Snowball: Extracting Relations from Large Plain-Text Collections

Eugene Agichtein Luis Gravano
Department of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027-7003, USA
{eugene, 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. We explore a technique for extracting such tables from document collections that requires only a handful of training examples from users. These examples are used to generate extraction patterns, that in turn result in new tuples being extracted from the document collection. We build on this idea and present our *Snowball* system. *Snowball* introduces novel strategies for generating patterns and extracting tuples from plain-text documents. At each iteration of the extraction process, *Snowball* evaluates the quality of these patterns and tuples without human intervention, and keeps only the most reliable ones for the next iteration. In this paper we also develop a scalable evaluation methodology and metrics for our task, and present a thorough experimental evaluation of *Snowball* and comparable techniques over a collection of more than 300,000 newspaper documents.

KEYWORDS: Text databases, information extraction, bootstrapping.

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*. If we need to find the location of the headquarters of, say, Microsoft, we could try and use traditional information-retrieval techniques for finding documents that contain the answer to our query [15, 14]. Alternatively, we could answer such a query more precisely if we somehow had available a *table* listing all the organization-location pairs that are mentioned in our document collection. A *tuple* $< o, \ell >$ in such table would indicate that the headquarters of organization o are in location ℓ, and that this information was present in a document in our collection. Tuple `<Microsoft, Redmond>` in our table would then provide the answer to our query. The web contains millions of pages whose text hides data that would be best exploited in structured form. If we could build structured tables from the information hidden in unstructured text, then we would be able to run more complex queries and analysis over these tables, and report precise results.

In this paper we develop the *Snowball* system for extracting structured data from plain-text documents with *minimal human participation*. Our techniques build on the ideas and general approach introduced by Brin [2], which we describe next.

DIPRE: Dual Iterative Pattern Expansion To extract a structured *relation* (or *table*) from a collection of HTML documents, Brin introduced the DIPRE method [2]. DIPRE works best in an environment like the World-Wide Web, where the table *tuples* to be extracted will tend to appear in uniform contexts repeatedly in the collection documents (i.e., in the available HTML pages). DIPRE exploits this redundancy and inherent structure in the collection to extract the target relation with minimal training from a user. In fact, DIPRE requires that the user just provide a handful of valid tuples of the target relation, with no other training. (This is in contrast to the way traditional information extraction systems operate.) We describe the DIPRE method, which forms the basis for the *Snowball* system that we present in Section 2.

As in the rest of the paper, we focus the presentation on the organization-location scenario defined above. Hence, in this context DIPRE's goal is to extract a table with all the organization-location tuples that appear in a given document collection. Initially, we provide DIPRE with a handful of instances of valid organization-location pairs. For example, we may indicate that `<Microsoft, Redmond>` is a valid pair, meaning that Microsoft is an organization whose headquarters are located in Redmond. Similarly, we provide DIPRE with a few other examples, as Table 1 shows. In addition, the user provides a general regular expression that the entities must match. For example, a potential organization value must match a regular expression $[A-Z0-9][A-Za-z0-9.,:'\ \#!?;\&]\{4,45\}[A-Za-z0-9]$. This regular expression says that an organization must begin either with a capital letter (e.g., Microsoft), or with a number, (e.g., 3Com), and be followed by four to 45 characters, ending in a letter or a number. This is all the training that DIPRE requires from the user.

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

Table 1: User-provided example tuples for DIPRE.

Computer servers at **Microsoft**'s headquarters in **Redmond**...
Exxon, **Irving**, said it will boost its stake in the...
In midafternoon trading, shares of **Irving**-based **Exxon** fell ...
The **Armonk**-based **IBM** has introduced a new line ...
...operate from **Boeing**'s headquarters in **Seattle**.
Intel, **Santa Clara**, cut prices of its Pentium...

Figure 1: Occurrences of the initial example tuples in text documents.

After this initial training phase, DIPRE looks for instances of the example organizations and locations in the text documents. Some occurrences of the example tuples in documents are listed in Figure 1. Then, DIPRE examines the text that surrounds the initial tuples. For example, DIPRE inspects the context surrounding Microsoft and Redmond in "`computer servers at` **`Microsoft`**`'s headquarters in` **`Redmond`**" to construct a pattern "`<STRING1>'s headquarters in <STRING2>`." Other possible patterns are listed in Figure 2.

The DIPRE algorithm for generating the patterns is described in detail in [2]. Briefly, the algorithm represents an occurrence of a seed tuple as a seven-tuple: $< o, \ell,$ `order, url, left, middle, right`$>$, where `url` is the URL of the source document where $< o, \ell >$ was found, `order` is 1 if o appeared before ℓ and 0 otherwise, and `left`, `middle`, and `right` are the parts of the context that surrounds the occurrence of $< o, \ell >$ in the document. A pattern (represented as a five tuple `<order, urlprefix, left, middle, right>`) is created by grouping together occurrences that all have equal `middle` string and `order`, and then setting the `urlprefix`, `left`, and `right` of the pattern to the longest common substrings of all the `url`, `left`, and `right` strings, respectively. The patterns are then filtered by requiring that each pattern be supported by more then one seed tuple, and that `urlprefix`, `left`, `middle`, and `right` all be non-empty.

Finally, after generating a number of patterns from the initial seed tuples, DIPRE scans the available documents in search of segments of text that match the patterns. As a result of this process, DIPRE generates new tuples and uses them as the new "seed." DIPRE starts the process all over again by searching for these new tuples in the documents to identify new promising patterns.

As we have seen, unlike most machine-learning systems for information extraction, DIPRE requires no training other than providing a handful of initial seed tuples and specifying the general pattern that the elements of the extracted tuples must match. By acquiring additional training examples automatically, DIPRE aims at capturing most of the tuples mentioned in the collection. A key assumption behind this method is that the table to be extracted appears redundantly in the document collection in question. 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. Instead, a more critical goal is to discard patterns that are not selective enough, and that may generate invalid tuples. (To combat this problem, [2] suggests assigning weights to patterns and tuples, and notes a potential relationship of this problem to Latent Semantic Indexing [7].) 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.

```
<STRING1>'s headquarters in <STRING2>
<STRING2>-based <STRING1>
<STRING1>, <STRING2>
```

Figure 2: Initial DIPRE patterns. `<STRING1>` and `<STRING2>` are regular expressions that would match an organization and a location, respectively.

Related Work Brin's DIPRE method and our *Snowball* system that we introduce in this paper both address issues that have long been the subject of information extraction research. Our task, though, is different in that we do not attempt to extract *all* the relevant information from each document, which has been the goal of traditional information extraction systems [10]. One of the major challenges in information extraction is the necessary amount of manual labor involved in training the system for each new task. This challenge has been addressed in different ways. One approach is to build a powerful and intuitive graphical user interface for training the system, so that domain experts can quickly adopt the system for each new task [16]. Nevertheless, these systems still require substantial expert manual labor to port the system to each new domain. In contrast, *Snowball* and DIPRE require only a handful of example tuples for each new scenario.

Another approach is to train the system over a large *manually tagged* corpus, where the system can apply machine learning techniques to generate extraction patterns [8]. The difficulty with this approach is the need for a large tagged corpus, which again involves a significant amount of manual labor to create. To combat this problem, some methods have been proposed to use an untagged corpus for training. [12] describes generating extraction patterns automatically by using the training corpus that consists of sets of documents, which were manually separated into the relevant vs. irrelevant set for the topic. This approach requires less manual labor than to tag the documents, but nevertheless the effort involved is substantial. [5] 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. Their 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. Bootstrapping has been an attractive alternative in automatic text processing. [17] demonstrates a bootstrapping technique for disambiguating senses of words by starting with a small set of seed collocations for each word (e.g., seed collocation "life" to disambiguate the biological sense of the noun "plant"), and iteratively classifies the occurrences of the word into one of the appropriate senses. [4] uses bootstrapping to classify named entities in text. They exploit two orthogonal features for classifying named entities, i.e., the spelling of the entity itself (e.g., having a suffix "Corp."), and the context in which the entity occurs. They present an algorithm that classifies named entities with high accuracy. [13] presents a bootstrapping technique to extract patterns to recognize and classify named entities in text. [1] present 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 As we have discussed, [2] describes a method for extracting relations from the web using bootstrapping. Our *Snowball* system, which we present in this paper, builds on DIPRE. Our main contributions include:

- **Techniques for generating patterns and extracting tuples:** We propose the use of named-entity tags for anchoring the search of new tuples in the documents. Also, we develop a new strategy for defining and representing 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 (Sections 2.1 and 2.2).
- **Strategies for evaluating patterns and tuples:** Since the amount of training that *Snowball* requires is minimal, it is crucial that the patterns and tuples that are generated during the extraction process be evaluated. This way, *Snowball* will be able to eliminate unreliable tuples and patterns from further consideration. We develop strategies for estimating the reliability of the extracted patterns and tuples (Section 2.3).
- **Evaluation methodology and metrics:** Evaluating systems like *Snowball* and DIPRE is challenging: these systems are designed to work over large document collections, so manually inspecting all documents to build the "perfect" table that should be extracted is just not feasible. We introduce a scalable evaluation methodology and associated metrics (Section 3), which we use in Sections 4 and 5 for large-scale experiments over collections of training and test documents. These collections have a total of over 300,000 real documents.

2 THE *SNOWBALL* SYSTEM

In this section we present the *Snowball* system, which develops key components of the basic DIPRE method that we described in the previous section. As Figure 3 shows, the *Snowball* architecture follows the general DIPRE outline. However, we will see that *Snowball* introduces key ideas that result in substantially better performance. More specifically, *Snowball* presents a novel technique to generate patterns and extract tuples from text documents, which we describe in Sections 2.1 and 2.2. Also, *Snowball* introduces a strategy for evaluating the quality of the patterns and the tuples that are generated in each iteration of the extraction process (Section 2.3). Only those tuples and patterns that are regarded as being "sufficiently reliable" will be kept by *Snowball* for the following iterations of the system (Section 2.3). These new strategies for generation and filtering of patterns and tuples improve the quality of the extracted tables significantly, as the experimental evaluation in Section 5 will show.

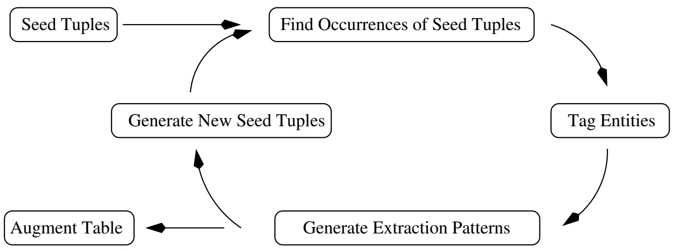


Figure 3: The main components of *Snowball*.

2.1 Generating Patterns

As we observed in Section 1, a crucial step in the table extraction 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. In this section, we introduce a novel way of generating such patterns from a set of seed tuples and a document collection.

Snowball is initially given a handful of example tuples. For every such organization-location tuple $< o, \ell >$, *Snowball* finds segments of text in the document collection where o and ℓ occur close to each other, just as DIPRE does, and analyzes the text that "connects" o and ℓ to generate patterns. A key improvement of *Snowball* from the basic DIPRE method is that *Snowball*'s patterns include named-entity tags. An example of such a pattern is `<LOCATION>-based <ORGANIZATION>`. This pattern will not match any pair of strings connected by "-based." Instead, `<LOCATION>` will only match a string identified by a tagger as an entity of type `LOCATION`. Similarly, `<ORGANIZATION>` will only match a string identified by a tagger as an entity of type `ORGANIZATION`. To understand the impact of using named-entity tags in the *Snowball* patterns, consider the pattern `<STRING2>-based <STRING1>`. This pattern matches the text surrounding correct organization-location tuples (e.g., "the Armonk-based IBM has introduced..."). Unfortunately, this pattern will also match any strings connected by "-

based," like "`computer-based learning`" or "`alcohol-based solvents.`" This might result in the inclusion of invalid tuples <`learning, computer`> and <`solvents, alcohol`> in our organization-location table. When used to generate patterns, these tuples may in turn result in wrong patterns.

In contrast, by using the version of the same pattern that involves named-entity tags, `<LOCATION>-based <ORGANIZATION>`, we have a better chance of avoiding this kind of spurious matches. Figure 4 shows additional patterns that *Snowball* might generate, based on the examples in Figure 1, which involve named-entity tags.

```
<ORGANIZATION>'s headquarters in <LOCATION>
<LOCATION>-based <ORGANIZATION>
<ORGANIZATION>, <LOCATION>
```

Figure 4: Patterns that exploit named-entity tags.

A key step in generating and later matching patterns like the one above is finding where `<ORGANIZATION>` and `<LOCATION>` entities occur in the text. For this, *Snowball* uses a state-of-the-art named-entity tagger, The MITRE Corporation's Alembic Workbench [6]. In addition to `ORGANIZATION` and `LOCATION` entities, Alembic can identify `PERSON` entities, and can be trained to recognize other kinds of entities. (See Section 6 for further discussion.)

Once the entities in the text documents are tagged, *Snowball* can ignore unwanted entities (e.g., `PERSON`s), focus on occurrences of `LOCATION` and `ORGANIZATION` entities (Figure 5), and analyze the context that surrounds each pair of such entities to check if they are connected by the right words and hence match our patterns.

To define patterns precisely, *Snowball* could follow DIPRE's approach, and have a pattern consist of a `left`, a `middle`, and a `right` string. An occurrence of an `ORGANIZATION` and a `LOCATION` entity would be regarded as a match for a pattern if the text surrounding the entities matches the three strings in the pattern exactly. As we will see, this approach results in somewhat selective patterns (i.e., most of these patterns tend not to generate invalid tuples), yet it suffers from limited coverage (i.e., these patterns might not capture all instances of valid tuples). Hence, *Snowball* represents the context around the `ORGANIZATION` and `LOCATION` entities in the patterns in a more flexible way. As a result, minor variations such as an extra comma or a determiner will not stop us from matching contexts that are otherwise very close to our patterns. More specifically, *Snowball* represents the left, middle, and right "contexts" associated with a pattern analogously as how the vector-space model of information retrieval represents documents and queries [15, 14]. Thus, the `left`, `middle`, and `right` contexts are three vectors associating weights (i.e., numbers between 0 and 1) with terms (i.e., arbitrary strings of non-space characters). These weights indicate the importance of each term in the corresponding context. We describe how to compute these weight vectors later in this section.

Definition 1 *A* Snowball pattern *is a 5-tuple* <`left, tag1, middle, tag2, right`>*, where* `tag1` *and* `tag2` *are named-entity tags, and* `left`, `middle`*, and* `right` *are vectors associating weights with terms.*

An example of a *Snowball* 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," where the word "the" (left context) precedes a location (Irving), which is in turn followed by the strings "-" and "based" (middle context) and an organization. What appears to the right of the organization in the string is unimportant in this case, hence the empty right context in the pattern. Slight variations of the given string will also match the pattern to a smaller extent. (We introduce a notion of "degree of match" later in this section.) For example, a string "...she said. Redmond-based Microsoft reportedly..." will tend to match our example pattern, even when the location, Redmond, is not preceded by any of the terms in the left context (i.e., "the"). This extra flexibility results in better coverage of the patterns.

To match text portions with our 5-tuple representation of patterns, *Snowball* also associates a 5-tuple with each document portion that contains two named entities with the correct tag (i.e., `LOCATION` and `ORGANIZATION` in our scenario). After identifying two such entities in a string S, *Snowball* creates three weight vectors l_S, r_S, and m_S from S by analyzing the left, right, and middle contexts around the named entities, respectively. l_S has a non-zero weight for each term in the w-term window to the left of the leftmost named entity in S, for some predefined window size w. Similarly, r_S has a non-zero weight for each term in the w-term window to the right of the rightmost named entity in S. Finally, m_S has a non-zero weight for each term in between the two named entities in S. The weight of a term in each vector is a function of the frequency of the term in the corresponding context. These vectors are scaled so their norm is one. Finally, they are multiplied by a scaling factor to indicate each vector's relative importance. From our experiments with English-language documents, we have found the middle context as the most indicative of the relationship between the elements of the tuple. Hence we will typically assign the terms in the middle vector higher weights than the left and right vectors.

After extracting the 5-tuple representation of string S, *Snowball* matches it against the 5-tuple pattern by taking the inner product of the corresponding left, middle, and right vectors.

Definition 2 *The degree of match* $Match(t_P, t_S)$ *between two 5-tuples* $t_P = < l_P, t_1, m_P, t_2, r_P >$ *(with tags* t_1 *and* t_2*) and* $t_S = < l_S, t'_1, m_S, t'_2, r_S >$ *(with tags* t'_1 *and* t'_2*) is defined as:*

$$Match(t_P, t_S) = \begin{cases} l_P \cdot l_S + m_P \cdot m_S + r_P \cdot r_S & \mathit{if\ the\ tags\ match} \\ 0 & \mathit{otherwise} \end{cases}$$

```
The <ENAMEX TYPE=LOCATION>Armonk</ENAMEX>-based <ENAMEX TYPE= ORGANIZATION>IBM
</ENAMEX> has introduced a new line...
-------------------------------------------------------------------------------
<ENAMEX TYPE=ORGANIZATION>Intel</ENAMEX>, <ENAMEX TYPE= LOCATION>Santa Clara,
</ENAMEX>, cut prices of its Pentium...
```

Figure 5: Portions of a document where `LOCATION` **and** `ORGANIZATION` **entities occur near each other.**

In order to generate a pattern, *Snowball* groups occurrences of known tuples in documents, if the contexts surrounding the tuples are "similar enough." More precisely, *Snowball* generates a 5-tuple for each string where a seed tuple occurs, and then clusters these 5-tuples using a simple single-pass clustering algorithm [9], using the $Match$ function defined above to compute the similarity between the vectors, with minimum similarity threshold τ_{sim}. 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* pattern $<\bar{l_s}, t_1, \bar{m_s}, t_2, \bar{r_s}>$, which will be later used to find new tuples in the document collection. Figure 6 shows the complete algorithm for computing patterns using clustering. (We will explain Line (3) in Section 2.3.)

```
sub GeneratePatterns
foreach (< o_seed, ℓ_seed >, str) in Occurrences
(1)   t_S =< l_s, t_1, m_s, t_2, r_s > = makeOccurrence(str);
(2)   cluster best = FindClosestCluster(t_S, τ_sim);
      if(best)
        best.Add(t_S);
        best.UpdateCentroid(t_S);
        best.AddTuple(< o_seed, ℓ_seed >);
      else
        CreateNewCluster(< o_seed, ℓ_seed >, t_S);
(3)Patterns = FilterPatterns( Clusters, τ_sup);
   return Patterns;
```

Figure 6: Clustering algorithm for generating patterns from tuple occurrences in the text documents.

2.2 Generating Tuples

After generating patterns (Section 2.1), *Snowball* scans the collection to discover new tuples. The basic algorithm is outlined in Figure 7.

Snowball 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* generates the 5-tuple $t = < l_c, t_1, m_c, t_2, r_c >$. 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 of Section 2.1.

Each candidate tuple will then have a number of patterns that helped generate it, each with an associated degree of match. *Snowball* 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. This filtering process will become clear in the next section, where we address the crucial issue of how to evaluate candidate tuples and patterns.

```
sub GenerateTuples(Patterns)
  foreach text_segment in corpus
(1) {< o, ℓ >, < l_s, t_1, m_s, t_2, r_s >} =
      = CreateOccurrence(text_segment);
    T_C = < o, ℓ >;
    Sim_Best = 0;
    foreach p in Patterns
(2)   sim = Match(< l_s, t_1, m_s, t_2, r_s >, p);
      if (sim ≥ τ_sim)
(3)     UpdatePatternSelectivity(p, T_C);
        if(sim ≥ Sim_Best)
          Sim_Best = sim;
          P_Best = p;
    if(Sim_Best ≥ τ_sim)
      CandidateTuples[T_C].Patterns[P_Best] =
        = Sim_Best;
  return CandidateTuples;
```

Figure 7: Algorithm for extracting new tuples using a set of patterns.

2.3 Evaluating Patterns and Tuples

Generating good patterns is challenging. For example, we may generate a pattern <{}, ORGANIZATION, <``,'', 1>, LOCATION, {}> from text occurrences like "Intel, Santa Clara, announced..." This pattern will be matched by any string that includes an organization followed by a comma, followed by a location. Unfortunately, a sentence "It's a great time to invest in **Microsoft**, **New York**-based analyst Jane Smith said" will then generate a tuple <Microsoft, New York>, which would be incorrect because Microsoft's headquarters are in Redmond. In summary, the pattern above is not selective, since it might generate incorrect tuples. *Snowball* will try to identify such patterns and not trust them, and instead focus on other more selective patterns. Under our redundancy assumption that tuples occur in different contexts in our collection, *Snowball* can afford to not use the less selective pattern above and still be able to extract the tuple <Intel, Santa Clara> from our example in Section 1 through a different, more selective pattern. Estimating the *selectivity* of the patterns, so that we do not trust patterns that tend to generate wrong tuples, is one of the problems that we address in this section. We can weigh the *Snowball* patterns based on their selectivity, and trust the tuples that they generate accordingly. Thus, a pattern that is not selective will have a low weight. The tuples generated by such a pattern will be discarded, unless they are supported by selective patterns.

Organization	*Location of Headquarters*
3COM CORP	SANTA CLARA
3M	MINNEAPOLIS
AIR CHINA	BEIJING
FEDERAL EXPRESS CORP	MEMPHIS
FRUIT JELLIES	APPLE
MERRILL LYNCH & CO	NEW YORK
NETSCAPE	MOUNTAIN VIEW
NINTENDO CORP	TOKYO

Table 2: Some tuples discovered during *Snowball*'s first iteration.

The case for tuples is analogous. Consider for example the candidate tuples in Table 2, which were generated during *Snowball*'s first iteration. It is clear that not all of these tuples are valid. For example, the tuple `<Fruit Jellies, Apple>` is invalid, and was generated because Alembic incorrectly tagged "Apple" as a location and "Fruit Jellies" as an organization. So, if we use all of these tuples as the new seed tuples for the next *Snowball* iteration, we may generate extraneous patterns that in turn might result in even more wrong tuples in the next iteration. We have explored different pruning schemes to select the new seed tuples that are likely to be valid. We only keep tuples with high *confidence*. The confidence of the tuple is a function of the selectivity and the number of the patterns that generated it. Intuitively, the confidence of a tuple will be high if it is generated by several highly selective patterns.

The pattern and tuple evaluation is the key part of our system, and is responsible for most of the improvement over the DIPRE scheme. As an initial filter, we eliminate all patterns *supported* by fewer than τ_{sup} seed tuples (Step (3) in the algorithm in Figure 6). We experimentally evaluated alternative methods for defining τ_{sup} and concluded that a simple static value for τ_{sup} works well. In addition to the filter based on the number of seed tuples that generated the patterns, we compute the *selectivity* of each pattern in Step (3) of the algorithm in Figure 7. In that step, the call to function `UpdatePatternSelectivity` checks each candidate tuple $t = < o, \ell >$ generated by the pattern in question. If there is a high confidence tuple $t' = < o, \ell' >$ generated during an earlier iteration of the system for the same organization o as in t, then this function compares locations ℓ and ℓ'. If the two locations are the same, then the tuple t is considered a *positive* match for the pattern. Otherwise, the match is *negative*. Intuitively, the candidate tuple that a pattern generates for the "known" organizations should match the locations of these organizations. Otherwise, the confidence in this pattern will be low. 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 *Snowball* patterns for relations without such a single-attribute key is part of our future work (Section 6).

Definition 3 *The* confidence *of a pattern P is:*

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

where $P.positive$ is the number of positive matches for P and $P.negative$ is the number of negative matches.

As an example, consider the pattern P = `<ORGANIZATION>, <LOCATION>` referred to in the previous section. Assume that this pattern only matches the three lines of text below:

"**Exxon**, **Irving**, said"
"**Intel**, **Santa Clara**, cut prices"
"invest in **Microsoft**, **New York**-based analyst Jane Smith said"

The first two lines generate candidate tuples `<Exxon, Irving>` and `<Intel, Santa Clara >`, which we already knew from previous iterations of the system. The third line generates tuple `<Microsoft, New York>`. The location in this tuple conflicts with the location in tuple `<Microsoft, Redmond>`, hence this last line is considered a negative example. Then, pattern P has confidence $Conf(P) = \frac{2}{2+1} = 66\%$.

Our definition of confidence of a pattern above is only one among many possibilities. An alternative that we evaluate experimentally in Section 5 is to account for a pattern's coverage in addition to its selectivity. For this, we adopt a metric originally proposed by Riloff [12] to evaluate extraction patterns generated by the Autoslog-TS information extraction system according to the formula $RlogF(p) = relevance\ rate(p) \cdot \log_2(frequency(p))$. We can define $Conf_{RlogF}(p)$ of pattern p similarly.

Definition 4 *The* RlogF *confidence of pattern p is:*

$$Conf_{RlogF}(p) = \frac{p.positive}{(p.positive + p.negative)} \cdot \log_2(p.positive)$$

Pattern confidences are defined to have values between 0 and 1. Therefore, we normalize the $Conf_{RlogF}$ values by dividing them by the largest confidence value of any pattern.

Having scored the patterns, we are now able to evaluate the new candidate tuples. Recall that for each tuple we store the set of patterns that produced it, together with the measure of similarity between the context in which the tuple occurred, and the matching pattern. Consider a candidate tuple T and the set of patterns $P = \{P_i\}$ that were used to generate T. For simplicity assume that T matched each of the patterns P_i perfectly, i.e., with degree of match equal to one. Let us assume for the moment that we know the probability $Prob(P_i)$ with which each pattern P_i generates valid tuples. If these probabilities are independent of each other, then the probability that T is valid, $Prob(T)$, can be calculated as:

$$\begin{aligned} Prob(T) &= 1 - Prob(All\ Patterns\ Fired\ Incorrectly) \\ &= 1 - \prod_{i=0}^{|P|} (1 - Prob(P_i)) \end{aligned}$$

Our confidence metric $Conf(P_i)$ was designed to be a rough estimate of $Prob(P_i)$, the probability of pattern P_i generating a valid tuple. We also account for the cases where T has occurred in contexts that did not match our patterns perfectly. For this, we scale each $Conf(P_i)$ term by the degree of match of the corresponding pattern and context:

Definition 5 *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.

For example, suppose that we just generated a tuple `<Netscape, Mountain View>` using the patterns "`<ORGANIZATION>, <LOCATION>`" and "`<ORGANIZATION> of <LOCATION>`." These patterns have been found to have confidences of 0.5 and 0.6, which means that individually, these patterns are almost as likely to generate valid tuples as they are to generate invalid tuples. However, the confidence of the tuple that is generated by *both* of these patterns is:

$$Conf(T_{new}) = 1 - ((1 - 0.5) \cdot (1 - 0.6)) = 1 - 0.5 \cdot 0.4 = 0.8$$

Note that when we described the calculation of the pattern confidence, we ignored any confidence values from previous iterations of *Snowball*. To control the learning rate of the system, we set the new confidence of the pattern as:

$$Conf(P) = Conf_{new}(P) \cdot W_{update} + Conf_{old}(P) \cdot (1 - W_{update})$$

The parameter W_{update} can be used to control the speed of learning from new examples. If $W_{update} < 0.5$ then the system in effect trusts new examples less on each iteration, which will lead to more conservative patterns and have a damping effect. For our experiments we set $W_{update} = 0.5$.

Similarly, we often rediscover tuples that we have already extracted on previous iterations. In this case, we also set the new confidence of the tuple as:

$$Conf(T) = Conf_{new}(T) \cdot W_{update} + Conf_{old}(T) \cdot (1 - W_{update})$$

After determining the confidence of the candidate tuples using the definition above, *Snowball* discards all tuples with low confidence. These tuples could add noise into the pattern generation process, which would in turn introduce more invalid tuples, degrading the performance of the system. The set of tuples to use as the seed in the next *Snowball* iteration is then $Seed = \{T | Conf(T) > \tau_t\}$, where τ_t is some prespecified threshold.

For illustration purposes, Table 3 lists three representative patterns that *Snowball* extracted from the document collection that we describe in Section 4.1.

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

3 EVALUATION METHODOLOGY AND METRICS

The goal of *Snowball* is to extract as many valid tuples as possible from the text collection. As we have discussed, 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. Our system extracts tuples from all of the documents in the collection and combines them into one table. 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 [15, 14]. 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 tuples that appear in the collection D (Section 3.1). After identifying *Ideal*, we compare it against the tuples produced by the system, *Extracted*, using the adapted precision and recall metrics (Section 3.2).

3.1 Methodology for Creating the *Ideal* Set

For small text collections, we could inspect all documents manually and compile the *Ideal* table by hand. Unfortunately, this evaluation approach does not scale, and becomes infeasible for the kind of large collections over which *Snowball* is designed to operate. To address this problem, we start by considering a large, publicly available directory of organizations provided on the "Hoover's Online" web site[1]. Although the directory does not cover every organization there is, it is large enough for our purposes, covering over 13,000 mostly publicly traded corporations. From this well structured directory, we generate a table of organization-location pairs. Unfortunately, we cannot use this table as is, since some of the organizations in it might not occur at all in the text collection that we use in our experiments.

To determine the target set of tuples *Ideal* from the Hoover's-compiled table above, we need to keep only the tuples that have the organization mentioned together with their location in the collection. To find all such instances, we identify all the variations of each organization name in the Hoover's table as they may appear in the collection, and then check if the headquarters of the test organization are mentioned nearby.

For this task, we generate a list of all organization-location pairs

[1] `http://www.hoovers.com`

that occur in the same line of text in our collection. We then use Whirl [3], a research tool developed at AT&T Research Laboratories for integrating similar textual information, to match each organization name, as it occurs in the collection, to the organization in the Hoover's table. For example, "Microsoft," "Microsoft Corporation," and "Microsoft Corp." are all references to the same organization ("Microsoft"), and if the company's location ("Redmond"), is mentioned in the same line with *any* of variations of the organization name, the tuple `<Microsoft, Redmond>` should be counted as occurring in the collection and hence it will be included in the *Ideal* table.

3.2 The *Ideal* Metric

Now that we have created the *Ideal* table, we can use it to evaluate the quality of the *Snowball* output, the *Extracted* table. If the initial directory of organizations from Hoover's contained all possible organizations, then we could just measure what fraction of the tuples in *Extracted* are in *Ideal* (precision) and what fraction of the tuples in *Ideal* are in *Extracted* (recall). Unfortunately, a large collection will contain many more tuples that are contained in any single manually compiled directory. (In our estimate, our training collection contains more then 80,000 valid organization-location tuples.) If we just calculated precision as above, all the valid tuples extracted by *Snowball*, which are not contained in our *Ideal* set, will unfairly lower the reported value of precision for the system.

To address this problem we create a new table, *Join*, as the join of tables *Ideal* and *Extracted* on a unique key (i.e., organization). For each tuple $T =< o, \ell >$ in the *Ideal* table, we find a matching tuple $T' =< o', \ell' >$ in the *Extracted* table (if any), such that $o \simeq o'$. (We describe how to deal with variations in the organization names in Section 3.3.) Using these values, we now create a new tuple $< o, \ell, \ell' >$ and include it in the *Join* table.

Given the table *Ideal* and the *Join* table that we have just created, we can define recall and precision more formally. We define *Recall* as:

$$Recall = \frac{\sum_{i=0}^{|Join|} [\ell_i = \ell'_i]}{|Ideal|} \cdot 100\% \tag{1}$$

where $[\ell_i = \ell'_i]$ is Iverson notation that is equal to 1 if the test value ℓ_i matches the extracted value ℓ'_i, and 0 otherwise. Thus, the sum in the numerator is the number of *correct* tuples of the *Ideal* set that we extracted, which we divide by the size of the *Ideal* table to obtain our recall. Similarly, we define *Precision* as:

$$Precision = \frac{\sum_{i=0}^{|Join|} [\ell_i = \ell'_i]}{|Join|} \cdot 100\% \tag{2}$$

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.) This evaluation method is time consuming, potentially error-prone, and will have to be redone for each new collection. Indeed, our system is specifically designed for large collections, where it is not possible for a human to manually examine any significant portion of the collection. In this sense, the sampling technique is inferior to the *Ideal* metric that we proposed. However, by sampling the extracted table we can detect invalid tuples whose organization is not mentioned in the Hoover's directory that we used to determine *Ideal*, for example. Similarly, we can detect invalid tuples that result from named-entity tagging errors. Hence, we also report precision estimates using sampling in Section 5.

3.3 Matching Location and Organization Names

A problem with calculating the *Ideal* metric above is introduced by the proliferation of variants of organization names. We combine all variations into one, by using a *self-join* of the *Extracted* table with itself. We use Whirl to match the organization names to each other, to create the table *Extracted'*. We pick an arbitrary variation of the organization name, o_s, as the "standard," and pick a location, ℓ_{max}, from the set of matching organization-location tuples, with the highest confidence value. We then insert the tuple $< o_s, \ell_{max} >$ into the *Extracted'* table.

Similarly, we need to decide when the location extracted for an organization is correct. For example, our system might conclude that California is the location of the headquarters of Intel. This answer is correct, although not as specific as could be. Our scoring system will in fact consider a tuple `<Intel, California>` as correct. Specifically, we consider tuple $< o, \ell >$ to be valid if (a) organization o is based in the U.S. and ℓ is the city or state where o's headquarters are based; or (b) organization o is based outside of U.S. and ℓ is the city or country where o's headquarters are based.

4 EXPERIMENTAL SETTING

We describe the training and text collections that we used for experiments in Section 4.1. We also enumerate the different extraction methods that we compare experimentally (Section 4.2).

4.1 Training and Test Collections

Our experiments use large collections of real newspapers from the North American News Text Corpus, available from LDC [2]. This corpus includes articles from 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 (Table 4).

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 5 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,

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

Collection	*Document Source*	*Documents*	*Year*
Training	The New York Times	96,000	1996
	The Wall Street Journal	56,000	1996
	Los Angeles Times	26,000	1996
Test	The New York Times	44,000	1995
	The Wall Street Journal	43,000	1995
	Los Angeles Times	35,000	1995
	Los Angeles Times	20,000	1997

Table 4: The document collections used for experiments.

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.

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

Table 5: Occurrence statistics of the test tuples in the experiment collections.

The first row of Table 5, corresponding to zero occurrences, deserves further explanation. If we wanted to evaluate the performance of our system on *all* the organizations that were 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.

4.2 Evaluating Alternative Techniques

We compared *Snowball* with two other techniques, the *Baseline* method and our implementation of the DIPRE method. These two methods require minimal or no training input from the user, and hence are comparable with *Snowball* in this respect. In contrast, state-of-the-art information extraction systems require substantial manual labor to train the system, or to create a hand-tagged training corpus.

The first method, *Baseline*, is based purely on the frequency of co-occurrence of the organization and the location. Specifically, *Baseline* reports the location that co-occurs in the same line with each organization most often as the headquarters for this organization. *Baseline* uses as input lines of text in the collection, tagged with the Alembic named entity tagger, and creates an index of the organizations and locations that occur in the same line. Then, *Baseline* simply selects the most frequent location for each organization. Despite its simplicity, the method works surprisingly well in this setting.

The second method is our implementation of DIPRE, which we described in Section 1. We did not have access to the original implementation, so we had to reimplement it. After testing our implementation on the "author-title" task, which is to the best of our knowledge the only application of the DIPRE method reported in the literature[2], we had to make some modifications, motivated by the nature of our collections. The original DIPRE implementation, uses *urlprefix* to restrict pattern generation and application. Since all of our documents came from just three sources, DIPRE was not able to use this feature, which was originally intended to generate patterns that would apply only to the documents with the URLs that match the *urlprefix* of each pattern. The second, and more important, modification had to do with the fact that DIPRE was designed to extract tuples from HTML-marked data, which is inherently more structured than the plain text that we used for experiments. Without HTML tags, DIPRE could not find occurrences of the seed tuples in plain text that were surrounded by exactly the same, non-empty, left, middle, and right contexts. To solve this problem, we used the named entity tagger to pre-tag the input to DIPRE. This way, all the organizations and locations were consistently surrounded by named entity tags. DIPRE could incorporate these tags as part of the surrounding context, and generate patterns that take advantage of these named-entity tags. Because the original DIPRE implementation had very low recall (having no access to the named-entity tags), the results we report for DIPRE are *not* for the original DIPRE implementation, but are rather results achieved by using the DIPRE method together with named-entity tags.

4.3 *Snowball*

As we described in Section 2, the basic *Snowball* process requires finding occurrences of the seed tuples in the corpus. For efficiency, we have indexed our collections using the Glimpse search engine [11], which supports boolean queries. Our scheme is to issue a boolean "AND" query for each seed tuple, requiring all elements of the tuple to be present in the same text segment. For example, a query "`Microsoft AND Redmond`" will be issued to find all the contexts in which the seed tuple `<Microsoft, Redmond>` appears in the collection. In our experiments we required the tuple elements to occur in the same line of text, but our approach can be used to retrieve occurrences of the example tuples within arbitrary text segments.

Once these example occurrences are retrieved, *Snowball* can act differently based on a number of parameters. We have attempted to determine the best combination of parameters by running the *Snowball* system on the training corpus. Some of the parameters we experimented with include:

- **Use of Punctuation**: We experimented with discarding punctuation and other non-alphanumeric characters from the contexts surrounding the entities. Our hypothesis was that punctuation may just add noise but carry little content to help extract tuples. We report results for *Snowball* and *Snowball-Plain*, where *Snowball* uses punctuation, and *Snowball-Plain* discards it.
- **Choice of Pattern Scoring Strategies:** We tried variations

on the basic framework for weighing patterns, as described in Section 2, with or without using the *RlogF* metric described in [12]. We will refer to the strategies that use the *RlogF* metric as *"RlogF-*"*. Additionally, we can normalize both patterns and tuples by dividing by the largest value of each. The normalized strategies will be referred to as *"*-Norm"*, and the not normalized ones as *"Raw"*. Thus, we have a list of four strategies: *Raw*, *RlogF*, *Norm*, and *RlogF-Norm*. The choice of the weighting strategy can have a significant effect on the quality of new seed tuples that we use to start the next iteration of the system.

- **Choice of Pattern Similarity Threshold** (τ_{sim}): This threshold controls how flexible the patterns are, both during the pattern generation stage (i.e., how similar the occurrences of the example tuples have to be in order to be grouped into one cluster), as well as during the tuple extraction stage, where τ_{sim} controls the minimum similarity between the context surrounding the potential tuple and a pattern, determining whether a tuple will be generated.
- **Choice of Tuple Confidence Threshold** (τ_t): This threshold determines the minimum confidence a tuple must have to be included in the seed set to start the next iteration.

5 EXPERIMENTAL RESULTS

In this section, we experimentally compare the performance of *Snowball* and the alternative techniques that we discussed in Section 4.2. Our experiments use the training and test collections of Section 4.1. In Section 5.1 we use the training collection to determine the test settings for the *Snowball* parameters of Section 4.3. Then, in Section 5.2 we compare the performance of *Snowball*, DIPRE, and *Baseline* on the test collection.

5.1 Training Phase

Figure 8 summarizes the experimental results on the training collection. As discussed in Section 3, we consider tuples in *Ideal* in different groups, based on their number of occurrences in the collection. (DIPRE and *Snowball* assume a scenario where tuples occur redundantly in the collection.) Figure 8 (a) reports the average recall of the techniques as a function of the minimum number of times that a tuple must appear in the training collection in order to be included in *Ideal*. For example, if we focus only on tuples that occur two or more times in the training collection and define *Ideal* accordingly, *Baseline* achieves an average *Recall* of around 70% while *Snowball*'s value is highest at 80% (Figure 8 (a)). From this figure, we can see that the average recall of DIPRE and *Snowball* improves as we require tuples to occur more times in the collection. This is consistent with the design principles underlying DIPRE and *Snowball* tailored to collections with redundancy. Figure 8 (a) also shows that it is important to use punctuation in the extraction process: the recall of *Snowball* is more than twice as high as that of *Snowball-Plain*. Figure 8 (b) reports the average precision values for the various techniques.

We ran experiments on the training collection to determine optimal values for τ_{sim}, τ_t, τ_{sup}, and the optimal weight distribution W_{left}, W_{middle}, and W_{right} for the left, middle, and right context vectors of each pattern. Among the pattern scoring strategies, *RlogF-Norm* performed the best in terms of precision and recall, producing enough new seed tuples to allow *Snowball* to sustain an acceptable rate of acquiring new patterns. In Section 5.2 we report results on the tables extracted after one iteration of the various techniques. As we will see, the performance of DIPRE tends to deteriorate after one iteration, while that of *Snowball* remains stable.

Parameter	*Value*	*Description*
τ_{sim}	0.6	minimum degree of match (Section 2.1)
τ_t	0.8	minimum tuple confidence (Section 2.3)
τ_{sup}	2	minimum pattern support (Section 2.1)
I_{max}	3	number of iterations of *Snowball*
W_{middle}	0.6	weight for the *middle* context (Section 2.1)
W_{left}	0.2	weight for the *left* context (Section 2.1)
W_{right}	0.2	weight for the *right* context (Section 2.1)

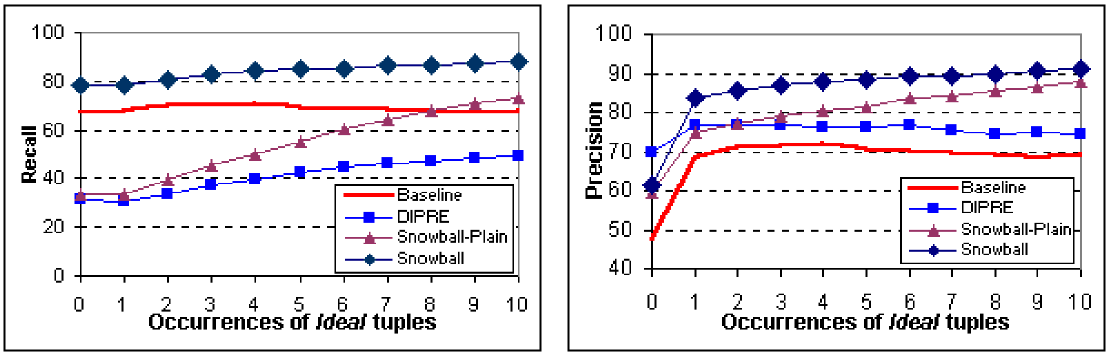
Table 6: Parameter values used for evaluating *Snowball* on the test collection.

5.2 Test Phase

As we discussed, the only input to the *Snowball* system 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 the *Snowball* system using the operational parameters listed in Table 6, which worked best on the training collection. The *RlogF-Norm* metric was used to score patterns for generating the set of seed tuples for the next iteration. The results are reported in Figure 9. The plot shows the performance of the systems as we attempt to extract test tuples that are mentioned more times in the corpus. As we can see, *Snowball* 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* on the 0-occurrence level (72% vs. 67% for *Snowball*), *Snowball* has at all occurrence levels significantly higher recall than DIPRE and *Baseline* do. This is consistent with the training results.

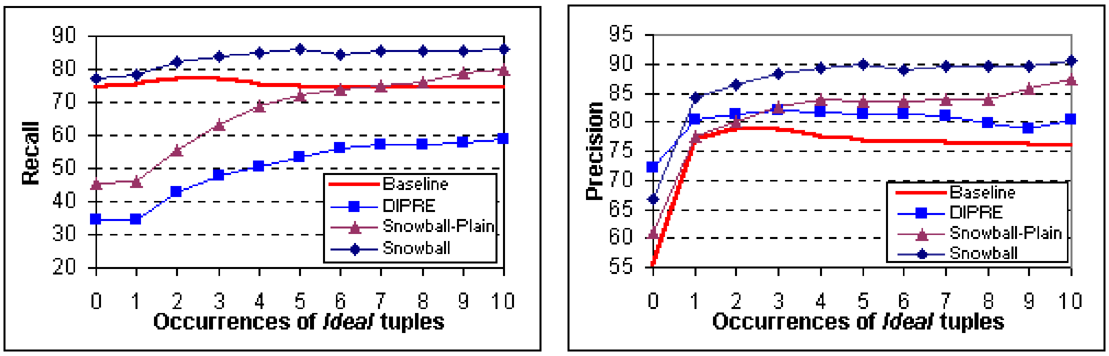
We attempted to determine if we could remedy DIPRE's low recall by running it for more iterations. Unfortunately, after the first iteration both recall and precision decreased. Figure 10 also demonstrates that *Snowball* is *stable* in a sense that it converges to some reasonable values, while DIPRE quickly diverges. The reason for DIPRE's behavior is that DIPRE has no way of selecting reliable tuples as the seed for its next iteration, while *Snowball* takes advantage of the tuple confidence metric for this. We report data for only two iterations for the *Snowball-Plain* because it converged after iteration 2 (i.e., it did not produce any new seed tuples).

As discussed in Section 3.2, we complete our evaluation of the precision of the extraction systems by manually examining a sample of their output. For this, we randomly selected 100 tuples from each of the extracted tables, and manually checked whether each of these tuples was a valid organization-location pair or not. We separate the errors into three categories: errors



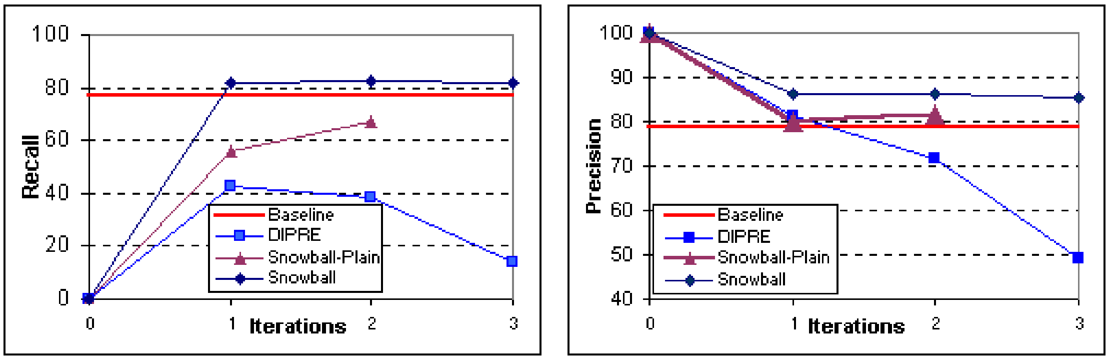
(a) (b)

Figure 8: Recall (a) and precision (b) of *Baseline*, DIPRE, and *Snowball* (training collection; Table 6 parameter settings).



(a) (b)

Figure 9: Recall (a) and precision (b) of *Baseline*, DIPRE, *Snowball* and *Snowball-Plain* (test collection).



(a) (b)

Figure 10: Recall (a) and precision (b) of *Baseline*, DIPRE, *Snowball*, and *Snowball-Plain* as a function of the number of iterations (*Ideal* tuples with occurrence $\geq$ 2; test collection).

due to mistagging a location and assigning it to a valid organization ("Location" error), errors due to including a non-existing organization ("Organization" error), and errors due to deducing an incorrect relationship between a valid organization and location ("Relationship" error). These different types of errors are significant because they highlight different "culprits": the "Location" and "Organization" errors could be prevented if we had a perfect named-entity tagger, whereas the "Relationship" errors are wholly the extraction system's fault (Table 7).

The last column in Table 7 (P_{Ideal}) is precision, calculated by ignoring the "Organization" errors and computing the fraction of valid organizations for which a correct location was found. These values, in effect, correspond to the values of precision we would have calculated if our *Ideal* table included all the valid organizations in the random samples. These figures, however, do not capture invalid tuples generated due to improper tagging of a string as an organization. From our manual inspection of a random sample of 100 tuples from each extracted table, we observed that DIPRE's sample contained 74 correct tuples and 26 incorrect ones. *Snowball*'s sample contained 52 correct tuples and 48 incorrect tuples, while *Baseline* has a majority of incorrect tuples (25 vs. 75). As we can see from the breakup of the errors in the table, virtually all of *Snowball*'s errors are tagging related (i.e., "Location" or "Organization" errors). If we prune the *Snowball*'s final output to only include those tuples t with $Conf(t) \geq 0.8 = \tau_t$, then most of these spurious tuples disappear. In effect, from a random sample of 100 tuples from this pruned table, 93 tuples are valid and only 7 are invalid. Furthermore, none of the invalid tuples are due to "Relationship" errors (third row of Table 7).

So far, the results that we have reported for *Snowball* are based on a table that contains all the "candidate" tuples generated during *Snowball*'s last iteration. As we saw in Table 7, the precision of *Snowball*'s answer varies dramatically if we prune this table using the tuple confidence threshold τ_t. Of course, this last-step pruning is likely to result in lower recall values. In Figure 11 we explore the tradeoff between precision and recall for different values of this last-step pruning threshold. A user who is interested in high-precision tables might want to use high values for this threshold, while a user who is interested in high-recall tables might want to use lower values of the threshold. For example, by setting $\tau_t = 0.4$ and filtering the *Extracted* table accordingly, we estimate the absolute precision of *Snowball*'s output to be 76% and recall to be 45%, both of which are higher than the corresponding metrics of DIPRE's output.

In summary, both *Snowball* and DIPRE exhibit significantly higher precision than *Baseline*. In effect, *Baseline* tends to generate many tuples, which results in high recall at the expense of low precision. *Snowball*'s recall is at least as high as that of *Baseline* for most of the tests, with higher precision values. *Snowball*'s recall is generally higher than DIPRE's, while the precision of both techniques is comparable.

6 CONCLUSIONS AND FUTURE WORK

This paper presents *Snowball*, a system for extracting relations from large collections of plain-text documents that requires minimal training for each new scenario. We introduced novel strategies for generating extraction patterns for *Snowball*, as well as techniques for evaluating the quality of the patterns and tuples generated at each step of the extraction process. Our large-scale experimental evaluation of our system shows that the new techniques produce high-quality tables, according to the scalable evaluation methodology that we introduce in this paper. Our experiments involved over 300,000 newspaper articles.

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. In effect, state-of-the-art taggers rely on textual clues from the text surrounding each entity, which may be absent in HTML documents that rely on visual formatting to convey information, for example. Handling arbitrary HTML documents is an important part of our future work. On a related note, 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 so that they learn 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, a crucial open problem is how to generalize our tuple and pattern evaluation strategy of Section 2.3 so that it does not rely on an attribute being a key for the relation.

ACKNOWLEDGMENTS

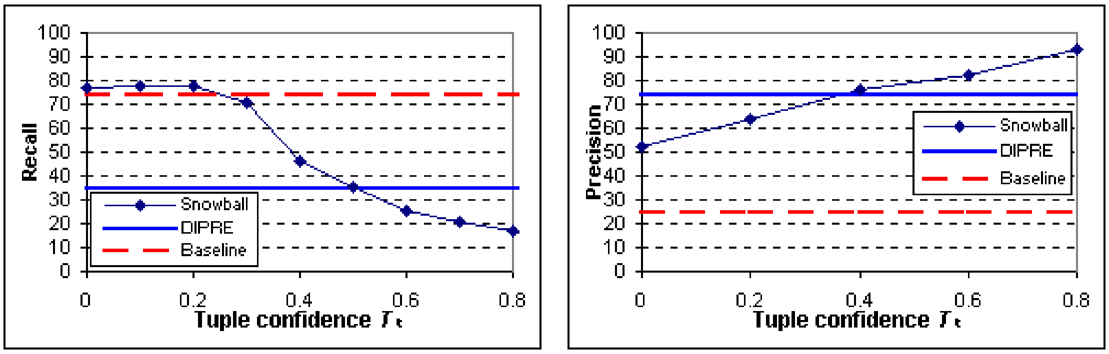
We thank Ralph Grishman and Vasilis Hatzivassiloglou for their helpful comments, and Eleazar Eskin for many fruitful discussions.

REFERENCES

1. Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 1998 Conference on Computational Learning Theory*, 1998.
2. 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.
3. 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.
4. 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.
5. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell,

			Type of Error			
	Correct	*Incorrect*	*Location*	*Organization*	*Relationship*	P_{Ideal}
DIPRE	74	26	3	18	5	90%
Snowball (all tuples)	52	48	6	41	1	88%
Snowball ($\tau_t = 0.8$)	93	7	3	4	0	96%
Baseline	25	75	8	62	5	66%

Table 7: Manually computed precision estimate, derived from a random sample of 100 tuples from each extracted table.



(a) (b)

Figure 11: Recall (a) and estimated precision (b) as a function of the threshold τ_t used for the last-step pruning of the *Snowball* tables (*Ideal* tuples with occurrence ≥ 1; test collection).

K. Nigam, and S. Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, 1999.

6. 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.

7. Scott Deerwester, Susan Dumais, George Furnas, Thomas Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of American Society for Information Science*, 1990.

8. 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.

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

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

11. Udi Manber and Sun Wu. Glimpse: A tool to search through entire file systems. In *Proceedings of the 1994 Winter USENIX Conference*, January 1994.

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

13. 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.

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

15. Gerard Salton and Michael J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1983.

16. Roman Yangarber and Ralph Grishman. NYU: Description of the Proteus/PET system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*. Morgan Kaufman, 1998.

17. 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.