FAST TEXT PROCESSING FOR INFORMATION RETRIEVAL

Tomek Strzalkowski and Barbara Vauthey

Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012
{tomek,vauthey}@cs.nyu.edu

ABSTRACT

We describe an advanced text processing system for information retrieval from natural language document collections. We use both syntactic processing as well as statistical term clustering to obtain a representation of documents which would be more accurate than those obtained with more traditional key-word methods. A reliable top-down parser has been developed that allows for fast processing of large amounts of text, and for a precise identification of desired types of phrases for statistical analysis. Two statistical measures are computed: the measure of informational contribution of words in phrases, and the similarity measure between words.

APPROXIMATE PARSING WITH TTP

TTP (Tagged Text Parser) is a top down English parser specifically designed for fast, reliable processing of large amounts of text. The parser operates on a tagged input, where each word has been marked with a tag indicating a syntactic category: a part of speech with selected morphological features such as number, tense, mode, case and degree.[1] As an example, consider the following sentence from an article appearing in the Communications of the ACM:

> The binary number system offers many advantages over a decimal representation for a high-performance, general-purpose computer.

This sentence is tagged as follows (we show the best-tags option only; dt - determiner, nn - singular noun, nns - plural noun, in - preposition, jj - adjective, vbz - verb in present tense third person singular):

```
[[the,dt],[binary,jj],[number,nn],[system,nn],[offers,vbz],
[many,jj],[advantages,nns],[over,in],[a,dt],[decimal,jj],
[representation,nn],[for,in],[a,dt],[high_performance,nn],
[comS,comS],[general_purpose,nn],[computer,nn],[perS,perS]]
```

Tagging of the input text substantially reduces the search space of a top-down parser since it resolves many lexical ambiguities, such as singular verb vs. plural noun, past tense vs. past participle, or preposition vs. wh-determiner. Tagging also helps to reduce the number of parse structures that can be assigned to a sentence, decreases the demand for consulting of the dictionary, and simplifies dealing with unknown words.

[1] At present we use the 35-tag Penn Treebank Tagset created at the University of Pennsylvania. Prior to parsing, the text is tagged automatically using a program supplied by Bolt Beranek and Newman. We wish to thank Ralph Weischedel and Marie Meeter of BBN for providing and assisting in the use of the tagger.

TTP is based on the Linguistic String Grammar developed by Sager [8] and partially incorporated in the Proteus parser [3]. TTP is written in Quintus Prolog, and currently implements more than 400 grammar productions. The restriction component of the original LSP Grammar as well as the lambda-reduction based "semantics" of the Proteus implementation have been redesigned for the unification-based environment.[2] TTP produces a regularized representation of each parsed sentence that reflects the sentence's logical structure. This representation may differ considerably from a standard parse tree, in that the constituents get moved around (e.g., de-passivization, de-dativization), and certain noun phrases get transformed into equivalent clauses (denominalization). The aim is to produce a uniform representation across different paraphrases; for example, the phrase *context-free language recognition or parsing* is represented as shown below:

```
[[verb,
  [or,[recognize,parse]]]
 [subject,anyone]
 [object,
  [np,[n,language],
    [adj,[context_free]]]]].
```

The parser is equipped with a time-out mechanism that allows for fast closing of more difficult sub-constituents after a preset amount of time has elapsed without producing a parse. When the time-out option is turned on (which happens automatically during the parsing), the parser is permitted to skip portions of input to reach a starter terminal for the next constituent to be parsed, and closing the currently open one (or ones) with whatever partial representation has been generated thus far. The result is an approximate partial parse, which shows the overall structure of the sentence, from which some of the constituents may be missing. Since the time-out option can be regulated by setting an appropriate flag before the parsing starts, the parser may be tuned to reach an acceptable compromise between its speed and precision.

The time-out mechanism is implemented using a straightforward parameter passing and is at present limited to only a subset of nonterminals used by the grammar. Suppose that X is such a nonterminal, and that it occurs on the right-hand side of a production S --> X Y Z. The set of "starters" is computed for Y, which consists of the word tags that can occur as the left-most

[2] See [10] for details.

constituent of Y. This set is passed as a parameter while the parser attempts to recognize X in the input. If X is recognized successfully within a preset time, then the parser proceeds to parse a Y, and nothing else happens. On the other hand, if the parser cannot determine whether there is an X in the input or not, that is, it neither succeeds nor fails in parsing X before being timed out, the unfinished X constituent is closed with a partial parse, and the parser is restarted at the closest element from the starters set for Y that can be found in the remainder of the input. If Y rewrites to an empty string, the starters for Z to the right of Y are added to the starters for Y and both sets are passed as a parameter to X. As an example consider the following clause in the TTP parser (some arguments are removed for expository reasons):

```
clause(SR,P) :-
    sa([pdt,dt,cd,pp,ppS,jj,jjr,jjs,nn,nns,np,nps],PA1),
    subject([vbd,vbz,vbp],Tail,P1),
    verbphrase(SR,Tail,P1,PA1,P),
    subtail(Tail).
```

In this production, a (finite) clause rewrites into an (optional) sentence adjunct (SA), a subject, a verbphrase and subject's right adjunct (SUBTAIL, also optional). With the exception of *subtail*, each predicate has a parameter that specifies the list of "starter" tags for restarting the parser, should the evaluation of this predicate exceed the allotted portion of time. Thus, in case *sa* is aborted before its evaluation is complete, the parser will jump over some elements of the unparsed portion of the input looking for a word that could begin a subject phrase (either a predeterminer, a determiner, a count word, a pronoun, an adjective, a noun, or a proper name). Likewise, when *subject* is timed out, the parser will restart with *verbphrase* at either vbz, vbd or vbp (finite forms of a verb). Note that if *verbphrase* is timed out, then *subtail* will be ignored, both *verbphrase* and *clause* will be closed, and the parser will restart at an element of set *SR* passed down from *clause*. The examples in Figures 1 to 3 show approximate parse structures generated by TTP.

The sentence in Figure 1 has been parsed nearly to the end, but TTP has failed to find the main verb and it has thrown out much of the last phrase *such as the LR(k) grammars*, partly due to an improper tokenization of *LR(k)*. In Figure 2, the parser has initially assumed that the conjunction in the sentence has the narrow scope, then it realized that something went wrong but, apparently, there was no time left to back up. Occasionally, however, sentences may come out substantially truncated, as shown in Figure 3 (where *although* has been mistagged as a preposition).

There are at least several options to realize the kind of time-regulated parsing discussed above. One involves allotting a certain amount of time per sentence and, when this time is up, entering the time-out mode for the rest of the current sentence processing. This amounts to a rapid, though controlled, exit from presently open constituents mostly ignoring the unparsed portion of the sentence. This option gives each sentence in the input roughly the same amount of time, and thus allows the parser to explore more alternatives while processing shorter sentences, while setting tight limits for the longer ones. In our experiments with the CACM collection we found that 0.7 sec/parse is acceptable for an average sentence. One other option is to set up time limits per nonterminal, and restore normal parsing after each time-out. The advantage here is that longer sentences receive proportionally more time to process (allowing for some backtracking to explore alternatives). The disadvantage is that one loses the

SENTENCE:
The problem of determining whether an arbitrary context-free grammar is a member of some easily parsed subclass of grammars such as the LR(k) grammars is considered.

APPROXIMATE PARSE
```
[assert,
 [[verb,be],
   [subject,[np,[n,problem],[t_pos,the],
            [of,
             [[verb,[determine]],
               [subject,anyone],
               [object,
                [[verb,[be]],
                  [subject,[np,[n,grammar],[t_pos,an],
                           [adj,[arbitrary]],[adj,[context_free]]]],
                  [object,[np,[n,member],[t_pos,a],
                          [of,[np,[n,subclass],[t_pos,some],
                               [a_pos_v,
                                [[verb,[parse,[adv,easily]]],
                                  [subject,anyone],
                                  [object,pro]]],
                                  [of,[np,[n,grammar],
                                       [m_wh,
                                        [[verb,be],
                                          [subject,[np,[n,k],[adj,[such]]...]
```

Figure 1. Parsing with TTP.

SENTENCE:
The TX-2 computer at MIT Lincoln Laboratory was used for the implementation of such a system and the characteristics of this implementation are reported.

APPROXIMATE PARSE:
```
[assert,
 [[be],
   [[verb,[use]],
     [subject,anyone],
     [object,[np,[n,computer],[t_pos,the],[adj,[tx_2]]]],
     [for,[np,[n,implementation],[t_pos,the],
          [of,
           [and,
             [np,[n,system],[t_pos,[such,a]]],
             [np,[n,characteristics],[t_pos,the]]]]]]],
     [at,[np,[n,laboratory],[adj,[mit]],[n_pos,[np,[n,lincoln]]]]],
     [of,[np,[n,implementation],[t_pos,this],
          [m_wh,
           [[verb,be],
             [subject,[]]]]]]]]
```

Figure 2. Parsing with TTP.

SENTENCE:
In principle, the system can deal with any orthography, although at present it is limited to 4000 Chinese characters and some mathematical symbols.

APPROXIMATE PARSE:
```
[assert,
 [[can_aux],
   [[verb,[deal]],
     [subject,[np,[n,system],[t_pos,the]]],
     [object,[]],
     [sub_ord,
       [with,
         [[verb,[limit]],
           [subject,anyone],
           [object,[]],
           [to,[np,[n,character],[count,[4000,chinese]]]]]]]],
   [in,[np,[n,principle]]]]]
```

Figure 3. Parsing with TTP.

tight control upon the overall speed of the parser; now complex sentences may take a considerably longer time to finish. Various mixed options are also possible, for instance one may initially allot x milliseconds to each sentence, and if necessary, restart it with a half that time, and so forth.

Another method for containing the time allowed for parsing is to limit the amount of nondeterminism by a stricter control over the rule selection and by disabling backtracking at certain points, again at the expense of producing only an approximate parse. Certain types of structural ambiguity, such as prepositional phrase attachment which cannot be resolved at the syntax level anyway, frequently remain unresolved in the parse structure generated by TTP (although, TTP attempts to resolve some structural ambiguities using preferences whenever possible).

TTP is also quite robust; it can parse nearly every sentence or phrase, provided the latter is reasonably correctly tagged. We parsed the entire CACM-3204 collection and only two sentences were returned unparsed, because of multiple tagging errors.[3] To assure a gradual degradation of output rather than an outright failure, and also to allow for handling of sentence fragments and isolated phrases such as titles, each sentence/phrase is attempted to be analyzed in up to four ways: (1) as a sentence, (2) as a noun phrase or a preposition phrase with a right adjunct(s), (3) as a gerundive clause, and eventually (4) as a series of simple noun phrases. Each of these attempts is allotted a new time slice, and the next analysis is started after the previous one fails but before it is timed out. Although parsing of some sentences may now approach four times the allotted time limit, we noted that the average parsing time per sentence remains basically unaffected.[4]

[3] CACM-3204 is a standard collection used in information retrieval experiments and includes, in addition to the abstracts, a set of 64 queries and relevance judgements for them. The pure text portion of the collection contains nearly 10,000 sentences and phrases, or about 235,000 words.

[4] The average parsing time per sentence is 0.745 sec.

EXTRACTION OF SYNTACTIC PHRASES

The similarity measure that we use for term classification is based on quantitative information about word and phrase frequencies and word co-occurrences within the text. We collected this information for two-word "phrases" extracted from the parsed documents.[5] The co-occurrence analysis gives the best results when the words are connected by the same grammatical relation, for example verb-object, or noun-right adjunct, etc. We noted, however, that including multiple relations in the analysis is possible so long as they could be considered to convey similar "semantic" dependencies. In our experiments the following types of word pairs are extracted: (1) a noun and its left noun adjunct, (2) a noun and the head of its right adjunct, (3) the main verb of a clause and the head of its object phrase, and (4) a noun and its adjective, where the noun is the head of a noun phrase as recognized by the parser.

The pairs are extracted from the regularized parse structures with a pattern-matching procedure which uses an exclusion list to disregard some "uninteresting" words (such as *be, such, any*). The words with the common stem but different forms are replaced by a single "normal" form. Working on the parsed text ensures a high degree of precision in capturing the meaningful phrases, which is especially evident when compared with the results usually obtained from a "raw" text (either unprocessed or only partially processed).[6] On the other hand, since our parser is allowed to skip some portions of each sentence that cannot be parsed within a preset time limit, the structures it produces are occasionally incomplete so that the extraction procedure will generate only a subset of all relevant phrases. The precision, however, remains very high: few undesired phrases are ever turned out (as far as the four specified types are concerned), which is particularly important in subsequent statistical processes, since these tend to be quite sensitive on the amount of noise in the analyzed material. An example is shown in Figure 4.

STATISTICAL SIMILARITY MEASURE

Classification of words and phrases based on similarities in their meaning is particularly important in information retrieval systems. Various word taxonomies derived from machine-readable dictionaries may be of relevance here [1], but general-purpose dictionaries, such as Oxford's Advanced Learner's Dictionary (OALD) or Longman's Dictionary of Contemporary English (LDOCE), both available on-line, are usually quite limited in their coverage of domain specific vocabulary, including domain-specific use of common words as well as technical terminology. Statistical methods for word clustering may provide a partial solution to this problem given a sufficient amount of textual data that display a certain uniformity of subject matter and style. These problems have been studied to some extent within the sublanguage paradigm [4,5], and also using elements of information theory [2,6]. One general problem with the latter approach is that information theory, which deals with code transmission,

[5] Lewis and Croft [7] define the syntactic phrase as "any pair of non-function words in a sentence that are heads of syntactic structures connected by a grammatical relation."

[6] Partial processing may include tagging and/or a limited parsing, see, for example [7], and also [9] for a more comprehensive view.

SENTENCE:
The techniques are discussed and related to a general tape manipulation routine.

PARSE STRUCTURE:
```
[assert,
 [[be],
  [[verb,[and,[discuss],[relate]]],
   [subject,anyone],
   [object,[np,[n,technique],[t_pos,the]]],
   [to,[np,[n,routine],[t_pos,a],[adj,[general]],
        [n_pos,[np,[n,manipulation]]],
        [n_pos,[np,[n,tape]]]]]]]]].
```

EXTRACTED PAIRS:
[discuss,technique], [relate,technique], [routine,general], [routine,manipulation], [manipulation,tape]

Figure 4. Extraction of syntactic pairs.

may not be straightforwardly applicable to the analysis of text where the basic tokens are words of natural language. Church and Hanks [2] used Fano's *mutual information* to compute word co-occurrence patterns in a 44 million word corpus of Associated Press news stories, but they also noted that this measure often produces counterintuitive results. The reason is that the observed frequencies of many words remain low even in very large corpora. For very small counts the mutual information becomes unstable and fails to produce credible results.[7]

Ideally, a measure of relation between words should be stable even at low counts and more sensitive to fluctuations in frequency among different words. We are particularly interested in the low and medium frequency words because of their high indexing value. An interesting comparison among different functions used to study word co-occurrences in the Longman dictionary is presented by Wilks et al. [11]. They assumed that the best function would most closely reflect a correlation between a chance co-occurrence and a minimum relatedness between words, on the one hand, and between the maximum observed frequency of co-occurrence and a maximum relatedness, on the other.[8]

Another question is whether the relatedness measure should be symmetric. In other words, for any given pair of words, can we assume that they contribute equally to their mutual relationship? We felt that the words making up a syntactic phrase do not contribute equally to the informational value of the phrase and that their contributions depend upon the distribution characteristics of each word within a particular type of text. For example, in a general computer science text the information attached to the phrase *parallel system* is more significantly related to the word *parallel* than to the word *system*. This relationship can change if the phrase is found in a different type of text where *parallel* is more commonplace than *system*, for example, in a text from a parallel computation subdomain.

Based on these considerations, we introduce an asymmetric measure of informational contribution of words in syntactic phrases. This measure $IC(x, [x,y])$ is based on (an estimate of) the conditional probability of seeing a word y to the right of the word x, [9] modified with a dispersion parameter for x. The dispersion parameter, d_x, understood as the number of distinct words with which x is paired, has been defined as follows ($f_{x,y}$ is the observed frequency of the pair [x,y]):

$$d_x = \sum_y \delta_{x,y}$$

where

$$\delta_{x,y} = \begin{cases} 0 & \text{if } f_{x,y}=0 \\ 1 & \text{if } f_{x,y}>0 \end{cases}$$

For each word x occurring in any of the selected syntactic phrases, the informational contribution of this word in a pair of words [x,y] is calculated according to the following formula:

$$IC(x, [x,y]) = \frac{f_{x,y}}{n_x + d_x - 1}$$

where n_x is the number of pairs in which x occurs at the same position as in [x,y]. $IC(x, [x,y])$ takes values from the $<0,1>$ interval; it is equal to 0 when x and y never occur together (i.e., $f_{x,y} = 0$), and it is equal to 1 when x occurs only with y (i.e., $f_{x,y} = n_x$ and $d_x = 1$). Empirical tests with this formula on the CACM-3204 collection give generally satisfactory results, and a further improvement may be possible if larger corpora are used (perhaps 1 million words or more). For each pair of words $[x,y]$ two informational contribution values are calculated: $IC(x, [x,y])$ and $IC(y, [x,y])$, and they may differ considerably as seen in Table 1.[10] The relative similarity between any two words is measured in terms of their occurrence in common contexts and is the sum of the informational contributions of the context weighted with the informational contribution of the less significant of the two words. A partial similarity for words x_1 and x_2 in the context of another word y is therefore given as:

$$sim_y(x_1,x_2) = \rho_y(x_1,x_2)\,(IC(y, [x_1,y]) + IC(y, [x_2,y]))$$

where

$$\rho_y(x_1,x_2) = min(IC(x_1,[x_1,y]), IC(x_2,[x_2,y]))$$

The total similarity between two words x_1 and x_2 is given as a sum of all partial similarities, normalized with a logarithmic function.

$$SIM(x_1,x_2) = log\,(1000 \sum_y sim_y(x_1,x_2))$$

We calculated the similarity measure for any two words which occurred in at least two common contexts, that is, those which have been paired with a common word in at least two distinct occasions. The results are summarized in Tables 1 to 3. In Table 1 we list the values of IC function for selected pairs. Tables 2

[7] This may be contrasted with a distribution of symbols from a small finite alphabet.

[8] A chance co-occurrence of a pair of words is when the probability of their occurring together is the product of the probabilities of their being observed independently. Two words have the largest possible frequency of co-occurrence if they never occur separately. Unfortunately, a chance co-occurrence is very difficult to observe.

[9] The conditional probability formula produced the best results in the experiments reported in [11].

[10] All tables are placed at the end of the paper.

and 3 show the top elements in the similarity classes generated for words *grammar* and *memory*. We noted that the similarity value of about 2.0 or more usually coincided with a high degree of correlation in meaning, while the smaller values were generally less interesting.

This first classification can be further improved by distinguishing among word senses. Many words have multiple senses, and these, rather than the lexical words themselves, should be used in indexing a text. However, obtaining a right dissociation between different senses of a word presents a separate research problem which is beyond the scope of this paper.

CONCLUSIONS

In this paper we described the experiments with an efficient processing of large collections of natural language documents that could lead to an effective and reliable method for automated indexing of text in information retrieval applications. The documents are initially tagged with a stochastic tagger, and then parsed with the TTP parser that generates approximate regularized "logical" structure for each sentence. These structures are subsequently analyzed by various statistical processes that collect data about word frequencies, co-occurrences and similarities. The results obtained in deriving word pairs show a marked improvement in precision for capturing the "correct" word dependencies as compared to more traditional methods in information retrieval that use only very limited parsing [7]. The computed similarity sets are quite interesting and they produce meaningful classifications. These results can still be improved if the statistical data is collected from a larger amount of text. We believe that the improved precision in text indexing will translate into an improved precision in document retrieval.

ACKNOWLEDGEMENTS

This paper is based upon work supported by the Defense Advanced Research Project Agency under Contract N00014-90-J-1851 from the Office of Naval Research, the National Science Foundation under Grant IRI-89-02304, and a grant from the Swiss National Foundation for Scientific Research.

REFERENCES

[1] Chodorow, Martin S., Roy J. Byrd, and George E. Heidorn. 1985. "Extracting semantic hierarchies from a large on-line dictionary." *Proc. of the 23rd Meeting of the ACL*, pp. 299-304.

[2] Church, Kenneth Ward and Hanks, Patrick. 1990. "Word association norms, mutual information, and lexicography." *Computational Linguistics*, 16(1), MIT Press, pp. 22-29.

[3] Grishman, Ralph. 1986. *Proteus Parser Reference Manual.* Proteus Project Memorandum #4, Courant Institute of Mathematical Sciences, New York University.

[4] Grishman, Ralph, and Kittredge, Richard (eds). 1986. *Analyzing Language in Restricted Domains: Sublanguage Description and Processing.* Lawrence Erlbaum Assoc., Hillsdale, NJ.

[5] Grishman, Ralph, Lynette Hirschman,and Ngo T. Nhan. 1986. "Discovery procedures for sublanguage selectional patterns: initial experiments". *Computational Linguistics*, 12(3), pp. 205-215.

[6] Hindle, Donald. 1990. "Noun classification from predicate-argument structures." Proc. 28 Meeting of the ACL, Pittsburgh, PA, pp. 268-275.

[7] Lewis, David D. and Croft, W. Bruce. 1990. "Term clustering of syntactic phrases." Proc. 13th ACM-SIGIR Conference, Brussels, Belgium, pp. 385-404.

[8] Sager, Naomi. 1981. *Natural Language Information Processing*. Addison-Wesley.

[9] Salton, Gerard. 1989. *Automatic Text Processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley, Reading, MA.

[10] Strzalkowski, Tomek. 1990. "Reversible logic grammars for natural language parsing and generation." *Computational Intelligence*, 6(3), NRC Canada, pp. 145-171.

[11] Wilks, Yorick A., Dan Fass, Cheng-ming Guo, James E. McDonald, Tony Plate, and Brian M. Slator. 1990. "Providing machine tractable dictionary tools." *Machine Translation*, 5, pp. 99-154.

Table 1. Informational Contribution for selected word pairs

$[x,y]$	$f_{x,y}$	n_x	d_x	$IC(x, [x,y])$	n_y	d_y	$IC(y, [x,y])$
[system,parallel]	2	910	322	0.0016	57	24	0.025
[system,computation]	78	910	322	0.0634	740	201	0.0830
[path,parallel]	1	19	14	0.0313	57	24	0.0125
[class,grammar]	5	128	86	0.0235	47	34	0.0625
[define,grammar]	3	131	80	0.0143	47	34	0.0375
[class,language]	1	128	86	0.0047	295	116	0.0024
[define,language]	9	131	80	0.0429	295	116	0.0220

Table 2. Words most similar to *grammar*		
word	*similarity value*	*common context*
logic	2.16	analysis, application, equivalent, inference, network
language	1.98	analysis, application, class, code, construct, define, extend, inference, program, sentence, syntax, term, use
reduction	1.93	class, equivalent
data	1.88	analysis, base, class, code, define, include, modify, network, processor, program, use
circuit	1.66	analysis, equivalent, use
match	1.64	equivalent, use
computation	1.56	analysis, application, class, code, construct, detect, extend, network, production, program, use
technique	1.54	analysis, application, base, class, code, extend, include, modify, program, show, use
area	1.53	class, cover, extend, include, use
algorithm	1.43	analysis, application, base, class, code, construct, extend, include, modify, program, restrict, show, use

Table 3. Words most similar to *memory*		
word	*similarity value*	*common context*
storage	3.41	access, allocate, amount, block, capacity, contain, effect, hierarchy, number, operate, organization, partition, reference, request, size, space, structure, system, time, unit, use, word
space	2.79	allocate, amount, concept, limit, model, multics, page, partition, request
resource	2.68	allocate, to the amount, compute, management, request, share, time, use
computation	2.56	allocate, character, compose, concept, configuration, effect, environment, function, management, model, number, operate, page, program, protection, request, resource, share, simulate, storage, system, technology, time, use, word
block	2.38	access, allocate, buffer, concept, locate, occupy, size, storage, structure, use
file	2.25	allocate, character, concept, contain, locate, number, organization, size, storage, structure, system, use
system	2.23	block, character, compute, concept, configuration, contain, core, effect, function, limit, model, number, operate, organization, part, program, protection, request, resource, section, share, simulate, storage, structure, technology, unit, use, view
buffer	2.16	allocate, block, request, size, storage, use
core	2.05	allocate, resident, storage, unit, use
process	1.96	allocate, amount, character, concept, configuration, effect, end, environment, function, hierarchy, number, object, operate, organization, program, request, share, size, system, time, unit, use, view