

Large-Scale Extraction and Use of Knowledge From Text

Peter Clark and Phil Harrison

Boeing Phantom Works

The Boeing Company

Seattle, WA 98124

`peter.e.clark@boeing.com`

`philip.harrison@boeing.com`

Abstract

A large amount of empirically derived world knowledge is essential for many language-processing tasks, to create expectations that can help assess plausibility and guide disambiguation. Following Schubert (2002), we present our work on creating a large database of "tuples" from the output of a parser, thus implicitly capturing simple world knowledge expectations, and then utilizing it for two tasks, namely improving parsing and improving the plausibility assessment of paraphrase rules used in textual entailment. With regard to parsing, our goal is to improve the performance of a large-scale phrase structure parser/grammar of English through a method that does not depend on the need to hand-annotate a large number of sentences beyond those available from existing treebanks. With regard to the plausibility of paraphrase rules, our goal is to use the tuples to assess the plausibility of instantiated rules used in textual entailment, and thus improve the confidence rankings of the entailments that used those rules. In preliminary experiments with these two applications, the use of the tuple data improved performance.

1 Introduction

A large amount of empirically derived world knowledge is essential for many language-processing tasks, to create expectations that can help assess plausibility and guide disambiguation.

While a number of researchers have extracted specific data from text, e.g., bigrams (Keller and Lapata, 2003), collocations (Lin, 1998), we have followed work by Schubert (2002) who proposed that it might be possible to extract a variety of simple types of general world knowledge from text (e.g., "airplanes can fly", "houses can have roofs", "doors can be opened") from abstracted parse structures, from which various statistics can then be computed. In this paper we present our work on generating a large database of "tuples" encoding this kind of general knowledge, following similar techniques. We also present and evaluate two applications of this database, the first to improving parsing and the second to improving textual entailment. The first application is similar in style of Lin's use of dependency triples to bias the MiniPar parser (described in Lin and Pantel, 2001), and essentially biases the parser to prefer structures which have been seen often in a large corpus, as reflected in the tuple database. The second application is, to our knowledge, a novel combination of world knowledge (as encoded in the tuple database) and inference rules for textual entailment. In this second application, although an inference rule may be generally reliable (e.g., IF X shoots Y then X injures Y), a specific instantiation of it may be unlikely or nonsensical (eg., IF a man shoots a gun THEN the man injures the gun). We use expectations in the tuple database to assess the unusualness of an instantiated rule, and hence revise the confidence associated with the inference and hence entailment decision. In preliminary experiments with both these applications, performance improved when using the tuple database. This pro-

vides some empirical validation of the utility of the tuple database, and expands on Schubert et al's evaluation of their own work which was based solely on human inspection of the acquired knowledge (Schubert and Tong, 2003).

2 Generating the Tuple Database

2.1 Generating Tuples

Our work follows Schubert's conjecture that "there is a largely untapped source of general knowledge in texts, lying at the level beneath the explicit assertional content." (Schubert 2002). This conjecture is based on the observation that a sentence such as

"The camouflaged helicopter landed near the embassy"

not only describes a specific event, but also implicitly reveals more general knowledge, such as: helicopters can land, and helicopters can be camouflaged. By parsing sentences and abstracting from the resulting syntactic structure (e.g., dropping modifiers), large numbers of general propositions can be obtained. While many will be erroneous (e.g., due to misparses), statistically higher frequency propositions can be considered more reliable, and hence statistics used to assign a confidence value to each proposition. Schubert and Tong (2003) followed this approach and showed that many propositions obtained this way were considered reasonable by a panel of human reviewers. While the approach has some overlap with work on constructing specific databases such as two-word collocations (Lin 1998) and selectional preferences (Resnik, 1997), it differs in that it collects a variety of larger structures, and makes (and assesses) claims about the resulting resource as a source of world knowledge in its own right. Banko and Etzioni (2007) have performed similar work in the TextRunner system, running over more text but using lighter-weight linguistic analysis (part-of-speech tagging and noun phrase chunking rather than full parsing), producing slightly less normalized and less structured data, but then being pulled together into simple frame structures in a subsequent processing step. Our work follows Schubert's general knowledge extraction methodology, differing in some of the details and producing a large resource of 53 million tuples.

A tuple is a data structure containing a piece of syntactic information extracted from text, produced during parsing by a set of tuple generation rules attached to grammar rules. Syntactically, each tuple can be thought of as a fragment of the parse tree that retains head words or embedded tuples from a parse subtree, for example "The camouflaged helicopter landed" produces the tuples (AN "camouflaged" "helicopter") and (S "helicopter" "land"). Semantically, each tuple denotes an example of the combination that exists in the world (as reflected by the corpus), and thus can be thought of as a statement of possibility, e.g., "helicopters can be camouflaged" and "helicopters can land". Extracting tuples is similar to extracting dependency triples between word pair collocations (e.g., Lin 1998), but working with a more expressive data structure that can relate more than 2 words together.

As a data structure, each tuple consists of a tuple type symbol, followed by arguments that typically are (root form) words from the input sentence, or, in some cases, other tuples. For example, the sentence "The lazy men from the city walked to the fancy store" produces four tuples:

```
;;; tuples for "The lazy men from the city walked to
the fancy store"
(S "man" "walk" (PP "to" "store"))
(NPN "man" "from" "city")
(AN "lazy" "man")
(AN "fancy" "store")
```

which (following Schubert, 2002) can be interpreted as: "men can walk to stores", "men can be from cities", "men can be lazy", and "stores can be fancy".

Our system generates eight types of tuple, listed below:

```
(S subj verb [obj] [(PP prep pobj)*]) ; sentential
(AN adj noun) ; adjective-noun
(NPN noun prep noun) ; noun-prep-noun
(NN noun noun) ; noun-noun
(QN (quantity unit) noun) ; quantity noun
(S-ADJ noun adj [(PP prep pobj)*]);sentential adj
(NV subj verb) ; noun-verb, derived from S tuples
(VN verb obj) ; verb-noun, derived from S tuples
```

where the arguments are either words, (**PN** *word*) for proper nouns, NIL if absent, or an entire S structure for sentential complements. Root forms (e.g. "man") replace the original plural and inflected verb forms. **S** tuples contain the (head word of the) subject, verb, and, if present, the object and PP tuples associated with PP attachment to the verb. There can also be other **S** tuples from sentential complements. **AN** tuples encode adjective-noun modification. Examples of the latter two tuple types are (**QN** "10" "day") "visit") and (**S-ADJ** "car" "red"). **NV** and **VN** tuples are derivatives of **S** tuples, containing just the subject-verb and verb-object pairs. An example from real data is:

;;; Tuples for "As Curran has forcefully shown, advertising also plays a part in shaping a newspaper."

(**S** (**PN** "Curran") "show")
 (**S** "advertising" "play" "part"
 (**PP** "in" (**S** NIL "shape" "newspaper")))
 (**S** NIL "shape" "newspaper")

Note that the tuple generator does not capture all grammatical elements (not all grammar rules have tuple generation), for example adverbs are not currently captured in the tuples. Also note that **S** tuples that are derived from gerundive NPs (from -ing verb phrases) can appear in the object position of other tuple types.

In addition to simply extracting head words and dropping modifiers, the tuple generator will perform additional transformations, for example it restores head words of fronted wh-noun phrases, as illustrated below:

;;; Tuples for "Which book did John bring?"
 (**S** (**PN** "John") "bring" "book")

Coordination structures are handled by computing all tuple combinations over the head words from each coordinated phrase, for example:

;;; Tuples for "The men and women ate beans and rice"
 (**S** "man" "eat" "bean")
 (**S** "woman" "eat" "bean")
 (**S** "man" "eat" "rice")
 (**S** "woman" "eat" "rice")

2.2 Generating the Database

We ran the tuple generator over 2 text corpora, namely the Reuters corpus¹ and the British National Corpus (BNC)², producing 53 million tuples in about 2 CPU months. Counts for the different types of tuple are, in millions: 20.0 (S), 14.6 (AN), 11.3 (NPN), 4.9 (NN), 0.4 (QN) and 1.7 (S-ADJ). (also VN and NV tuples are derivatives of S tuples). The large number of tuples allows us to also compute frequencies of particular tuples and tuple patterns, as a measure of their likelihood that they will occur in text (and by extrapolation, the world the text is describing). As well as supplying pairwise collocations, the database can generate additional information such as likely PP attachments to verbs, likely noun modifiers, etc.

2.3 Using the Database

To measure the "strength" of a tuple, i.e., how much the observed frequency indicates a genuine association between the words rather than one that could have arisen by chance, we use a Mutual Information measure (following others, e.g., Alshawi and Carter, 1994; Lin 2001). The mutual information of a triple, e.g., (**NN** $w_1 w_2$), compares the actual and expected probabilities of the triple if w_1 and w_2 were chosen at random:

$$MI = \log \frac{p(NN w_1 w_2)}{p(NN w_1 *) \cdot p(NN * w_2)}$$

where $p(NN w_1 w_2)$ is the probability of an NN triple being (**NN** $w_1 w_2$), $p(NN w_1 *)$ is the probability of an NN triple having w_1 as its first argument, and $p(NN * w_2)$ is the probability of an NN triple having w_2 as its second argument. Following Lin (2001), $p(NN w_1 w_2)$ is measured from frequencies in the tuple database using the formula:

$$p(NN w_1 w_2) = \frac{f(NN w_1 w_2) - c}{f(NN **)}$$

where $f(NN w_1 w_2)$ is the frequency of (**NN** $w_1 w_2$) tuples in the database, $f(NN **)$ is the total number of NN tuples, and c is a constant (following Lin, we also use 0.95) to reduce problems of over-estimation for rare, observed events. We also constrain mutual information to be ≥ 0 (i.e., discount

¹ <http://trec.nist.gov/data/reuters/reuters.html>

² <http://www.natcorp.ox.ac.uk/>

evidence of negatively associated words) to reduce problems of under-estimation for rare, unobserved events. $p(NN w_1 *)$ and $p(NN * w_2)$ are computed via:

$$p(NN w_1 *) = p(w_1) \cdot f(NN * *)$$
$$p(NN * w_2) = p(w_2) \cdot f(NN * *)$$

where $p(w_1)$, $p(w_2)$ comes from a word frequency database³. Similar computations are used to compute Mutual Information for other types of tuples.

3 Using Tuples

3.1 Improving parsing

Method

There are many applications using corpus statistics to guide parsing; while the majority (e.g., Collins, 1999, Charniak, 2000) use supervised training data, e.g., the Penn Treebank (Marcus et al., 1993), a few use unsupervised data, as is our goal here. For example, Ratnaparkhi (1998) acquires (verb,prep,noun) and (noun,prep,noun) tuples from unambiguous attachments found in a training corpus, then uses them to guide ambiguous attachments. Similarly, the MiniPar parser (as described in Lin and Pantel, 2001) uses dependency triple (collocation) frequencies found from parsing a 1GB newswire corpus to guide future parsing. We take a similar approach here, except we use the full range of tuple structures described earlier to provide statistical bias.

Our parser (Harrison and Maxwell, 1986) is a hand-built, broad coverage phrase structure parser that has been used successfully since the early 1990s in a commercial grammar checking application. Parse selection utilizes a set of hand crafted cost functions attached to the grammar rules that return a small positive or negative number (negative is favorable, positive is unfavorable) for each node. The system selects the tree with the minimum total cost as the preferred parse. To take account of tuples, the cost functions have been modified to assess the "cost" of the tuples associated with each node -- that is, for nodes in the parse tree which would produce a tuple in tuple generation mode, the parser looks in the tuple database to find how many such tuples there are, and adjusts the cost function accordingly based on the number found, reflecting that this part of the parse

is "more likely" if it has been seen often before. There are many possible ways to do this and our initial version is very simple. The values of the costing function in the parser typically vary between -2 ("moderately strong" preference) and 0 (no preference) (in addition, positive values are used to express negative preference). Similarly, the Mutual Information (MI) values of the tuples typically lie between 10 (high) and 0 (no information). We thus assign the "cost" of the tuples:

$$\text{Cost}(\text{tuple}) = -\text{MI}(\text{tuple})/5$$

This provides a simple linear mapping between the MI values and the cost values. We plan to experiment with alternative mapping functions in future.

Before the main run to generate the full tuple database itself, we seeded the database with a small set, numbering in the hundreds, of hand-written tuples. The parser was then used to parse the Reuters and BNC database and generate the full tuple database. In future we plan to regenerate the database using the original database as a guide, to obtain better parses (and hence better tuples) than the first generation, thus bootstrapping the process.

Examples

An example of how the tuple database can improve parsing, the sentence:

"Worn parts of old dresses were replaced with contrasting fabric and unfashionable outer and under garments were unpicked and remade in more fashionable styles."

was, without tuples, incorrectly parsed as the coordination of two sentences, the first ending with the word "outer", i.e., "(worn...outer) and (under...styles)". Also, the word "contrasting" was incorrectly analyzed as a verb. With tuples, the tuple database contains support for (i.e., examples of) the tuple (**AN** "outer" "garment"), leading the parser to construct two sentences with the first correctly ending at the word "fabric", i.e., "(worn...fabric) and (unfashionable...styles)". Also, the tuple (**AN** "contrasting" "fabric") in the database caused "contrasting" to be analyzed as an adjective.

As a second example, the compound noun in the sentence:

"World zinc metal production fell by 0.3 percent"

³ <http://www.kilgarriff.co.uk/bnc-readme.html>

is, without the tuples, misbracketed as "(((world zinc metal) production)", while with tuples is more correctly bracketed as "(world (zinc (metal production)))" because the tuple database contains no tuples of the form (NN "world" "zinc") or (NN "world" "metal"), but it does contain 7 instances of (NN "world" "production"), 44 instances of (NN "zinc" "metal"), 65 instances of (NN "zinc" "production"), and 12 instances of (NN "metal" "production").

Evaluation

We evaluated the parser without and with the tuple database, to assess whether the knowledge in the tuples improved parsing. For this evaluation, sixty sentences were randomly selected from the BNC, subject to two constraints: The sentences were between 20 and 30 words in length and the correct parse was somewhere in the parse forest built by the parser. (We did not want to confound the problem of grammar deficiencies with the issue of tuple utility.) The correct parse for each sentence was specified by manual sentence bracketing. Two sets of “automated” parse trees were constructed by using first the initial version of the parser and then the version utilizing the tuple database.

To compute evaluation scores, the automated trees were compared, node by node, with the corresponding manual parse. Each node was characterized as a triple consisting of its part of speech, the index of the first word “covered” by the node, and the index of the last word covered. If a triple (node) from an automated tree was also in the manual tree, then it was counted as *Correct* for the automated tree. If a node was in the manual tree, but not in an automated tree, it was *Missing* with respect to that automated tree. Finally, if a node was in an automated tree, but not in the manual tree, it was counted an *Incorrect* for that automated tree. In this way we computed values, for each tree, of the sums *Correct*, *Missing*, and *Incorrect*. We then defined for each tree:

$$\text{Recall} = \text{Correct} / (\text{Correct} + \text{Missing})$$

$$\text{Precision} = \text{Correct} / (\text{Correct} + \text{Incorrect})$$

We also aggregated the numbers of correct, incorrect, and missing phrases over the two sets of automated trees to provide aggregated recall and precision scores for the each set of parses. The aggregated results were as follows:

	Miss-ing	Cor-rect	Incor-rect	Recall	Prec-ision
No tuples	330	3379	360	0.911	0.904
Tuples	271	3438	278	0.927	0.925

Overall for the 60 sentences:

- For 18 sentences recall increased
- For 30 sentences recall was unchanged
- For 12 sentences recall decreased

Although the sample size is relatively small, these results are encouraging because overall performance improved. It is also important to note that in 12 cases, performance got worse; We discuss the reasons for this shortly in Section 4, following discussion of our second application.

3.2 Use of Tuples in Textual Entailment

Method

Our second application of the tuple database is in recognizing textual entailment (RTE). The RTE task can be described as: given a text T, decide whether a hypothesis text H can be “reasonably concluded” from T. For example, given text T “The president visited Iraq”, it is reasonable to conclude H: “The president traveled to Iraq”.

Like several others, our RTE system (Clark et al, 2008) exploits the DIRT paraphrase database (Lin and Pantel, 2001) to supply some of the required lexical and world knowledge. DIRT contains ~12 million machine-discovered rules of the form “**IF** (X relation Y) **THEN** (X relation’ Y)” (along with an associated confidence value), where relation is a dependency path between X and Y, for example, “**IF** X visits Y **THEN** X travels to Y”. However, although such rules provide leverage, they can often go wrong because they can be instantiated in unlikely or nonsensical ways. Consider the DIRT rule “**IF** X shoots Y **THEN** X injures Y”. While the rule applied to “Fred shot Sue” produces the plausible conclusion “Fred injured Sue”, it produces unusual/nonsensical conclusions applied to “Fred shot the gun”, “Fred shot the bulls-eye”, or “Fred shot the movie”, as we do not normally think of guns/bulls-eyes/movies as being injured. Our world knowledge provides expectations of the way the world might be, and the notion of an “injured gun/bulls-eye/movie” is not within those expecta-

tions, and so we consider this instantiation of the rule unlikely. In our work, we model this filtering using our tuple database to compute confidence in a specific instantiation of a rule (not the rule in general), based on the plausibility of the rule's components. For the conclusion "Fred injures the gun", for example, while there are tuple examples of "person injures X" there are no examples of "X injures gun", so the conclusion is deemed relatively unlikely, even though it is the result of a DIRT rule firing.

Specifically, a DIRT rule's condition and action are each a dependency path between two variables, X and Y. After a rule is instantiated, each link in the instantiated path can be expressed as a tuple, e.g., the path in the instantiated condition "Fred shot the gun" can be expressed as 2 tuples (NV "person" "shoot") and (VN "shoot" "gun"). Similarly the instantiated action "Fred injured the gun" can be expressed as (NV "person" "injure") and (VN "injure" "gun"). Counts for these 4 tuples are 28, 12, 31, and 0 respectively, producing Mutual Information (MI) values of 5.57, 6.76, 7.62, and 0.00. We take the overall plausibility of the rule as the minimum of the MI values (here 0.00), i.e., the rule is only as plausible as its least plausible part (In the case of a tie, we use the MI of the second least plausible tuple as a second-order ranking). While the uninstantiated rule also has an associated confidence value, we empirically found that simply ignoring this rather than combining it with the tuple-derived confidence produced the best results.

It is important to note that this approach only assesses the plausibility of (independently) the condition and action of a particular instantiation of a rule, not the implication itself nor the rule in general.

Evaluation

To recognize entailment, our software first parses the T and H sentences and generates a shallow logical representation (set of ground clauses) for each (Further details are provided in Clark et al., 2008). It then computes whether (the logic for) T, or some DIRT-based implication of T, is subsumed by (the logic for) H. For example T:"A black cat devoured a mouse" is subsumed by (i.e., implies) H: "An animal ate", as (using WordNet's hyponym tree) a cat is an animal, and devouring is

an eating event. Similarly T:"A black cat devoured a mouse" implies H: "The cat swallowed the mouse" via the DIRT rule "IF X eats Y THEN X swallows Y" with confidence 0.14, the general rule's confidence value as supplied in the DIRT database, computed statistically (Lin and Pantel, 2001).

We can use the DIRT-supplied confidences to order the entailments inferred with a DIRT rule. Ideally the most confident entailments will be the actual positive entailments, as labeled in the test suite. Alternatively, we can order the inferred entailments using the tuple-derived confidences of the rule's instantiation. Our interest is in whether the tuple-based confidences produce a better ordering than the DIRT confidences, in particular by giving a low confidence to rule instantiations which seem unusual (as judged by the tuple database).

To assess this, we ran our RTE software on two datasets:

- Our own entailment test suite (250 T-H pairs)⁴
- the PASCAL RTE3 test suite (800 T-H pairs)⁵

For pairs where a DIRT rule was used to connect T and H (49 pairs in our suite, 137 pairs in the PASCAL RTE3 suite), we produced a rank-ordering of those pairs using the DIRT rule confidences, and an alternative ranking using the tuple-derived confidences of the rules' instantiations. To assess the overall quality of a particular rank-ordering, a commonly used measure is "average precision" (AP) (Voorhees and Harman, 1999) defined as "the average of the system's precision values at all points in the ranked list in which recall increases, that is at all points in the ranked list for which the gold standard annotation is YES". More formally, it can be written as follows:

$$AP = \frac{1}{R} \sum_{i=1}^n E(i) \cdot \#correct_i / i$$

where n is the number of the pairs in the test set, $\#correct_i$ is the number of positive pairs in the top i predictions, R is the total number of positive pairs in the test set, $E(i)$ is 1 if the i -th pair is positive and 0 otherwise, and i ranges over the pairs, ordered by their ranking. In the perfect case, the top

⁴ <http://www.cs.utexas.edu/users/pclark/bpi-test-suite/>

⁵ <http://www.pascal-network.org/>

R cases will be the R positive pairs in the test set. The Average Precision results we obtained were as follows:

Data Set	Average Precision	
	Using DIRT confidences	Using tuple confidences
BPI RTE	0.489	0.549
PASCAL RTE3	0.622	0.646

In both cases, the average precision improved. We also experimented with various combination functions to combine both confidence measures, but did not find one where the Average Precision was raised higher than using the tuple-derived confidence measure alone.

4 Discussion and Conclusions

A large amount of world knowledge is essential for many language processing tasks, to create expectations to guide the interpretation process. We have described the extraction of general knowledge fragments (“tuples”) from text and their use in two applications, and our preliminary experiments show the extracted knowledge can improve performance.

Despite these positive indicators, the construction and use of such a database is not a panacea, and there are several significant limitations with our current approach:

- **Missing tuples:** Tuples are generated by procedures associated with grammar rules, so that when a grammar rule is used a tuple is produced. However, some grammar rules do not yet have tuple generation procedures (e.g. for phrases like “force Mary to leave”) and as a result some tuples suggested by the text are not generated.
- **Misparses due to grammar deficiencies:** In some cases, deficiencies in the underlying grammar result in incorrect parses (and thus incorrect tuples). Further improvements to the grammar will reduce these cases.
- **Misparses due to bad costing:** When generating the tuple database, misparses due to bad costing sometimes occur, resulting in bad tuples. As our experiments suggest, reparsing using the tuple database may reduce misparses the second time round, and hence improve the

tuple database itself. We plan to iterate in this fashion in the near future.

- **Systematic bias in the tuple database:** Our hypothesis is that the statistical weight of good tuples (i.e., from correct parses) will overwhelm noise from bad tuples (incorrect parses), and our preliminary experiments provide some support to this hypothesis. However, it is still possible that systematic parsing errors could result in distortions in the tuple database which are more than just low-level noise. In particular, we have chosen to include tuples from parsing all sentences, rather than just tuples from unambiguous sentences, which potentially could cause this phenomenon. Although we do not see evidence of this, we plan to investigate this further.
- **Tuple scoring:** We have used the tuple frequencies and Mutual Information values in our two applications in fairly simple ways. There is clearly room for improving the scoring strategies used in both parsing and textual entailment given the tuple data.
- **Limited data:** Despite generating 53 million tuples, there are often cases where tuple frequencies are zero or near zero. Further increasing the size of the tuple database (by parsing additional corpora) would improve the coverage and reliability of the tuple data.
- **Corpus bias:** Any bias in the corpora used can skew the tuple statistics towards those corpora. In particular, one of our sources was the Reuters corpus, which has a large number of business/financial news stories within it. As a result, business-related tuples have a higher frequency in the database than might be expected otherwise, skewing the tuple statistics from those one would get from a broader coverage source. Similarly, the genre and style of sources will bias tuples also.

Despite these, the accumulation of this type of world knowledge appears advantageous both in theory and in practice. The tuple database provides a multifunctional resource for language processing, and we have illustrated its use for both parsing and, in a novel way, for textual entailment. We are optimistic that further developments are possible, both to improve these existing applications and create new ones.

References

- Alshawi, H., Carter, D. 1994. Training and Scaling Preference Functions for Disambiguation. *Computational Linguistics* 20 (4) pp635-648.
- Banko M., Etzioni, O., 2007. Strategies for Lifelong Knowledge Extraction from the Web. In *Proc. Fourth International Conference on Knowledge Capture (KCAP) 2007*
- Charniak, E. 2000. A Maximum-Entropy-Inspired Parser. In *Proc. 1st NAACL*.
- Clark, P., Hobbs, J., Fellbaum, C., Harrison, P., Murray, W., Thompson, J. 2008. Augmenting WordNet for Deep Understanding of Text. To appear in *Proc. SIGSEM STEP*.
- Collins, M. 2003. Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics*.
- Harrison, P., and Maxwell, M. 1986. A New Implementation of GPSG, *Proc. 6th Canadian Conf on AI (CSCSI'86)*, pp78-83.
- Keller, F., and Lapata, M., 2003. Using the Web to Obtain Frequencies for Unseen Bigrams. *Computational Linguistics* 29 (3).
- Lin, D. 1998. Extracting Collocations from Text Corpora. *Workshop on Computational Terminology*. pp. 57-63.
- Lin, D., and Pantel, P. 2001. Discovery of Inference Rules for Question Answering. *Natural Language Engineering* 7 (4) pp 343-360.
- Marcus, M., Santorini, B., Marcinkiewicz, M. 1993. Building a Large Annotated Corpus of English : The Penn Treebank. *Computational Linguistics*, 19 (2). 313-330.
- Ratnaparkhi, A. 1998. Unsupervised Statistical Models for Prepositional Phrase Attachment. *Proc. COLING-ACL'98*
- Resnik, P. 1997. Selectional preference and sense disambiguation. In *Proceedings of the ACL SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What, and How?*, pages 52-57.
- Schubert, L. 2002. Can we derive general world knowledge from texts?", M. Marcus (ed.), *Proc. of the 2nd Int. Conf. on Human Language Technology Research (HLT 2002)*,
- Schubert, L. and Tong, M. 2003. Extracting and evaluating general world knowledge from the Brown corpus, *Proc. of the HLT/NAACL 2003 Workshop on Text Meaning*.
- Voorhees E., and Harman, D. 1999. Overview of the seventh text retrieval conference. In *Proceedings of the Seventh Text Retrieval Conference (TREC-7)*. NIST Special Publication.