an overview of the graph-walk based similarity metric (Section 3), as well as the known edge weight tuning and reranking learning techniques (Section 4). We next present the proposed algorithm of path-constrained graph walks (Section 5). The paper proceeds with a review of related work (Section 6), a discussion of the coordinate term extraction task, empirical evaluation and our conclusions (Sections 7-9).

2 Representing a Corpus as a Graph

A typed dependency parse tree consists of directed links between words, where dependencies are labelled with the relevant grammatical relation (e.g., *nominal subject*, *indirect object* etc.). We suggest representing a text corpus as a connected graph of dependency structures, according to the scheme shown in Figure 1. The graph shown in the figure includes the dependency analysis of two sentences: "boys like playing with all kinds of cars", and "girls like playing with dolls". In the graph, each word mention is represented as a node, which includes the index of the sentence in which it appears, as well as its position within the sentence. Word mentions are marked as circles in the figure. The "type" of each word – henceforth a *term* node – is denoted by a square in the figure. Each word mention is linked to the corresponding term; for example, the nodes "like$_1$" and "like$_2$" represent distinct word mentions and both nodes are linked to the *term* "like". For every edge in the graph, we add another edge in the opposite direction (not shown in the figure); for example, an inverse edge exists from "like$_1$" to "girls$_1$" with an edge labelled as "nsubj-inv". The resulting graph is highly interconnected and cyclic.

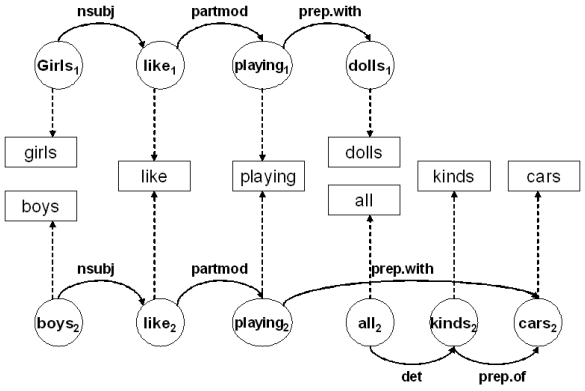

Figure 1: The suggested graph schema, demonstrated for a two-sentence corpus.

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

(1) girls $\overset{mention}{\longrightarrow}$ girls1 $\overset{nsubj}{\longrightarrow}$ like1 $\overset{as-term}{\longrightarrow}$ like $\overset{mention}{\longrightarrow}$ like2 $\overset{nsubj-inverse}{\longrightarrow}$ boys2 $\overset{as-term}{\longrightarrow}$ boys

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

Intuitively, in a graph representing a large corpus, terms that are more semantically related will be linked by a larger number of connecting paths. In addition, shorter connecting paths may be in general more meaningful. In the next section we show that

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

the graph walk paradigm addresses both of these requirements. Further, different edge types, as well as the paths traversed, are expected to have varying importance in different types of word similarity (for example, verbs and nouns are associated with different connectivity patterns). These issues are addressed using learning.

3 Graph Walks and Similarity Queries

This section provides a quick overview of the graph walk induced similarity measure. For details, the reader is referred to previous publications (e.g., (Toutanova et al., 2004; Minkov and Cohen, 2007)).

In summary, similarity between two nodes in the graph is defined by a weighted graph walk process, where an edge of type ℓ is assigned an edge weight, θ_ℓ, determined by its type.[2] The transition probability of reaching node y from node x over a single time step, $Pr(x \longrightarrow y)$, is defined as the weight of their connecting edge, θ_l, normalized by the total outgoing weight from x. Given these transition probabilities, and starting from an initial distribution V_q of interest (a *query*), we perform a graph walk for a finite number of steps K. Further, at each step of the walk, a proportion γ of the probability mass at every node is emitted. Thus, this model applies exponential decay on path length. The final probability distribution of this walk over the graph nodes, which we denote as R, is computed as follows: $R = \sum_{i=1}^{K} \gamma^i V_q \mathbf{M}^i$, where M is the transition matrix.[3] The answer to a query, V_q, is a list of nodes, ranked by the scores in the final distribution R. In this multi-step walk, nodes that are reached from the query nodes by many shorter paths will be assigned a higher score than nodes connected over fewer longer paths.

4 Learning

We consider a supervised setting, where we are given a dataset of example queries and labels over the graph nodes, indicating which nodes are relevant to which query. For completeness, we describe here two methods previously described by Minkov and Cohen (Minkov and Cohen, 2007): a hill-climbing method that tunes the graph weights; and a reranking method. We also specify the feature set to be used by the reranking method in the domain of parsed text.

4.1 Weight Tuning

There are several motivations for learning the graph weights Θ in this domain. First, some dependency relations – foremost, *subject* and *object* – are in general more salient than others (Lin, 1998; Padó and Lapata, 2007). In addition, dependency relations may have varying importance per different notions of word similarity (e.g., noun vs. verb similarity (Resnik and Diab, 2000)). Weight tuning allows the adaption of edge weights to each *task* (i.e., distribution of queries).

The weight tuning method implemented in this work is based on an error backpropagation hill climbing algorithm (Diligenti et al., 2005). The algorithm minimizes the following cost function:

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

where e_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}.[4] Specifically, p_z^{Opt} is set to 1 in case that the node z is relevant or 0 otherwise. The error is averaged over a set of example instantiations of size N. The cost function is minimized by gradient descent where the derivative of the error with respect to an edge weight θ_ℓ is derived by decomposing the walk into single time steps, and considering the contribution of each node traversed to the final node score.

4.2 Node Reranking

Reranking of the top candidates in a ranked list has been successfully applied to multiple NLP tasks (Collins, 2002; Collins and Koo, 2005). In essence, discriminative reranking allows the re-ordering of results obtained by methods that perform some form of local search, using features that encode higher level information.

[2] In this paper, we consider either uniform edge weights; or, learn the set of weights Θ from examples.

[3] We tune K empirically and set $\gamma = 0.5$, as in (Minkov and Cohen, 2007).

[4] For every example query, a handful of the retrieved nodes are considered, including both relevant and irrelevant nodes.

A number of features describing the set of paths from V_q can be conveniently computed in the process of executing the graph walk, and it has been shown that reranking using these features can improve results significantly. It has also been shown that reranking is complementary to weight tuning (Minkov and Cohen, 2007), in the sense that the two techniques can be usefully combined by tuning weights, and then reranking the results.

In the reranking approach, for every training example i ($1 \leq i \leq N$), the reranking algorithm is provided with the corresponding output ranked list of l_i nodes. Let z_{ij} be the output node ranked at rank j in l_i, and let $p_{z_{ij}}$ be the probability assigned to z_{ij} by the graph walk. Each output node z_{ij} is represented through m features, which are computed by pre-defined feature functions $f_1, \ldots, f_m$. The *ranking function* for node z_{ij} is defined as:

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

where $\bar{\alpha}$ is a vector of real-valued parameters. Given a new test example, the output of the model is the output node list reranked by $F(z_{ij}, \bar{\alpha})$. To learn the parameter weights $\bar{\alpha}$, we here applied a boosting method (Collins and Koo, 2005) (see also (Minkov et al., 2006)).

4.2.1 Features

We evaluate the following feature templates. *Edge label sequence* features indicate whether a particular sequence of edge labels ℓ_i occurred, in a particular order, within the set of paths leading to the target node z_{ij}. *Lexical unigram* feature indicate whether a word mention whose lexical value is t_k was traversed in the set of paths leading to z_{ij}. Finally, the *Source-count* feature indicates the number of different source query nodes that z_{ij} was reached from. The intuition behind this last feature is that nodes linked to multiple query nodes, where applicable, are more relevant. For example, for the query term "girl" in the graph depicted in Figure 1, the target node "boys" is described by the features (denoted as *feature-name.feature-value*): *sequence.nsubj.nsubj-inv* (where *mention* and *as-term* edges are omitted) , *lexical.'"like"* etc.

In this work, the features encoded are binary. However, features can be assugned numeric weights that corresponds to the probability of the indicator being true for any path between x and z_{ij} (Cohen and Minkov, 2006).

5 Path-Constrained Graph Walk

While node reranking allows the incorporation of high-level features that describe the traversed paths, it is desirable to incorporate such information earlier in the graph walk process. In this paper, we suggest a variant of a graph-walk, which is *constrained* by path information. Assume that preliminary knowledge is available that indicates the probability of reaching a relevant node after following a particular edge type sequence (path) from the query distribution V_q to some node x. Rather than fix the edge weights Θ, we can evaluate the weights of the outgoing edges from node x dynamically, given the history of the walk (the path) up to this node. This should result in gains in accuracy, as paths that lead mostly to irrelevant nodes can be eliminated in the graph walk process. In addition, scalability gains are expected, for the same reason.

We suggest a path-constrained graph walk algorithm, where path information is maintained in a compact path-tree structure constructed based on training examples. Each vertex in the path tree denotes a particular walk history. In applying the graph walk, the nodes traversed are represented as a set of node pairs, comprised of the graph node and the corresponding vertices in the path tree. The outgoing edge weights from each node pair will be estimated according to the respective vertex in the path tree. This approach needs to address two subtasks: learning of the path-tree; and updating of the graph walk paradigm to co-sample from the graph and the path tree. We next describe these two components in detail.

The Path-Tree

We construct a path-tree T using a training set of N example queries. Let a *path* p be a sequence of $k < K$ edge types (where K is the maximum number of graph walk steps). For each training example, we recover all of the connecting paths leading to the top M (correct and incorrect) nodes. We consider only acyclic paths. Let each path p be associated with its count, within the paths leading to the correct nodes, denoted as C_p^+. Similarly, the

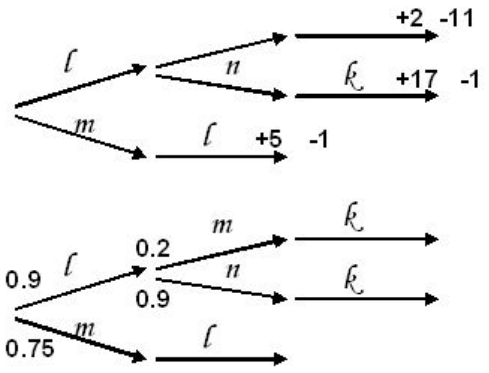

Figure 2: An example path-tree.

count within paths leading to the negatively labelled nodes is denoted C_p^-. The full set of paths observed is then represented as a tree.[5] The leaves of the tree are assigned a Laplace-smoothed probability: $Pr(p) = \frac{C_p^+ + 1}{C_p^+ + C_p^- + 2}$.

Given path probabilities, they are propagated backwards to all tree vertices, applying the MAX operator.[6] Consider the example given in Figure 2. The path-tree in the figure includes three paths (constructed from edge types k, l, m, n). The top part of the figure gives the paths' associated counts, and the bottom part of the figure gives the derived outgoing edge probabilities at each vertex. This path-tree specifies, for example, that given an edge of type l was traversed from the root, the probability of reaching a correct target node is 0.9 if an edge of type n is followed, whereas the respective probability if an edge of type m is followed is estimated at a lower 0.2.

A Concurrent Graph-walk

Given a generated path tree, we apply *path-constrained* graph walks that adhere both to the topology of the graph G, and to the path tree T. Walk histories of each node x visited in the walk are compactly represented as pairs $< t, x >$, where t denotes the relevant vertex in the path tree. For example, suppose that after one walk step, the maintained node-history pairs include $< T(l), x_1 >$ and $< T(m), x_2 >$. If x_3 is reached in the next walk step from both x_1 and x_2 over paths included in the path-tree, it will be represented by multiple node pairs, e.g., $< T(l \rightarrow n), x_3 >$ and $< T(m \rightarrow l, x_3 >$. A pseudo-code for a path-constrained graph walk is given in Figure 3. It is straight-forward to discard paths in T that are associated with a lower probability than some threshold. A threshold of 0.5, for example, implies that only paths that led to a majority of positively labelled nodes in the training set are followed.

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

Figure 3: Pseudo-code for path-constrained graph walk

6 Related Work

Graph walks over typed graphs have been applied to derive semantic similarity for NLP problems using WordNet as a primary information source. For instance, Hughes and Ramage (2007) constructed a graph which represented various types of word relations from WordNet, and compared random-walk similarity to similarity assessments from human-subject trials. Random-walk similarity has also been used for lexical smoothing for prepositional word attachment (Toutanova et al., 2004) and query expansion (Collins-Thompson and Callan, 2005). In contrast to these works, our graph representation describes parsed text and has not been (consciously) engineered for a particular task. Instead, we include learning techniques to optimize the graph-walk based similarity measure. The learning methods described in this paper can be readily applied to

[5] The conversion to a tree is straight-forward, where identical path prefixes are merged.

[6] Another possibility is to average the downstream cumulative counts at each vertex. The MAX operation gave better results in our experiments.

other directed and labelled entity-relation graphs.[7]

The graph representation described in this paper is perhaps most related to syntax-based vector space models, which derive a notion of semantic similarity from statistics associated with a parsed corpus (Grefenstette, 1994; Lin, 1998; Padó and Lapata, 2007). In most cases, these models construct vectors to represent each word w_i, where each element in the vector for w_i corresponds to particular "context" c, and represents a count or an indication of whether w_i occurred in context c. A "context" can refer to simple co-occurrence with another word w_j, to a particular syntactic relation to another word (e.g., a relation of "direct object" to w_j), etc. Given these word vectors, inter-word similarity is evaluated using some appropriate similarity measure for the vector space, such as cosine vector similarity, or *Lin's similarity* (Lin, 1998).

Recently, Padó and Lapata (Padó and Lapata, 2007) have suggested an extended syntactic vector space model called *dependency vectors*, in which rather than simple counts, the components of a word vector of contexts consist of *weighted scores*, which combine both co-occurrence frequency and the importance of a context, based on properties of the connecting dependency paths. They considered two different weighting schemes: a *length* weighting scheme, assigning lower weight to longer connecting paths; and an *obliqueness* weighting hierarchy (Keenan and Comrie, 1977), assigning higher weight to paths that include grammatically salient relations. In an evaluation of word pair similarity based on statistics from a corpus of about 100 million words, they show improvements over several previous vector space models. Below we will compare our framework to that of Padó and Lapata. One important difference is that while Padó and Lapata make manual choices (regarding the set of paths considered and the weighting scheme), we apply learning to adjust the analogous parameters.

7 Extraction of Coordinate Terms

We evaluate the text representation schema and the proposed set of graph-based similarity measures on the task of *coordinate term* extraction. In particular, we evaluate the extraction of named entities, including *city names* and *person names* from newswire data, using word similarity measures. Coordinate terms reflect a particular type of word similarity (relatedness), and are therefore an appropriate test case for our framework. While coordinate term extraction is often addressed by a rule-based (templates) approach (Hearst, 1992), this approach was designed for very large corpora such as the Web, where the availability of many redundant documents allows use of high-precision and low-recall rules. In this paper we focus on relatively small corpora. Small limited text collections may correspond to documents residing on a personal desktop, email collections, discussion groups and other specialized sets of documents.

The task defined in the experiments is to retrieve a ranked list of city or person names given a small set of seeds. This task is implemented in the graph as a query, where we let the query distribution V_q be uniform over the given seeds (and zero elsewhere). Ideally, the resulting ranked list will be populated with many additional city, or person, names.

We compare graph walks to *dependency vectors* (DV) (Padó and Lapata, 2007),[8] as well as to a vector-based bag-of-words co-occurrence model. DV is a state-of-the-art syntactic vector-based model (see Section 6). The co-occurrence model represents a more traditional approach, where text is processed as a stream rather than syntactic structures. In applying the vector-space based methods, we compute a similarity score between *every* candidate from the corpus and each of the query terms, and then average these scores (as the query distributions are uniform) to construct a ranked list. For efficiency, in the vector-based models we limit the considered set of candidates to named entities. Similarly, the graph walk results are filtered to include named entities.[9]

Corpora. As the experimental corpora, we use the training set portion of the MUC-6 dataset (MUC, 1995) as well as articles from the Associated Press (AP) extracted from the AQUAINT corpus (Bilotti

[7] We refer the reader to the TextGraph workshop proceedings, http://textgraphs.org.

[8] We used the code from http://www.coli.uni-saarland.de/ pado/dv.html, and converted the underlying syntactic patterns to the Stanford dependency parser conventions.

[9] In general, graph walk results can be filtered by various word properties, e.g., capitalization pattern, or part-of-speech.

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

Table 1: Corpus statistics

et al., 2007), all parsed using the Stanford dependency parser (de Marneffe et al., 2006).[10] The MUC corpus provides true named entity tags, while the AQUAINT corpus includes automatically generated, noisy, named entity tags. Statistics on the experimental corpora and their corresponding graph representation are detailed in Table 1. As shown, the MUC corpus contains about 140 thousand words, whereas the MUC+AP experimental corpus is substantially larger, containing about 2.5 million words.

We generated 10 queries, each comprised of 4 city names selected randomly according to the distribution of city name mentions in MUC-6. Similarly, we generated a set of 10 queries that include 4 person names selected randomly from the MUC corpus. (The MUC corpus was appended to AP, so that the same query sets are applicable in both cases.) For each task, we use 5 queries for training and tuning and the remaining queries for testing.

8 Experimental Results

Experimental setup. We evaluated cross-validation performance over the training queries in terms of mean average precision for varying walk lengths K. We found that beyond $K = 6$ improvements were small (and in fact deteriorated for $K = 9$). We therefore set $K = 6$. Weight tuning was trained using the training queries and two dozens of target nodes overall. In reranking, we set a feature count cutoff of 3, in order to avoid over-fitting. Reranking was applied to the top 200 ranked nodes output by the graph walk using the tuned edge weights. Finally, path-trees were constructed using the top 20 correct nodes and 20 incorrect nodes retrieved by the uniformly weighted graph walk. In the experiments, we apply a threshold of 0.5 to the path constrained graph walk method.

We note that for learning, true labels were used for the fully annotated MUC corpus (we hand labelled all of the named entities of type location in the corpus as to whether they were city names). However, noisy negative examples were considered for the larger automatically annotated AP corpus. (Specifically, for cities, we only considered city names included in the MUC corpus as correct answers.)

A co-occurrence vector-space model was applied using a window of two tokens to the right and to the left of the focus word. Inter-word similarity was evaluated in this model using cosine similarity, where the underlying co-occurrence counts were normalized by log-likelihood ratio (Padó and Lapata, 2007). The parameters of the DV method were set based on a cross validation evaluation (using the MUC+AP corpus). The *medium* set of dependency paths and the *oblique* edge weighting scheme were found to perform best. We experimented with cosine as well as Lin similarity measure in combination with the dependency vectors representation. Finally, given the large number of candidates in the MUC+AP corpus (Table 1), we show the results of applying the considered vector-space models to the top, high-quality, entities retrieved with reranking for this corpus.[11]

Test set results. Figure 4 gives results for the city name (top) and the person name (bottom) extraction tasks. The left part of the figure shows results using the MUC corpus, and its right part – using the MUC+AP corpus. The curves show precision as a function of rank in the ranked list, up to rank 100. (For this evaluation, we hand-labeled all the top-ranked results as to whether they are city names or person names.) Included in the figure are the curves of the graph-walk method with uniform weights (G:Uw), learned weights (G:Lw), graph-walk with reranking (Rerank) and a path-constrained graph-walk (PCW). Also given are the results of the co-occurrence model (CO), and the syntactic vector-space DV model, using the Lin similarity measure (DV:Lin). Performance of the DV model using cosine similarity was found comparable or inferior to using the Lin measure, and is omitted from the figure for clarity.

Several trends can be observed from the results. With respect to the graph walk methods, the graph walk using the learned edge weights consistently outperforms the graph walk with uniform weights. Reranking and the path-constrained graph walk,

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

[11] We process the union of the top 200 results per each query.

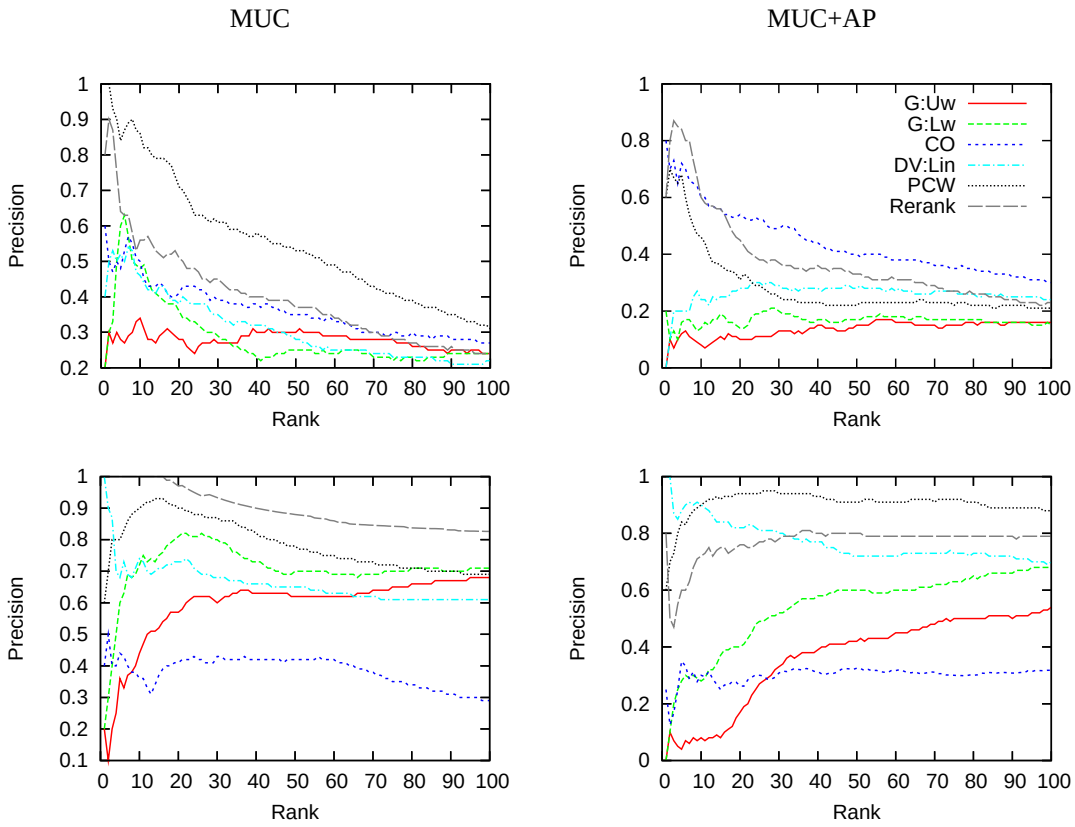

Figure 4: Test results: Precision at the top 100 ranks, for the city name extraction task (top) and person name extraction task (bottom).

however, yield superior results. Both of these learning methods utilize a richer set of features than the graph walk and weight tuning, which can consider only local information. In particular, while the graph walk paradigm assigns lower importance to longer connecting paths (as described in Section 3), reranking and the path-constrained walker allow to discard short yet irrelevant paths, and by that eliminate noise at the top ranks of the retrieved list. In general, the results show that edge sequences carry additional meaning compared with the individual edge label segments traversed.

Out of the vector-based models, the co-occurrence model is preferable for the city name extraction task, and the syntactic dependency vectors model gives substantially better performance for person name extraction. We conjecture that city name mentions are less structured in the underlying text. In addition, the syntactic weighting scheme of the DV model is probably not optimal for the case of city names. For example, a *conjunction* relation was found highly indicative for city names (see below). However, this relation is not emphasized by the DV weighting schema. As expected, the performance of the vector-based models improves for larger corpora (Terra and Clarke, 2003). These models demonstrate good performance for the larger MUC+AP corpus, but only mediocre performance for the smaller MUC corpus.

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

Models. Following is a short description of the models learned by the different methods and tasks. Weight tuning assigned high weights to edge types such as *conj-and*, *prep-in* and *prep-from*, *nn*, *appos* and *amod* for the city extraction task. For per-

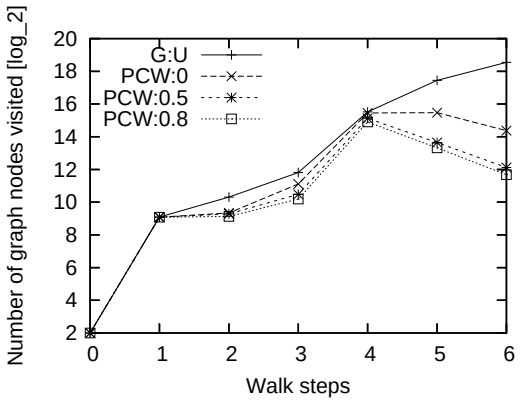

Figure 5: The graph walk exponential spread is bounded by the path constrained walk.

son extraction, prominent edge types included *subj*, *obj*, *poss* and *nn*. (The latter preferences are similar to the linguistically motivated weights of DV.) High weight features assigned by reranking for city name extraction included, for example, lexical features such as "based" and "downtown", and edge bigrams such as "prep-in-Inverse→conj-and" or "nn-Inverse→nn". Positive highly predictive paths in the constructed path tree included many symmetric paths, such as ...→conj_andInverse...→.conj_and..., ...→prep_inInverse...→.prep_in..., for the city name extraction task.

Scalability. Figure 5 shows the number of graph nodes maintained in each step of the graph walk (logarithm scale) for a typical city extraction query and the MUC+AP corpus. As shown by the solid line, the number of graph nodes visited using the weighted graph walk paradigm grows exponentially with the length of the walk. Applying a path-constrained walk with a threshold of 0 (PCW:0) reduces the maximal number of nodes expanded (as paths not observed in the training set are discarded). As shown, increasing the threshold leads to significant gains in scalability. Overall, query processing time averaged at a few minutes, using a commodity PC.

9 Conclusion and Future Directions

In this paper we make several contributions. First, we have explored a novel but natural representation for a corpus of dependency-parsed text, as a labelled directed graph. We have evaluated the task of coordinate term extraction using this representation, and shown that this task can be performed using similarity queries in a general-purpose graph-walk based query language. Further, we have successfully applied learning techniques that tune weights assigned to different dependency relations, and re-score candidates using features derived from the graph walk.

Another orthogonal contribution of this paper is a path-constrained graph walk variant, where the graph walk is guided by high level knowledge about meaningful paths, learned from training examples. This method was shown to yield improved performance for the suggested graph representation, and improved scalability compared with the local graph walk. The method is general, and can be readily applied in similar settings.

Empirical evaluation of the coordinate term extraction task shows that the graph-based framework performs better than vector-space models for the smaller corpus, and comparably otherwise. Overall, we find that the suggested model is suitable for deep (syntactic) processing of small specialized corpora. In preliminary experiments where we evaluated this framework on the task of extracting general word synonyms, using a relatively large corpus of 15 million words, we found the graph-walk performance to be better than DV using cosine similarity measures, but second to DV using Lin's similarity measure. While this set of results is incomplete, we find that it is consistent with the results reported in this paper.

The framework presented can be enhanced in several ways. For instance, WordNet edges and morphology relations can be readily encoded in the graph. We believe that this framework can be applied for the extraction of more specialized notions of word relatedness, as in relation extraction (Bunescu and Mooney, 2005). The path-constrained graph walk method proposed may be enhanced by learning edge probabilities, using a rich set of features. We are also interested in exploring a possible relation between the path-constrained walk approach and reinforcement learning.

Acknowledgments

The authors wish to thank the anonymous reviewers and Hanghang Tong for useful advice. This material is based upon work supported by Yahoo! Research.

References

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

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

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

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

Kevyn Collins-Thompson and Jamie Callan. 2005. Query expansion using random walk models. In *CIKM*.

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

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

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

Gregory Grefenstette. 1994. *Explorations in Automatic Thesaurus Discovery*. Kluwer Academic Publishers, Dordrecht.

Marti Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *COLING*.

Thad Hughes and Daniel Ramage. 2007. Lexical semantic relatedness with random graph walks. In *EMNLP*.

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

Dekang Lin. 1998. Automatic retrieval and clustering of similar words. In *COLING-ACL*.

Einat Minkov and William W. Cohen. 2007. Learning to rank typed graph walks: Local and global approaches. In *WebKDD/KDD-SNA workshop*.

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

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

Sebastian Padó and Mirella Lapata. 2007. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2).

Philip Resnik and Mona Diab. 2000. Measuring verb similarity. In *CogSci*.

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. 2005. Learning syntactic patterns for automatic hypernym discovery. In *NIPS*.

Egidio Terra and C. L. A. Clarke. 2003. Frequency estimates for statistical word similarity measures. In *NAACL*.

Kristina Toutanova, Christopher D. Manning, and Andrew Y. Ng. 2004. Learning random walk models for inducing word dependency distributions. In *ICML*.