Learning to Walk Structured Text Networks

CMU-LTI-08-002

Einat Minkov and William W. Cohen

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

March 5, 2008

Abstract

We propose representing a text corpus as a labeled directed graph, where nodes represent words and weighted edges represent the syntactic relations between them, as derived by dependency parsing. Given this graph, we adopt a graph-based similarity measure based on random walks to derive a similarity measure between words, and also use supervised learning to improve the derived similarity measure for a particular task. Empirical evaluation of the approach on the task of coordinate term extraction shows that the suggested framework improves on a state-of-the-art distributional similarity measure.

Contents

1 Introduction

In recent years, researchers have used syntactic structure in the form of *dependency parses* for a number of NLP tasks, including relation extraction (e.g., (Bunescu & Mooney, 2005)), question answering (e.g., (Cui et al., 2005)), and machine translation (e.g., (Quirk et al., 2005)). One advantage of dependency parsing is that it readily provides semantically useful predicate-argument structure.

In this paper, we propose representing a text corpus as a labeled directed graph of dependency parse trees. In the suggested graph scheme, nodes denote words or word occurrences, and edges represent the dependency relations between word occurrences, or links between a word occurrence and the "generic version" of a word. Given such a labeled directed graph, previously-developed techniques based on graph walks (Minkov et al., 2006) can be used to impose a measure of similarity between the graph nodes, and machine learning techniques can be used to optimize these similarity metrics for a specific family of tasks (Minkov et al., 2006; Minkov & Cohen, 2007). Here we evaluate these graph-based similarity metrics on a coordinate term extraction task, and show them to be comparable to a strong baseline method, which also uses dependency-parse information—a state-of-the-art distributional similarity technique due to Padó and Lapata (Padó & Lapata, 2007).

Below we will first outline our proposed scheme for representing a dependency-parsed text corpora as a graph, and provide some intuitions about the associated similarity metric. We then present the similarity metric in detail, and describe two learning techniques: one that tunes a set of weights associated with each edge type, and one that discriminatively reranks graph nodes, using features that describe the possible paths between a graph node and the initial "query nodes". We next present the task of coordinate term extraction, and evaluate performance of these methods on three small to moderately sized corpora. Relative to distributional similarity metrics, the graph walk has an computational advantage: rather than only supporting *scoring* of candidate pairs, the graph supports *retrieval* of similar objects (i.e., finding all objects y that are similar to a given query object x). We also show that, for this task, the graph walk gives much better results than a state-of-the-art distributional similarity method (Padó & Lapata, 2007).

2 Representing a corpus as a graph

A dependency parse tree consists of directed links between words. A *typed* dependency parse additionally labels dependencies with the relevant grammatical relation, such as *subject*, *indirect object* etc. For example, consider the sentence parse in Figure 1. In this example sentence, annotated according to the Stanford parser conventions (de Marneffe et al., 2006), the subject, "boy", is linked to the verb, "like", with the typed link "nsubj" (nominal subject).

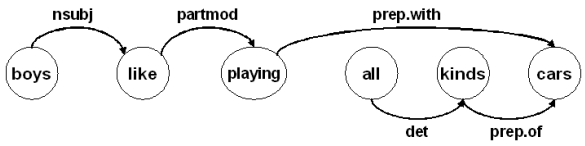

Figure 1: A dependency parse for the sentence "Boys like playing with all kinds of cars."

Rather than process dependency relations on a per-sentence basis (e.g., (Snow et al., 2005)), we suggest representing a text corpus as a connected graph of dependency structures. In particular, we consider the scheme shown in Figure 2. 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". To ensure that graph retains

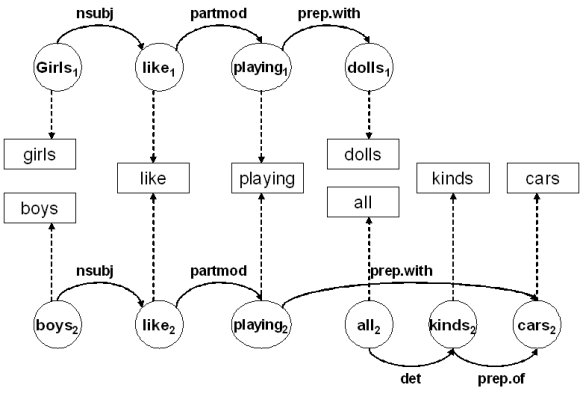

Figure 2: The suggested graph schema, demonstrated for a two-sentence corpus.

the original structures of each dependency tree, each word mention is represented as a special object, which includes the index of the sentence in which it appears, and its position in that sentence (omitted for clarity in the figure) . Nodes denoting word mentions are marked as circles in the figure. The "generic version" 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 edge exists from "like$_1$" to "girls$_1$" with an edge labelled as "nsubj-inv". The resulting graph is highly interconnected and cyclic.

A larger corpus of parsed sentences will be represented just as a larger graph. We will apply graph walks to derive an extended measure of similarity, or relatedness, between word *terms* (as defined above). For example, suppose we are to apply graph walks in the graph depicted in Figure 2 in order to find terms similar to the term "girls". Starting from the term "girls", we will reach the semantically related term "boys" via the following two paths:

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

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

Intuitively, in this graph, terms that are more closely related semantically will be linked by a larger number of connecting paths in a corpus, and the connecting paths will often be shorter.

3 Graph walks and similarity queries

Below, we will denote graph nodes by letters such as x, y, or z, and an edge from x to y with label ℓ as $x \stackrel{\ell}{\longrightarrow} y$. Every node x is associated with a type, denoted $\tau(x)$. For example, the schema in Figure 2 includes node types of *word mention* and *term*.

Similarity between two nodes in the graph is defined by a graph walk process, controlled by a set of parameters Θ, where an edge of type ℓ is assigned an edge weight determined by its type, θ_ℓ. In this paper, we consider either uniform edge weights; and, a setting where the set of weights Θ is learned from examples (Section 4).

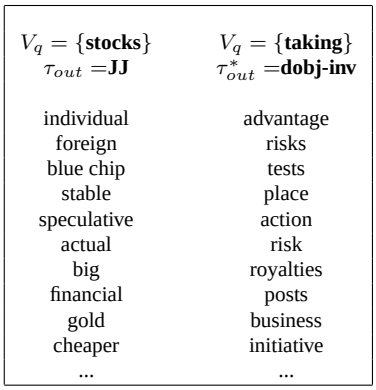

Figure 3: Two example queries and the corresponding top ranked results, where walk length $k = 3$ and Θ are uniform, using a graph representing the MUC-6 corpus

The graph walk process is defined as follows. Let L_{xy} denote the set of edge types of the outgoing edges from x to y. The probability of reaching node y from node x over a single time step is defined as:

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

where $ch(x)$ denotes the set of immediate children of x. That is, the outgoing weights from x are normalized to form a probability distribution.

Define $\mathbf{M}_{xy} \equiv Pr(x \longrightarrow y)$. If we associate nodes with integers, and make $\mathbf{M}$ a matrix indexed by nodes, then the probability of reaching y from x with a path of length i is the (x, y)- element of $\mathbf{M}^i$.

Finally consider the vector $\mathbf{R} = \sum_{i=1}^{k} \gamma^i V_x \mathbf{M}^i$, where V_x is a unit vector representing a probability distribution over nodes. If V_x puts weight 1 on x and weight 0 on all other nodes, then the y-th component of $\mathbf{R}$ is proportional to the probability of reaching y in an i-step random walk from x, where the walk length i is chosen with probability proportional to γ^i. (Due to $0 < \gamma < 1$, an exponential decay applies over path length.) Hence $s = \mathbf{R}_y$ is a plausible measure of the similarity of y to x.

Given V_x (any initial distribution) the vector $\mathbf{R}$ can be computed quite straightforwardly by sparse matrix multiplication for small values of k. This is the approach taken here (with γ and k being fixed by the user). In our experiments, this operation requires only a few minutes (at most 10 minutes for the largest corpus) on a commodity PC. However, the stationary distribution associated with $\mathbf{R}$ (as $k \to \infty$) is well-studied—it is known variously as "personalized PageRank" or "random walk with restart—and techniques for approximating it more efficiently are known (e.g., (Tong et al., 2006)) .

More generally, we define a *similarity query* to consist of an initial distribution V_q over nodes, plus a desired output type τ_{out}. The answer to a query is a list of nodes z filtered by type τ_{out}, and ranked by the node's score in the final distribution $\mathbf{R}$.

In addition to the node types of *word mention* and *term*, one can extend the graph with additional node types, allowing more interesting filters and queries. For example, Figure 3 shows the queries for finding the nouns most related to the term "stocks", and for finding the "direct-object terms"[1] that are most related to the term "taking".

[1]I.e., terms with a the appropriate incoming dependency link.

4 Learning

We consider a supervised setting, given a dataset of example queries, and labels over the graph nodes, indicating whether they are considered to be relevant answers, per query. We use here two methods previously described by Minkov and Cohen (Minkov & Cohen, 2007): a hill-climbing method that tunes the graph weights; and a reranking method. For completeness, we include a short overview of the two approaches in this section. We also describe the feature set used.

4.1 Weight Tuning

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

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:

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

where e_z is the error for 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 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, 2002b; Collins & 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.

A number of features describing the set of paths from V_q can be conveniently computed in the process of computing similarity scores, and it has been shown that reranking using these features can improve results significantly (Minkov et al., 2006). It has also been shown that reranking is complementary to weight tuning (Minkov & 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 the voted perceptron algorithm (Collins, 2002a).

4.3 Features

We evaluate the following feature templates. *Edge label bigram* features indicate whether a particular sequence of edge labels ℓ_i and ℓ_j occurred, in this order, within the set of paths leading to the target node z_{ij}. *Lexicalized edge label bigram* features indicate whether a term t_k was traversed between two consecutive edges ℓ_i and ℓ_j (i.e., a trigram ℓ_i, t_k, ℓ_j). *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}.

In this work, each feature is given a numeric weight that corresponds to the probability of the indicator being true for any path between x and z_{ij}, as in earlier work by Cohen and Minkov (Cohen & Minkov, 2006)).

5 Related work

This work is not the first to apply graph walks to obtain a notion of semantic similarity for NLP problems. For instance, Hughes and Ramage (Hughes & 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 & Callan, 2005). Alternative methods have even been suggested to represent a text corpus as a graph—for example, graphs have been evaluated for automatic text summarization (Erkan & Radev, 2004), where nodes are sentences and links are drawn between similar sentences. To our knowledge this paper is novel in representing a corpus represented as a graph that includes syntactic information (in particular, dependency-parsed text), and is novel in exploring the use of random-walk similarity on such a graph.

We note that graphs derived from individual sentences been have widely used–e.g., Snow et al (Snow et al., 2005) used dependency paths in order to extract hyponyms from a corpus of parsed text, but separate sentences were not connected.

In contrast to most earlier researchers, our graph representation has not been (consciously) engineered for any particular task, although we do include learning techniques to adapt it to the data. However, engineering the structure graph (e.g., by adding WordNet edges, linking together morphologically similar words, etc) is straightforward to do in this framework, and we may explore this in future work.

The framework 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 & 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 & 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 consist of *weighted scores*, which combine both co-occurrence frequency and the importance of a context.

Context importance is based on properties of the context. They considered two different weighting schemes: a *length* weighting scheme, assigning lower weight to longer connecting paths (computed as inverse of path length); and an *obliqueness* weighting hierarchy (Keenan & Comrie, 1977), assigning higher weight to paths that include gramatically salient relations. Another parameter controlling the computed scores in their framework limits the set of considered paths to a manually designed set, representing various types of linguistic interesting phenomena. In an evaluation of word pair similarity based on statistics from a corpus of about 100 million words, they show improvements over several previous vector space models.

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.

6 Extraction of coordinate terms

We evaluate the suggested framework on the task of *coordinate terms* extraction. Coordinate terms are terms that share the same (immediate) parent in a taxonomy: for example, "tomato", "corn" and "pepper" are coordinate terms, as they are all instances of "vegetable".

In this paper, we evaluate the extraction of *city names* from newswire data. The task is to retrieve a ranked list of city names given a small set of seeds (in the experiments, four seeds). This task is implemented as a similarity query by retrieving nodes with type $\tau =$"named entity", and letting the query distribution V_q be uniform over the four seeds (and zero elsewhere). Ideally, the resulting ranked list will be populated with many additional city names, due to semantic similarity.

We are interested in coordinate term extraction for two reasons. First, this task is intrinsically important, as it is required for automatic knowledge base construction—e.g., (Etzioni et al., 2005). Second, it is a clearly-defined task, for which we can easily identify correct answers in a corpus. This will allow us to evaluate performance in terms of both precision and recall.

6.1 Experimental Setup

In order to obtain a labelled dataset, we parsed the training set portion of the MUC-6 dataset (MUC6, 1995), using the Stanford dependency parser.[2] The MUC-6 collection includes 317 Wall Street Journal articles. In addition to the text, it provides gold standard annotations of named entities (NEs) and their types—e.g., "New York" is annotated as "Location".[3] For the experiments, we hand-labeled all location NEs as to whether they were city names. Overall, we identified 185 unique city names in the corpus.[4] We then generated 10 queries comprised of cities' names. Each query includes 4 city names, selected randomly according to the distribution of city name mentions in the MUC-6 corpus. Given every query, we extract a ranked list of named entities of type $\tau =$"named entity"[5] in the corpus (query terms are of course discarded from this list). We use 5 labelled queries for training and tuning purposes, and reserve the remaining 5 queries for testing. In addition to the small MUC-6 corpus we constructed two incrementally larger corpora, by adding to the MUC-6 corpus parsed articles of the Associated press, extracted from the AQUAINT corpus (Bilotti et al., 2007). The AQUAINT corpus has been annotated with named entities automatically, so it includes noisy tags; nevertheless, we processed queries in the these graphs in the same

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

[3] We process named entities as if they were single words.

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

[5] Annotated as either Location, Person or Organization.

Corpus	*words*	*nodes*	*edges*	*unique NEs*
MUC	140K	82K	244K	3K
MUC+AP1	715K	326K	1,077K	12K
MUC+AP2	2,440K	1,030K	3,550K	36K

Table 1: Corpus statistics

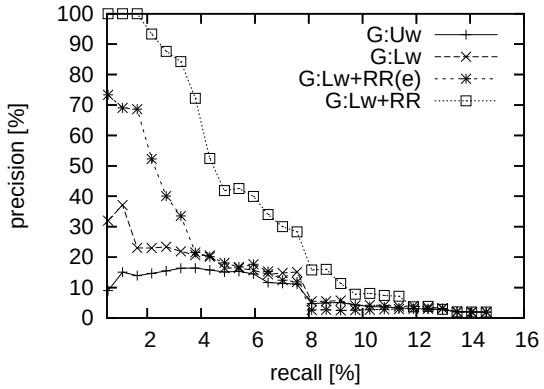

Figure 4: Cross validation evaluation, using the MUC-6 corpus: contribution of learned weights and feature sets (top) and final results for varying walk length (bottom)

manner described above. Statistics of these corpora and their corresponding graph representation are given in Table 1.

6.2 Graph Walk Tuning

We performed a 5-fold cross validation evaluation using the MUC-6 corpus and the train set queries, to tune the parameters involved in the graph walk and learning. Figure 4 gives a precision-recall curve for applying graph walks with walk length $k = 6$ and with uniform weights (G:Uw); with learned weights (G:Lw); for re-ranking the top 200 nodes of the weighted walk, using the edge type bigram features (G:Lw+RR(e)); and reranking using all of the features described in Section 4.3, including lexical information (G:Lw+RR). As shown in these results, weight tuning improves performance.[6] Further, reranking gives a significant boost to precision, at all recall levels. As shown, lexical information is informative for this task. In the rest of the experiments we apply the best graph walk configuration from these cross-validation experiments (i.e., tuned weights and the full set of features).

One of the distinctive aspects of this framework is the fact that the graph-walk can span multiple sentences. This is a potential advantage, as intra-sentence co-occurrence of coordinate terms is rare (Snow et al., 2005). To measure the potential impact of this, we measured the maximum possible recall obtainable by processing dependency paths of a collection of *individual* sentences. The top of Table 2 shows the maximum recall obtainable using co-occurrence data of increasing distance d in a disconnected graph of individual sentences (i.e., d is the maximal number of dependency edges traversed within a sentence), and the bottom of Table 2 shows the maximum recall reached by graph walks for increasing length k, for the MUC and MUC+AP2 corpora. As can be deduced from the graph schema (Figure 2), a graph walk of $k = 6$

[6] Weight tuning (trained on the MUC+AP1 corpus and two dozens of target nodes) assigned higher weights to edge types such as *conj-and*, *prep-of*, *nn*, *dobj* and *appos*.

Intra-sentence	*d=1*	*d=2*	*d=3*	*d=4*	*d=5*
MUC	0.9	2.8	3.4	3.8	4.0
MUC+AP2	1.7	5.7	7.0	7.5	7.7
Graph walk	*k=4*	*k=5*	*k=6*	*k=7*	*k=8*
MUC	2.8	3.4	8.6	17.9	27.6
MUC+AP2	5.7	7.0	19.6	33.3	-

Table 2: avg. Max. recall

steps or more is required for a path to cross two sentences—which coincides in a jump in maximum recall in both corpora.

To set the parameter k, we evaluated cross-validation performance in terms of mean average precision on varying walk lengths. The performance of a graph walk where $k = 6$ steps gave substantial improvements over shorter walks, and beyond that further improvements were small (and in fact deteriorated for $k = 9$). We therefore set the graph walk length to $k = 6$.

6.3 A Comparative Evaluation

We conducted an empirical evaluation of the graph walks against *dependency vectors* (DV)[7] (Padó & Lapata, 2007), a weighted semantic vector space model (see Section 4). In applying the DV method, we compute a similarity score between *every* named entity 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.

Evaluating similarity for all candidate entities can be expensive for a large corpus. One advantage of the graph-walk method is that it supports "retrieval" of similar objects—i.e., given a set of seeds, one can directly compute a (sparse) vector **R** that lists all other nodes similar to the seeds. Distribution similarity methods do not directly support this operation.

We set the parameters of the DV method based on a 5-fold cross validation evaluation over the training queries and the MUC and MUC+AP1 corpora. We used the medium set of dependency paths, an edge weighting scheme, and a cosine similarity measure.

Test set results. The top graphs of Figure 5 compare the DV method and the graph-walk method (with learned weights only, and with reranking) on the test-set queries from the MUC and MUC+AP1 corpora. Here we evaluate precision as a function of rank in the ranked list. (We hand-labeled all the top-ranked results as to whether they are city names; hence we show accurate measures of precision, considering also city names that are not contained in MUC, as relevant for the larger corpus.) The performance of graph walks with re-ranking is very encouraging. For all queries, the top ranks were densely populated with city names. Such high quality results could perhaps support a process of bootstrapping, where new queries (seeds) are constructed based on the returned lists, in order to gradually expand the the list of extracted items.

For this task, as we know all the city names that occur in the MUC corpus, we can measure recall as well as precision with respect to this known set. The graphs at the bottom Figure 5 give these results. It is clear that both instances of the graph-walk are performing substantially better than DV for the large corpus as well. Precision of both the graph walk and the DV method appears worse for the larger corpus; however, this is likely an artifact of the details of experimental procedure, as we consider a city name as "current" only if it is included in the MUC-6 corpus—i.e., precision is underestimated for the larger corpus. Another

[7] We used the code from http://www.coli.uni-saarland.de/ pado/dv.html, converting the minimal and medium context files to the Stanford dependency parser conventions.

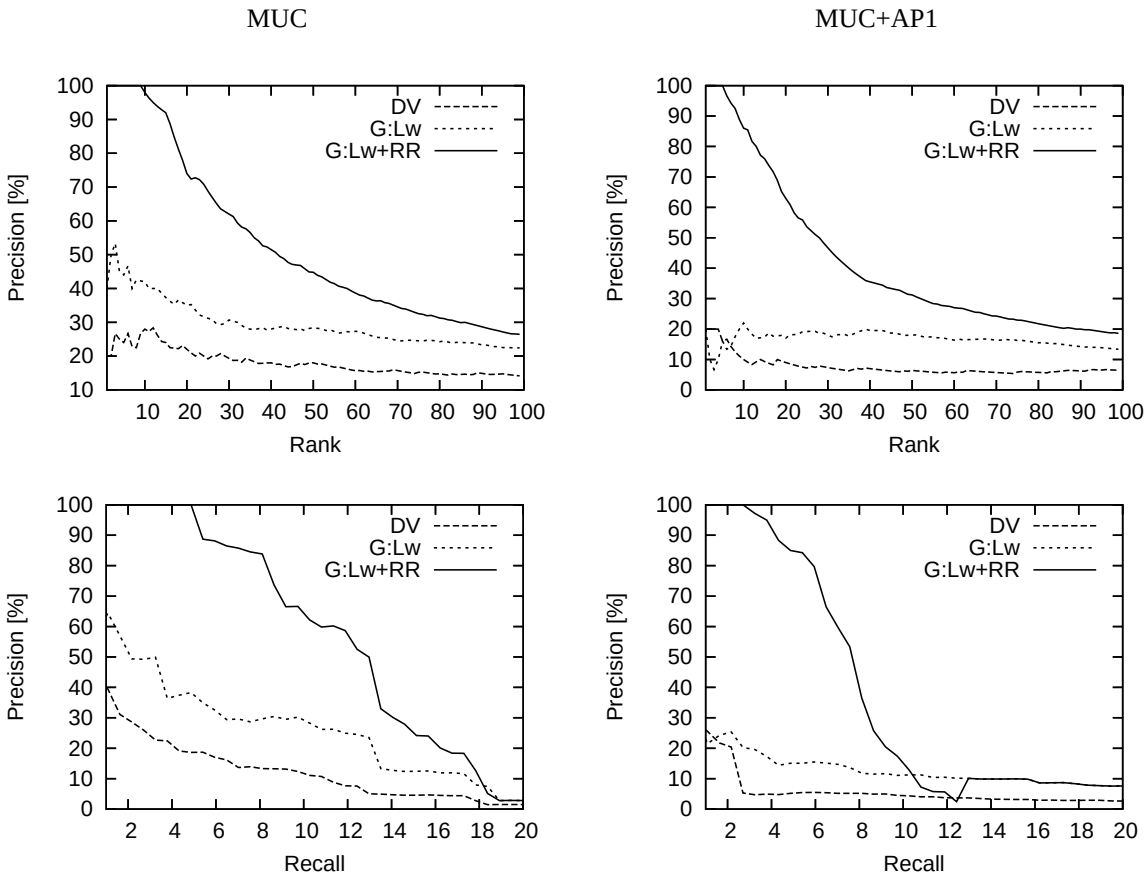

Figure 5: Test results: Precision vs. recall (top) and precision at different ranks (bottom)

possible reasons for the precision decrease in the larger corpus is that the candidate entities are noisy in MUC+AP1; also, city names appear to be more concentrated in the MUC-6 corpus.

Scalability questions. While the *dependency vectors* model and its variants can be applied on very large corpora, they require a small set of candidate words to be selected for scoring, and we could not run these methods to score all the entities from the larger MUC+AP2 corpus. As an alternative, we used the DV method to rerank the top items retrieved by the weighted graph walk.[8]

Figure 6 shows the result of this experiment (see the G:Lw+DV curve). Here we hand-labeled all the top-ranked results as to whether they are city names; hence we show accurate measures of precision versus rank. Given this filtered candidate set, DV improves the performance relative to the weighted graph walk in the top ranks, but not at lower ranks. Its performance is clearly inferior to the graph walk with reranking (which uses discriminative reranking to reorder nodes based on the walk-related features).

[8]The set we used is a union of the top 200 results per each query.

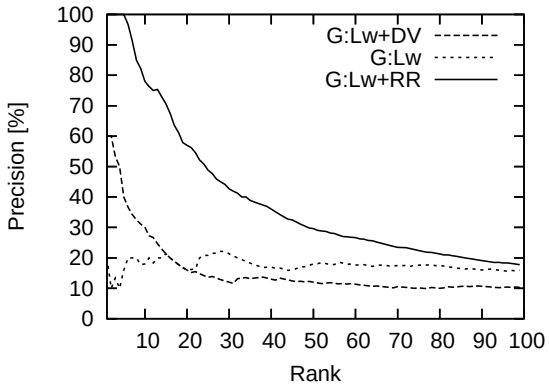

Figure 6: Test results for the MUC+AP2 corpus, where DV is used for reranking the top candidates retrieved by GW

7 Conclusion and future directions

In this paper, we have explored a novel but natural represention for a corpus of dependency-parsed parsed text, as a labeled directed graph. We have evaluated the task of coordinate term extraction using this representation, and shown that coordinate term extraction can be performed using similarity queries in a general-purpose graph-walk based query language—a query language used for a variety of personal-information tasks in the past (Minkov et al., 2006; Minkov & Cohen, 2007). 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.

In the empirical evaluation on the coordinate term extraction task, we compared this framework to the state-of-the-art syntactic vector space model, both in terms of accuracy and recall. We found that after having learned the graph weights, the graph walks give favorable performance. Discriminative reranking using features derived directly from the graph-walk further boosts performance, giving almost perfect accuracy at the top ranks. Another difference between the compared approaches is that unlike vector based models that only *score* pairs of terms provided by the user, the graph walk method *retrieves* similar items items.

The framework presented is general, in that additional information, such as WordNet edges, can be readily encoded in the graph. Other directions of future work include further exploration of more specialized features describing the set of paths, and scaling to larger corpora. Finally, we believe that this framework can be applied for the extraction of more specialized notions of word relatedness, as in relation extraction (Bunescu & Mooney, 2005).

References

Bilotti, M. W., Ogilvie, P., Callan, J., & Nyberg, E. (2007). Structured retrieval for question answering. *SIGIR*.

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

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

Collins, M. (2002a). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. *EMNLP*.

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

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

Collins-Thompson, K., & Callan, J. (2005). Query expansion using random walk models. *CIKM*.

Cui, H., Sun, R., Li, K., Kan, M.-Y., & Chua, T.-S. (2005). Question answering passage retrieval using dependency relations. *SIGIR*.

de Marneffe, M.-C., MacCartney, B., & Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. *LREC*.

Diligenti, M., Gori, M., & Maggini, M. (2005). Learning web page scores by error back-propagation. *IJCAI*.

Erkan, G., & Radev, D. R. (2004). Lexpagerank: Prestige in multi-document text summarization. *EMNLP*.

Etzioni, O., Cafarella, M., Downey, D., Shaked, A.-M. P. T., Soderland, S., S.Weld, D., & Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, *165*, 91–134.

Grefenstette, G. (1994). *Explorations in automatic thesaurus discovery*. Kluwer Academic Publishers, Dordrecht.

Hughes, T., & Ramage, D. (2007). Lexical semantic relatedness with random graph walks. *EMNLP*.

Keenan, E., & Comrie, B. (1977). Noun phrase accessibility and universal grammar. *Linguistic Inquiry*, *8*.

Lin, D. (1998). Automatic retrieval and clustering of similar words. *COLING-ACL*.

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

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

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

Padó, S., & Lapata, M. (2007). Dependency-based construction of semantic space models. *Computational Linguistics, 33*.

Quirk, C., Menezes, A., & Cherry, C. (2005). Dependency tree translation: Syntactically informed phrasal smt. *ACL*.

Resnik, P., & Diab, M. (2000). Measuring verb similarity. *CogSci*.

Snow, R., Jurafsky, D., & Ng, A. Y. (2005). Learning syntactic patterns for automatic hypernym discovery. *NIPS*.

Tong, H., Faloutsos, C., & Pan, J.-Y. (2006). Fast random walk with restart and its applications. *ICDM*.

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