Combining Strategies for Extracting Relations from Text Collections

Eugene Agichtein Eleazar Eskin Luis Gravano

Department of Computer Science
Columbia University
{eugene,eeskin,gravano}@cs.columbia.edu

Abstract

Text documents often contain valuable structured data that is hidden in regular English sentences. This data is best exploited if available as a relational table that we could use for answering precise queries or for running data mining tasks. Our *Snowball* system extracts these relations from document collections starting with only a handful of user-provided example tuples. Based on these tuples, *Snowball* generates patterns that are used, in turn, to find more tuples. In this paper we introduce a new pattern and tuple generation scheme for *Snowball*, with different strengths and weaknesses than those of our original system. We also show preliminary results on how we can combine the two versions of *Snowball* to extract tuples more accurately.

1 Introduction

Text documents often hide valuable *structured data*. For example, a collection of newspaper articles might contain information on the *location* of the headquarters of a number of *organizations*. The web contains millions of pages whose text hides data that would be best exploited in structured form.

Brin [Bri98] proposed the idea of DIPRE, which uses bootstrapping for extracting structured relations (or tables) from the web. A key assumption is that the table to be extracted appears redundantly in the document collection. As a result of this assumption, the patterns that DIPRE generates need not be overly general to capture *every instance* of an organization-location tuple. In effect, a system based on the DIPRE method will perform reasonably well even if certain instances of a tuple are missed, as long as the system captures one such instance. This approach is in contrast with the goals of traditional information extraction research, where a system attempts to extract as much information as possible from each *document* [Gri97]. DIPRE, on the other hand, attempts to build the most comprehensive table from *all of the documents in the collection*. In [AG00] we built on this approach and introduced *Snowball*. We developed a method for defining and representing extraction patterns that is at the same time flexible, so that we capture most of the tuples that are hidden in the text in our collection, and selective, so that we do not generate invalid tuples. We also introduced a strategy for estimating the reliability of the extracted patterns and tuples. Finally, we presented a scalable evaluation methodology and associated metrics, which we used for large-scale experiments over collections of over 300,000 real documents. Our experiments showed that *Snowball* was able to extract more than 80% of the organization-location pairs mentioned in the collection with high precision.

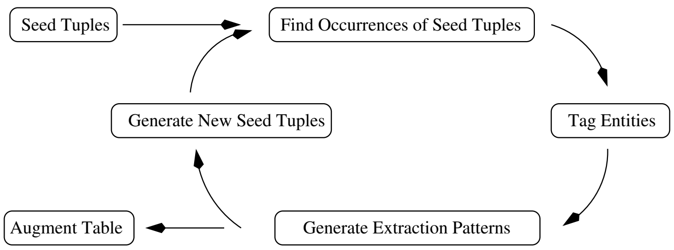


Figure 1: The main components of *Snowball*.

Organization	*Location of Headquarters*
MICROSOFT	REDMOND
EXXON	IRVING
IBM	ARMONK
BOEING	SEATTLE
INTEL	SANTA CLARA

Table 1: User-provided example tuples.

The basic architecture of *Snowball* is shown in Figure 1. Initially, we provide *Snowball* with a handful of instances of valid organization-location pairs such as the tuple <*Microsoft, Redmond*> (Table 1). Our system searches for occurrences of the example tuples' organizations and locations in the documents, identifying text lines where an organization and its corresponding location occur together. From these tagged example contexts, the system learns patterns that would indicate the desired relationship. For instance, from examining the occurrences of the seed tuples, we might learn that a context *"<LOCATION>-based <ORGANIZATION>"* is likely to indicate that *LOCATION* is the headquarters of the *ORGANIZATION*. Patterns built from examples like these are then used to scan through the corpus, discovering new tuples. The new tuples are evaluated, the most reliable ones are used as the new seed tuples, and the process repeats. A key step in generating and later matching patterns is finding where <*ORGANIZATION*> and <*LOCATION*> entities occur in the text. For this we tag the text documents using the MITRE corporation's Alembic Workbench [DAH$^+$97].

Related Work Brin's DIPRE method and our *Snowball* system both address issues that have long been the subject of information extraction research. However, DIPRE and *Snowball* do not attempt to extract *all* the relevant information from each document, which has been the goal of traditional information extraction systems [Gri97, FSM$^+$95]. One of the major challenges in information extraction is the necessary amount of manual tagging involved in training the system for each new task. [Ril96] generates extraction patterns automatically by using a training corpus of documents that were manually marked as either relevant or irrelevant for the topic. This approach requires less manual labor than to tag the documents, but nevertheless the effort involved is substantial. [CDF$^+$99] describes machine learning techniques for creating a knowledge base from the web, consisting of classes of entities and relations, by exploiting the content of the documents, as well as the link structure of the web. This method requires training over a large set of web pages, with relevant document segments manually labeled, as well as a large training set of page-to-page relations.

Finally, a number of systems use unlabeled examples for training. This direction of research is closest to our work. Specifically, the approach we are following falls into the broad category of bootstrapping techniques that have been successfully applied in other contexts. [Yar95] demonstrated a bootstrapping technique for disambiguating senses of ambiguous nouns. [CS99] and [RJ99] use bootstrapping to classify named entities in text. [YS99] describes an extension of DIPRE to mining the Web for acronyms and their expansions. [BM98] presents a methodology and theoretical framework for combining unlabeled examples with labeled examples to boost performance of a learning algorithm for classifying web pages. While the underlying principle of using the systems' output to generate the training input for the next iteration is the same for all of these approaches, the tasks are different enough to require specialized methodologies.

Our Contributions In this paper we consider two alternative methods for representing the textual contexts around the tuples that we want to identify. In Section 2.1 we briefly review the original *Snowball* system that we presented in [AG00], and which we refer to as *Snowball-VS* in this paper. *Snowball-VS* considers the textual context around the entities as an unordered collection of keywords. In Section 2.2 we introduce *Snowball-SMT*, a new system that takes advantage of the order of the words in the contexts. In Section 3 we present our preliminary exploration of methods to combine these complementary systems. Our approach allows us to exploit different representations of data for the problem. In Section 4 we outline the experimental setup and evaluation methodology for the experiments in Section 5.

Section 6 contains our preliminary conclusions and a discussion of future work.

2 Snowball

In this section, we explore different methods to learn patterns and generate tuples for *Snowball*: *Snowball-VS*, our original implementation [AG00], uses a vector-space model, whereas *Snowball-SMT*, a new system that we present in this paper, represents text as an ordered sequence of terms.

2.1 *Snowball-VS*

Snowball-VS is initially given a handful of example tuples. For every such organization-location tuple $< o, \ell >$, *Snowball-VS* finds segments of text in the document collection where o and ℓ occur close to each other, and analyzes the text that "connects" o and ℓ to generate extraction patterns that will later be used to discover new tuples.

Generating Patterns and Tuples A crucial step in the mining process is the generation of patterns that will be used to find new tuples in the documents. Ideally, we would like patterns both to be *selective*, so that they do not generate incorrect tuples, and to have high *coverage*, so that they identify many new tuples.

To improve the generality of the patterns, we represent the left, middle, and right "contexts" associated with a pattern analogously to the way the vector-space model of information retrieval represents documents and queries [Sal89]. Thus, the *left*, *middle*, and *right* contexts are three vectors associating weights with terms. These weights indicate the importance of each term in the corresponding context. An example of a *Snowball-VS* pattern is the 5-tuple *<{<the, 0.2>}, LOCATION, {<-, 0.5>, <based, 0.5>}, ORGANIZATION, {}>*. This pattern will match strings like "the Irving-based Exxon Corporation...". To match text portions with our 5-tuple representation of patterns, *Snowball-VS* also associates a 5-tuple t with each document portion that contains two named entities with the correct tag (i.e., *LOCATION* and *ORGANIZATION* in our scenario), and matches it against the 5-tuple pattern p, where the degree of match *Match(t, p)* is calculated as the normalized sum of inner products of the corresponding *left*, *middle*, and *right* context vectors.

In order to generate a pattern, we group occurrences of known tuples in documents that occur in similar contexts. More precisely, *Snowball-VS* generates a 5-tuple for each string where a seed tuple occurs, and then clusters these 5-tuples using a simple single-pass bucket clustering algorithm [FBY92], using the *Match* function described above to calculate the similarity between the 5-tuples, with minimum similarity threshold τ_{sim}. The pattern is represented as the representative 5-tuple of the cluster: the *left* vectors in the 5-tuples of clusters are represented by a *centroid* $\bar{l_s}$. Similarly, we collapse the *middle* and *right* vectors into $\bar{m_s}$ and $\bar{r_s}$, respectively. These three centroids, together with the original tags (which are the same for all the 5-tuples in the cluster), form a *Snowball-VS* pattern $< \bar{l_s}, tag_1, \bar{m_s}, tag_2, \bar{r_s} >$. As an initial filter, we eliminate all patterns *supported* by fewer than τ_{sup} seed tuples.

Using these patterns, *Snowball-VS* scans the collection to discover new tuples. The system first identifies sentences that include an organization and a location, as determined by the named-entity tagger. For a given text segment with an associated organization o and location ℓ, *Snowball-VS* generates the 5-tuple $t =< l_c, tag_1, m_c, tag_2, r_c >$. A candidate tuple $< o, \ell >$ is generated if there is a pattern t_p such that $Match(t, t_p) \geq \tau_{sim}$, where τ_{sim} is the clustering similarity threshold. Each candidate tuple may be generated multiple times from different text segments, using either a single pattern to match the segments, or different patterns. For each candidate tuple, we store the set of *patterns* that generated it, each with an associated degree of match. *Snowball-VS* uses this information, together with information about the selectivity of the patterns, to decide what candidate tuples to actually add to the table that it is constructing.

Evaluating Patterns and Tuples We can weigh the *Snowball-VS* patterns based on their selectivity, and trust the tuples that they generate accordingly. Thus, a pattern that is not selective will have a low *confidence* value. The tuples generated by such a pattern will be discarded, unless they are supported by selective patterns. Intuitively, the confidence of a tuple will be high if it is generated by several highly selective patterns.

We estimate the selectivity of each pattern during our scan of the corpus to discover new tuples. If a sentence matches one of our patterns and contains an organization that we have discovered in an earlier iteration of the system, we check whether the new location agrees with a previously extracted, "known" headquarters location for this organization. If so, this new match is considered *positive* for the pattern. Otherwise, the match is *negative*. This allows us to compute the *confidence* of the pattern. Note that this confidence computation assumes that organization is a key for the relation that we are extracting (i.e., two different tuples in a valid instance of the relation cannot agree on the organization attribute). Estimating the confidence of the patterns in discovering relations without such a single-attribute key is part of our future work. The *confidence* of a pattern P is defined as:

$$Conf(P) = \frac{P.positive}{(P.positive + P.negative)}$$

where P.positive is the number of positive matches for P and P.negative is the number of negative matches. For illustration purposes, Table 2 lists three representative patterns that *Snowball-VS* extracted from the document collection described in Section 4.

Conf	*middle*	*right*
1	`<based, 0.53>,<in, 0.53>`	`<, , 0.01>`
0.69	`<', 0.42> <s, 0.42>< headquarters, 0.42><in, 0.12>`	
0.61	`<(, 0.93>`	`<), 0.12>`

Table 2: Actual patterns discovered by *Snowball*. (For all three of these patterns, the *left* vectors are empty, *tag1* = *ORGANIZATION*, and *tag2* = *LOCATION*.)

Having scored the patterns, we are now able to evaluate the new candidate tuples. For each tuple we store the set of patterns that produced it, together with the degree of match between the context in which the tuple occurred and the matching pattern. The *confidence* of a candidate tuple T is:

$$Conf(T) = 1 - \prod_{i=0}^{|P|} (1 - (Conf(P_i) \cdot Match(C_i, P_i)))$$

where $P = \{P_i\}$ is the set of patterns that generated T and C_i is the context associated with an occurrence of T that matched P_i with degree of match $Match(C_i, P_i)$. From the set of discovered tuples, the most reliable ones are selected as seed for the next iteration of the system. A tuple T is added to the seed set if $Conf(T) \geq \tau_{min}$.

2.2 *Snowball-SMT*

The *Snowball-VS* patterns model each context as a bag of words, ignoring word order. These patterns then concentrate on the presence or absence of certain keywords. For instance, a context such as "... where Microsoft is located. Which Silicone Valley startup ..." will match a pattern $<\{\}$, *ORGANIZATION*, $\{$<*which, 0.5*>, <*is, 0.5*>, <*located, 0.5*>, <*in, 0.5*>$\}$, *LOCATION*, $\{\}>$, producing an incorrect tuple <*Microsoft, Silicone Valley* >. In this section we introduce *Snowball-SMT*, a variant of *Snowball* that takes into account the *order* of the words in each context, while keeping the patterns flexible enough to have high coverage. For this purpose, we model the textual contexts as ordered sequences of tokens and try to estimate the probability of sentences containing an instance of the organization-location relationship.

Thus, if a seed organization and its correct location are mentioned in the same sentence, the text context surrounding the entities is converted into a sequence of tokens, and a positive example is added to the training set. If the location does not match the "known" headquarters of this organization, a negative example is added. In each iteration, *Snowball-SMT* is trained on this set of examples, and builds a model that best describes the training set. *Snowball-SMT* then scans the

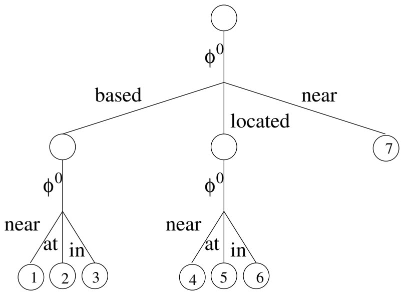


Figure 2: An example sparse Markov tree.

corpus again, generating a tuple each time that a sequence of terms in the context surrounding the entities is accepted by the model.

We represent contexts as ordered sequences using *sparse Markov transducers* (SMTs), which estimate a probability distribution conditioned on a sequence. In our problem, we compute the probability that a tuple is an organization-location pair conditioned on the sequence of terms that make up the context of the tuple. The probability distribution is conditioned on some of these words and not the others. We wish to represent a part of the conditional sequence of words as "don't care", or *ϕ-terms* in the probability model. For instance, the probability of a text fragment "near Boeing's renovated Seattle headquarters" contaning a tuple *T* = *<Boeing, Seattle>* would be calculated as

$$Conf(T) = P(T|near, \ 's, \phi^1, headquarters)$$

where the system ignores the term "renovated" as irrelevant.

More formally, a sparse Markov transducer is a conditional probability of the form:

$$P(T|\phi^{n_1}t_1\phi^{n_2}t_2...\phi^{n_k}t_k)$$

where T is the output label that *Snowball-SMT* returns upon recognizing a tuple. Each t_i is the ith term in the context surrounding the entities, arranged into a sequence by starting from the terms on the left of the leftmost entity, adding the terms between entities, and followed by the terms to the right of the rightmost entity (as in the example above). In the equation, ϕ^{n_i} represents n_i consecutive ϕ-terms, and for a sequence of length n, $n_1 + ... + n_k + k = n$.

To estimate SMTs we use a type of prediction suffix tree called a *sparse prediction tree*, which is representationally equivalent to sparse Markov transducers. These trees probabilistically map the context of a tuple to a probability that the tuple is an organization-location pair. A sparse prediction tree is a rooted tree where each node is either a leaf node or contains one branch labeled with ϕ^n ($n \geq 0$), which forks into a branch for each word. The paths from the root node to the leaf nodes represent the sequences of terms that make up the contexts surrounding the entities. Each leaf node stores an estimate of the probability that if the node was reached, the context that was used to generate the path to the node contains a valid organization-location tuple. Figure 2 shows a sparse Markov tree. For example, the node labeled 3 would be reached by following the terms making up the context *"<ORGANIZATION> based in <LOCATION>."*

A tree is used to obtain a probability for a tuple by following the context from the root node to a leaf node skipping a token in the context for each ϕ along the path. The leaf node contains the tuple's probability of being an organization-location pair. The topology of a tree encodes the positions of the ϕ-terms in the probability distribution. Because we do not know the positions of the ϕ-terms for each context *a priori*, we do not know the best topology of the prediction tree to use. We approximate the best tree using a Bayesian mixture (weighted sum) technique. Instead of using a single tree, we use a weighted sum of all possible trees as our predictor. We then use a Bayesian update rule (described in Section 3) to update the weight of each tree based on its performance on a given element in the data set. At the end of this process, we have a weighted sum of trees in which the best performing trees in the set of all trees have the highest weights. The sparse prediction tree is rebuilt from scratch from the set of positive and negative examples on each iteration of *Snowball-SMT*. An in-depth description of SMTs is given in [Esk00]. The sparse Markov transducer probability modeling method is an extension of adaptive mixtures of probabilistic transducers [Sin97, PS99].

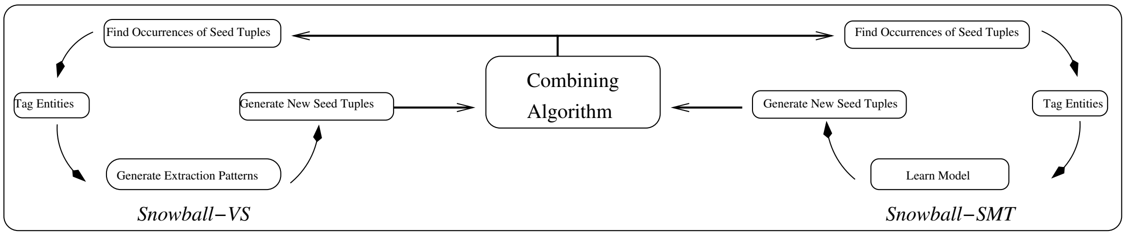


Figure 3: Combining *Snowball-VS* and *Snowball-SMT* into one system.

3 Combining *Snowball-VS* and *Snowball-SMT*

The two systems that we used in our experiments, *Snowball-VS* and *Snowball-SMT*, focus on two different aspects of the textual context: the presence or absence of keywords that tend to indicate the correct relationships (*Snowball-VS*), and the order of words in the contexts surrounding the entities (*Snowball-SMT*). We explore how to combine the two systems with the goal of improving our overall extraction accuracy. Combining predictors to increase accuracy is an active area of research. Some of the methods we considered include *sleeping-experts* [Blu97], *boosting by majority* [Fre95], and *co-training* [BM98]. In this section, we explore preliminary ways in which we can combine *Snowball-VS* and *Snowball-SMT*. (We discuss this issue further in Section 6.)

Initially, both *Snowball-VS* and *Snowball-SMT* receive the same set of seed tuples (Figure 3). Each system runs for one iteration, producing a set of tuples $Seed_{VS}$ and $Seed_{SMT}$, respectively. These two sets of tuples are combined into one set $Seed_{Combined}$ (we will describe how shortly). $Seed_{Combined}$ is then used as the set of seed tuples for both *Snowball-VS* and *Snowball-SMT* and both systems are run for another iteration. This process repeats until we stop discovering new tuples. The final step of the extraction process returns the set $Seed_{Combined}$, containing the combination of the final set of tuples discovered by *Snowball-VS* and *Snowball-SMT*.

We explored three options for combining the tuples discovered by *Snowball-VS* and *Snowball-SMT* to create the new set of seed tuples: the *Union*, the *Intersection*, and the weighted *Mixture* of the tuples produced by the individual systems. The *Intersection* strategy was motivated by [BM98]. To implement the *Union* and *Intersection* strategies, the sets of tuples produced by *Snowball-VS* and *Snowball-SMT* are filtered using each system's individual thresholds for generating seed tuples, and the resulting sets are combined. In the *Union* model, seed tuples proposed by either *Snowball-VS* or *Snowball-SMT* are added to the combined set, unless the locations that the two systems propose for the same organization do not match (based on our unique-key assumption, only one of these can be correct). In the *Intersection* model, only seed tuples proposed by *both Snowball-VS* and *Snowball-SMT* are added.

To implement the weighted *Mixture* model, a tuple T is added to the combined set if $Conf(T) \geq \tau_{min}$ where $Conf(T)$ is calculated as the weighted sum of the confidence values that each system assigns to tuple T. The weights are based on the accuracy of each system over the training data. To calculate these weights, we use an implementation of a Bayesian update rule. We first calculate the absolute weight W'_{VS} of *Snowball-VS* as:

$$W'_{VS} = \sum_{correct\ tuples\ T} log(Conf(T)) + \sum_{incorrect\ tuples\ T} log(1 - Conf(T))$$

The absolute weight of W'_{SMT} of *Snowball-SMT* is calculated similarly. Then the relative weights W_{VS} and W_{SMT} are:

$$W_{VS} = \frac{W'_{VS}}{W'_{VS} + W'_{SMT}} \quad and \quad W_{SMT} = \frac{W'_{SMT}}{W'_{VS} + W'_{SMT}}$$

Finally, the combined confidence in tuple T is defined as:

$$Conf(T) = W_{VS} \cdot Conf_{VS}(T) + W_{SMT} \cdot Conf_{SMT}(T)$$

In our experiments, we compare the performance of *Snowball-VS* and *Snowball-SMT* as well as that of the three combining strategies.

	Organization-Location Pairs	
Occurrences	*Training Collection*	*Test Collection*
0	5455	4642
1	3787	3411
2	2774	2184
5	1321	909
10	593	389

Table 3: Occurrence statistics of the test tuples in the experiment collections.

4 Experimental Setting

The goal of *Snowball* is to extract as many valid tuples as possible from the text collection. We do not attempt to capture every *instance* of such tuples. Instead, we exploit the fact that these tuples will tend to appear multiple times in the types of collections that we consider. As long as we capture one instance of such a tuple, we will consider our system to be successful for that tuple.

Methodology To evaluate this task, we adapt the recall and precision metrics from information retrieval to quantify how accurate and comprehensive our *combined table of tuples* is [Sal89]. Our metric for evaluating the performance of an extraction system over a collection of documents D is based on determining *Ideal*, the set of all the known test tuples that appear in collection D. After identifying *Ideal*, we compare it against the tuples produced by the system, *Extracted*, using adapted precision and recall metrics [AG00]. To create the *Ideal* set automatically, we start by considering a large, publicly available directory of more than 13,000 organizations provided on the "Hoover's Online" web site[1]. To determine the target set of tuples *Ideal* from the Hoover's-compiled table above, we keep only the tuples that have the organization mentioned together with their location in the collection. We match possible variations of companies' names by using Whirl [Coh98], a research tool developed at AT&T Research Laboratories for integrating similar textual information.

An alternative to using our *Ideal* metric to estimate precision could be to sample the extracted table, and check each value in the sample tuples by hand. (Similarly, we could estimate the recall of the system by sampling documents in the collection, and checking how many of the tuples mentioned in those documents the system discovers.) For completeness, we also report precision estimates using sampling in Section 5. Please refer to [AG00] for more details on our evaluation methodology.

Document Collections Our experiments used large collections of real newspaper articles from the North American News Text Corpus, available from LDC [2]. This corpus includes articles from the Los Angeles Times, The Wall Street Journal, and The New York Times for 1994 to 1997. We split the corpus into two collections: training and test. The *training* collection consists of 178,000 documents, all from 1996. The *test* collection is composed of 142,000 documents, from 1995 and 1997.

Both *Snowball* and DIPRE rely on tuples appearing multiple times in the document collection at hand. To analyze how "redundant" the training and test collections are, we report in Table 3 the number of tuples in the *Ideal* set for each frequency level. For example, 5455 organizations in the *Ideal* set are mentioned in the training collection, and 3787 of them are mentioned in the same line of text with their location at least once. So, if we wanted to evaluate how our system performs on extracting tuples that occur at least once in the training collection, the *Ideal* set that we will create for this evaluation will contain 3787 tuples. The first row of Table 3, corresponding to zero occurrences, deserves further explanation. If we wanted to evaluate the performance of our system on *all* the organizations that were

[1] `http://www.hoovers.com`
[2] `http://www.ldc.upenn.edu`

System	*Parameter*	*Value*	*Description*
Snowball-VS	τ_{sim}	0.6	degree of match
	τ_t	0.9	seed confidence
	τ_{sup}	2	pattern support
	τ_{final}	0.3	tuple confidence
	window	2	length of *left* and *right* contexts
Snowball-SMT	τ_t	0.99	seed confidence
	τ_{final}	0.99	tuple confidence

Table 4: Parameter values used for evaluating *Snowball-VS* and *Snowball-SMT* on the test collection.

Strategy	*Snowball-VS*		*Snowball-SMT*		*Combined*	
	τ_{seed}	τ_{final}	τ_{seed}	τ_{final}	τ_{seed}	τ_{final}
Intersection	0.9	0.3	0.99	0.99	-	-
Union	0.9	0.3	0.99	0.99	-	-
Mixture	-	-	-	-	0.97	0.6

Table 5: Parameter values used for generating new seed tuples by combining *Snowball-VS* and *Snowball-SMT*.

mentioned in the corpus, even if the appropriate location never occurred near its organization name anywhere in the collection, we would include all these organizations in our *Ideal* set. So, if the system attempts to "guess" the value of the location for such an organization, any value that the system extracts will automatically be considered wrong in our evaluation.

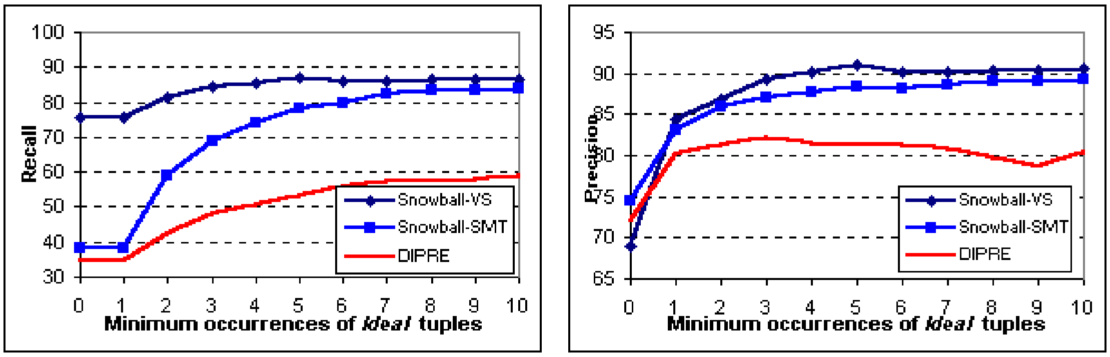
5 Experimental Results

In [AG00] we extensively examined the performance of *Snowball-VS*, together with an implementation of DIPRE. In this paper we compare the performance of *Snowball-VS* and *Snowball-SMT*, and explore the effect of combining the two into a single system.

In the *training phase* of our experiments, we empirically determined the best individual operating parameters for *Snowball-VS* and *Snowball-SMT* by running the systems on the training collection. We then evaluated the systems on the test collection using the parameters in Table 4. As we discussed, the only input to both *Snowball* systems during this evaluation on the test collection were the five seed tuples of Table 1. All the extraction patterns were learned from scratch by running each *Snowball* system on a previously unseen test collection using the operational parameters of Table 4.

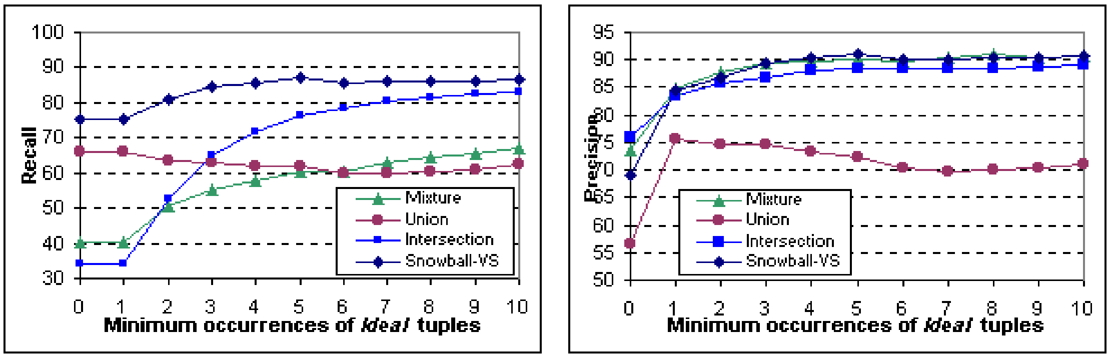
Figure 4 shows the performance of the individual systems as they attempt to extract test tuples that are mentioned more times in the test collection. For example, *Snowball-VS* correctly extracts 85% of the tuples that occur at least three times in the collection, with precision of 89%. Not surprisingly, *Snowball-VS* performs increasingly well as the number of times that the test tuples are required to be mentioned in the collection is increased. Also, notice that while DIPRE has better precision than *Snowball-VS* at the 0-occurrence level (72% vs. 69%), *Snowball-VS* has at all occurrence levels significantly higher recall than DIPRE. However, *Snowball-SMT* has the highest precision when we consider all tuples (75%), and its precision steadily increases for more frequently occurring tuples.

We explored the three combining strategies on the training collection and used the parameters in Table 5 to run the combined system on the test collection. The individual systems were run using the parameters in Table 4. The τ_{seed} parameter of Table 5 is the minimum confidence value for a tuple to be chosen as seed by each individual system, and the τ_{final} parameter is the threshold used to filter the final table. As we can see in Figure 5, the simple combining



(a) (b)

Figure 4: Recall (a) and precision (b) of DIPRE, *Snowball-VS*, and *Snowball-SMT* (test collection).



(a) (b)

Figure 5: Recall (a) and precision (b) of the combined system for the *Intersection*, *Union*, and *Mixture* strategies, against *Snowball-VS* (test collection).

strategies we explored do not help us discover new tuples, but can be used to improve the precision of the extracted table. This claim is further supported by randomly sampling the tables produced by *Snowball-VS*, *Snowball-SMT*, and those produced by using the *Intersection*, *Union*, and *Mixture* combining strategies. The samples were manually checked for accuracy of the discovered tuples, with results shown in Table 6. We classify the errors into three types (*Location*, *Organization*, and *Relationship*), where the former two are due to the errors of the named-entity tagger, and the latter is completely the extraction system's fault. As we can see, *Snowball-SMT* produces few incorrect tuples (24 out a sample of 100) while *Snowball-VS* is less selective, producing 48 incorrect tuples out of a sample of 100. The incorrect tuples are due mainly to erroneously tagging phrases as organizations (41 out of the 48 incorrect tuples for *Snowball-VS*). If we were querying the table extracted by *Snowball-VS* by organization, we would expect to find the correct headquarters for the organization approximately 88% of the time ($\frac{52\ correct\ tuples}{52+6\ incorrect\ locations+1\ incorrect\ relationship} \cdot 100\%$). Observe that the *Intersection* strategy appears to produce the cleanest table overall (81 tuples out of 100 are correct).

Thus, if we want a high-recall system, we should run *Snowball-VS*. Alternatively, if we want to create a table of high-quality tuples, we should run *Snowball-SMT*. Finally, we could combine the two systems using the *Intersection* strategy to create a table with high precision that also approaches *Snowball-VS*'s recall values for high-frequency tuples.

6 Conclusions and Future Work

This paper presents significant extensions of *Snowball*, a system for extracting relations from large collections of plain-text documents that requires minimal training for each new scenario. We compared two alternatives for representing text for our extraction task, and presented preliminary results on combining the systems.

			Type of Error		
	Correct	*Incorrect*	*Location*	*Organization*	*Relationship*
Snowball-VS	52	48	6	41	1
Snowball-SMT	76	24	3	19	2
Union	49	51	6	42	3
Mixture	73	27	4	19	4
Intersection	81	19	4	14	1

Table 6: Manually computed precision estimate, derived from a random sample of 100 tuples from each extracted table.

We only evaluated our techniques on plain text documents, and it would require future work to adopt our methodology to HTML data. While HTML tags can be naturally incorporated into *Snowball*'s pattern representation, it is problematic to extract named-entity tags from arbitrary HTML documents. State-of-the-art taggers rely on clues from the text surrounding each entity, which may be absent in HTML documents that often rely on visual formatting to convey information.

In the context of processing HTML data, we plan to explore the question of combining complementary information as part of the *Snowball* system. In present work, we only had two systems to combine, and sophisticated methods for combining predictors (e.g., [Fre95, Blu97]) were not likely to make a significant impact. In addition to the two systems that operate on the text immediately surrounding the entities, we could have a third system that considers the links between documents, each containing one of the attributes in the relation. Having one or multiple systems operating over this additional information will allow us to compare and exploit the benefits of more sophisticated methods for combining predictors. More importantly, this might result in even higher quality extraction strategies.

We have assumed throughout that the attributes of the relation we extract (i.e., organization and location) correspond to named entities that our tagger can identify accurately. As we mentioned, named-entity taggers like Alembic can be extended to recognize entities that are distinct in a context-independent way (e.g., numbers, dates, proper names). For some other attributes, we will need to extend *Snowball* so that its pattern generation and matching could be anchored around, say, a noun phrase as opposed to a named entity as in this paper. In the future, we will also generalize *Snowball* to relations of more than two attributes. Finally, another open problem is how to extend our tuple and pattern evaluation strategy of Section 2.1 so that it does not rely on an attribute being a key for the relation.

Acknowledgements This material is based upon work supported by the National Science Foundation under Grant No. IIS-9733880. We also thank Kazi Zaman for his helpful comments.

References

[AG00] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. *To appear in Proceedings of the 5th ACM International Conference on Digital Libraries*, June 2000. Accessible at `http://www.cs.columbia.edu/~eugene/papers/dl00.pdf`.

[Blu97] Avrim Blum. Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, 1997.

[BM98] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 1998 Conference on Computational Learning Theory*, 1998.

[Bri98] Sergey Brin. Extracting patterns and relations from the World-Wide Web. In *Proceedings of the 1998 International Workshop on the Web and Databases (WebDB'98)*, March 1998.

[CDF+99] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, 1999.

[Coh98] William Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the 1998 ACM International Conference on Management of Data (SIGMOD'98)*, 1998.

[CS99] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[DAH+97] David Day, John Aberdeen, Lynette Hirschman, Robyn Kozierok, Patricia Robinson, and Marc Vilain. Mixed-initiative development of language processing systems. In *Proceedings of the Fifth ACL Conference on Applied Natural Language Processing*, April 1997.

[Esk00] Eleazar Eskin. Detecting errors within a corpus using anomaly detection. *To appear in Proceedings of 2000 Conference of the North American Association for Computational Linguistics*, April 2000.

[FBY92] William B. Frakes and Ricardo Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

[Fre95] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 1995.

[FSM+95] D. Fisher, S. Soderland, J. McCarthy, F. Feng, and W. Lehnert. Description of the UMass systems as used for MUC-6. In *Proceedings of the 6th Message Understanding Conference*. Columbia, MD, 1995.

[Gri97] Ralph Grishman. Information extraction: Techniques and challenges. In *Information Extraction (International Summer School SCIE-97)*. Springer-Verlag, 1997.

[PS99] Fernando Pereira and Yoram Singer. An efficient extension to mixture techniques for prediction and decision trees. *Machine Learning*, 36(3):183–199, 1999.

[Ril96] Ellen Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1044–1049, 1996.

[RJ99] Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.

[Sal89] Gerard Salton. *Automatic Text Processing: The transformation, analysis, and retrieval of information by computer*. Addison-Wesley, 1989.

[Sin97] Yoram Singer. Adaptive mixtures of probabilistic transducers. *Neural Computation*, 9(8):1711–1733, 1997.

[Yar95] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196. Cambridge, MA, 1995.

[YS99] Jeonghee Yi and Neel Sundaresan. Mining the web for acronyms using the duality of patterns and relations. In *Proceedings of the 1999 Workshop on Web Information and Data Management*, 1999.