An Efficient Chart-based Algorithm for Partial-Parsing of Unrestricted Texts

David D. McDonald
14 Brantwood Road, Arlington MA 02174-8004 USA
MCDONALD @ BRANDEIS.EDU (617) 646-4124

Abstract

We present an efficient algorithm for chart-based phrase structure parsing of natural language that is tailored to the problem of extracting specific information from unrestricted texts where many of the words are unknown and much of the text is irrelevant to the task. The parser gains algorithmic efficiency through a reduction of its search space. As each new edge is added to the chart, the algorithm checks only the topmost of the edges adjacent to it, rather than all such edges as in conventional treatments. The resulting spanning edges are insured to be the correct ones by carefully controlling the order in which edges are introduced so that every final constituent covers the longest possible span. This is facilitated through the use of phrase boundary heuristics based on the placement of function words, and by heuristic rules that permit certain kinds of phrases to be deduced despite the presence of unknown words. A further reduction in the search space is achieved by using semantic rather than syntactic categories on the terminal and nonterminal edges, thereby reducing the amount of ambiguity and thus the number of edges, since only edges with a valid semantic interpretation are ever introduced.

1. Introduction

Much of the research being done in parsing today is directed towards the problem of information extraction, sometimes referred to as "message processing" or "populating data-bases". The goal of this work is to develop systems that can robustly extract information from massive corpora of unrestricted ("open") texts. We have developed such a system and applied it to the task of extracting information about changes in employment found in articles from the "Who's News" column of the Wall Street Journal. We call the system "*Sparser*". There are many possible design goals for a parser. For Sparser we have three:

- It must handle unrestricted texts, to be taken directly from online news services without human intervention or preprocessing.
- It must operate efficiently and robustly, and be able to handle articles of arbitrary size using a fixed, relatively small set of resources.
- Its purpose is the identification and extraction specifically targeted, literal information in order to populate a database. Linguistically motivated structural descriptions of the text are a means to that end, not an end in themselves.

In the rest of this section we will look at some of the consequences of these goals for the parser's design. The most important of these is how to cope with the fact that many of the words in the text will be unknown, which we will take up first. We then look at the consequences of designing the parser for the specific purpose of information extraction. In section two we will look at how well Sparser has been able to meet these goals after roughly fifteen months of development following more than two years of experimentation with other designs, and then go on in the later sections to situate and describe its phrase structure algorithm.

1.1 "Partial Parsing"

Attempting to understand an entire text, or even to give all of it a structural description, is well beyond the state of the art for today's parser's on unrestricted texts. Instead, text extraction systems are typically designed to recover only a single kind of information. They focus their analyses on only those portions of the text where this information occurs, skipping over the other portions or giving them only a minimal analysis. Following an emerging convention, we will call such a system a *partial parser*.

The competence of a partial parser may be compared with that of people who are learning a second language. They can be expected to know most of the language's syntax, function words, and morphology, to know certain idioms, the conventions for forming common constructions such as names, dates, times, or amounts of money, and the high frequency open-class vocabulary such as "said", "make", "new", etc. However, their knowledge of the vocabulary and phrasings for specific subjects will be severely limited and particular. A few topics will be understood quite well, others not at all.

Their limited comprehension notwithstanding, such people can nevertheless scan entire articles, accurately picking out and analyzing those parts that are about topics within their competence, while ignoring the rest except perhaps for isolated names and phrasal fragments. They will know when they have correctly understood a portion of the text that fell within their competence, and can gauge how thorough or reliable their understanding of these segments is.

1.2 The impact of unknown words

Mirroring this kind of partial but precise competence in a parser is not simply a matter of finding the portions of the text on the understood topics and then proceeding to analyze them as a normal system would. Such a strategy will not work because instances of unknown words and subject matter can occur at any granularity—not just paragraphs and sentences, but appositives, clausal adjuncts, adverbials, all the way down to adjectives and compound nouns within otherwise understandable NPs. For example, an understandable subject-verb combination may be separated by an appositive outside the system's competence, understandable pp-adjuncts separated by incomprehensible ones, and so on. The example below (from the Wall Street Journal for February 14, 1991) shows a case of a relative clause, off the topic of employment change, situated between an understood subject NP and an understood VP. (Understood segments shown in bold.)

... ***Robert A. Beck, a 65-year-old former Prudential chairman*** *who originally bought the brokerage firm,* ***was named chief executive of Prudential Bache.*** *...*

As a result of this and other factors, the design of a partial parser must be adapted to a new set of expectations quite different from the customary experience working with carefully chosen example sentences or even with most question-answering systems. In particular:

(1) Don't expect to complete full sentences

Because the unknown vocabulary can occur at any point, one cannot assume that the parser will be able to reliably recover sentence boundaries, and its grammar should not depend on that ability.

To this end, Sparser parses opportunistically and bottom up rather than predicting that an S will be completed. Its structural descriptions are typically a "forest" of minimal phrasal trees interspersed with unknown words. The only reliable boundaries are those signalled orthographically, such as paragraphs.

(2) Develop new kinds of algorithms for connecting constituents separated by unparsed segments of the text.

The standard phrase structure algorithms are based on the completion of rewrite rules that are driven by the adjacency of labeled constituents. When an off-topic and therefore uncompleted text segment intervenes, as in the example just above, an adjacency-based mechanism will not work, and some other mechanism will have to be employed.

Sparser includes a semantically-driven search mechanism that scans the forest for compatible phrases whenever a syntactically incomplete or unattached phrase is left after conventional rules have been applied. It is sensitive to the kind of grammatical relation that would have to hold between the two constituents, e.g. subject – predicate, and constrains the search to be sensitive to the features of the partially parsed text between them, e.g. that if in its search it finds evidence of a tensed verb that is not contained inside a relative clause, then it should abort the search.

(3) Supplement the phrase structure rule backbone of the parser with an independent means of identifying phrase boundaries.

Very often, the off-topic, unknown vocabulary is encapsulated within quite understandable phrasal contexts. Consider the real example "... ***this*** *gold mining* ***company was*** ...". Here the initial determiner establishes the beginning of a noun phrase, and the tensed auxiliary verb establishes that whatever phrase preceded it has finished (barring adverbs). Forming an NP over the entire phrase is appropriate, even when the words "gold" and "mining" are unknown because they are part of an open-ended vocabulary.

Sparser includes a set of function word-driven phrase boundary rules. And it has a very successful heuristic for forming and categorizing text segments such as this example ("successful" in that it generated no false positives in the test described in §2). Simply stated, if there is a rule in the grammar that combines the first and last edges in a bounded segment (e.g. a rule that would succeed on the phrase "*this company*"), then allow that rule to complete, covering the unknown words as well as the known.

1.3 Objects rather than expressions

Sparser was written to support tasks based on populating data bases with commercially significant literal information extracted in real time from online news services, and this requirement has permeated nearly every aspect of its design. In particular, it has meant that it is not adequate to have the output of an analysis be just a syntactic structural description (a parse tree), or even a logical form or its rendering into a database access language like SQL, as is done in many question-answering systems. Instead, the output must be tuples relating individual companies, people, titles, etc., most of which will have never been seen by the system before and which are not available in pre-stored tables. These requirements led to the following features of Sparser's design:

- The system includes a domain model, wherein classes and individuals are represented as unique, first-class objects and indexed by keys and tables like they would be in a database, rather than stored as descriptive expressions.
- While many individuals will have never been seen before, a very significant number will continually reoccur: specific titles, months, dates, numbers, etc., and they should be referenced directly. The system includes facilities for defining rules for the parser as a side-effect of defining object classes or individuals in the domain.
- Interpretation is done as part of the parsing of linguistic form, rather than as a follow-on process as is customary. This is greatly facilitated by the next point:
- semantic categories are used as the terms in the phrase structure rules.[1]

Space limitations do not permit properly describing th tie-in from the parsing of structural form (the realm of th parser proper) to the domain model/database. Briefly, a rule by-rule correspondence is supported between the syntax an

[1] This is sometimes referred to as using a "semantic grammar" This nomenclature can be misleading, as the form and use of th phrase structure grammar is just the same as in a conventional syntactically labeled grammar, i.e. phrases are formed on th basis of the adjacency of labeled constituents or terminals. Al that changes is that most of the terms in rules are now label like "company" or "year", rather than "NP" or "verb".

the semantics,[2] whereby each rewrite rule is given a corresponding interpretation in the model. Individuals and types in the model are structured so that their compositionality mimics the compositional structure of the corresponding English phrase(s).

The correspondence is grounded in the means by which individual content words are defined for the parser. Briefly, the point is that whenever one defines a class of objects or particular individuals so that they can be represented in one's domain model, that same act of definition can be made to result in a rule(s) being written for the parser so that whenever that rule completes, the resulting edge can immediately include a pointer to the domain object. Compound objects such as events are formed through the composition of these basic individuals, under the control of the interpretation rules that accompany the rules that dictate the composition of the phrases in the English text.

```
(define-title-head "president")
(define-title-modifier "assistant")
(define-month :name "December"
         :abbreviation "Dec"
         :position-in-the-year 12
         :number-of-days 31 )
```

2. Test Results

We put SPARSER and its first large grammar through a substantial test at the end May 1991. The task was to extract information on people changing jobs. The articles were from the Wall Street Journal, as downloaded off the Dow Jones News service; the example below is a faithful reproduction of what one of those articles looks like as the news service provides them.

The test consisted of 203 articles; literally the second half of all articles that the Journal published in February 1991 whose electronic version had the tag "WNEWS". They included long columns on advertising and law that mentioned a job change incidentally, and some feature articles. About two thirds were from the Journal's "Who's News" column, where the article below is a typical example. It is the first article from the test set.

AN 910214-0090
HL Who's News: Goodyear Tire & Rubber Co.
DD 02/14/91
SO WALL STREET JOURNAL (J), PAGE B8
CO * GT WNEWS
IN PIPELINE OPERATORS (PIP) PETROLEUM (PET) AUTO PARTS AND EQUIPMENT INCLUDING TIRES (AUP)
TX GOODYEAR TIRE & RUBBER Co. (Akron, Ohio) -- George R. Hargreaves, vice president and treasurer of Goodyear, will become president and chief executive officer of the Celeron Corp. unit, a holding company for Goodyear's All American Pipeline. Mr. Hargreaves, 61, will assume the post effective March 1 and will retain his current posts. Robert W. Milk, Celeron's current president and chief executive, as well as an executive vice president for Goodyear, will be on special assignment until he retires April 30.

The task was to extract relations (database tuples), such as the one below, that give the action, person or persons affected, the position (title), and the company or subsidiary. In the text, this corresponds to each clause with a relevant verb, and their variants in reduced clauses, conjunctions, relatives, lists, etc. (though by convention it does not include the appositives, since they give current information rather than changes). It also included redundant instances, such as nominalizations or anaphoric references (*the post*). There are four instances in this article, of which SPARSER found three, missing the meaning of ***his current posts*** because of rule interference with a recently changed definition for *current*. The example below is the first of those relations.

```
#<edge75  80 Job-event 105
 event: #<event-type become-title>
 title: (#<title "president"
        #<title "chief executive officer">)
 person: #<person Hargreaves, George R.>
 company: #<subsidiary
          of: #<company Goodyear Tire &
                   Rubber Company>
          name: #<co-name
                    "Celeron Corporation">>>
```

Overall, SPARSER found 81% of all the possible job-change relations (597/735). Within the relations that it found, 81.5% of them had all of their fields filled with the correct values (486/597). The false positive rate was 3% (19/616). Given the limited size of the test set (203 articles, 735 possible relations), a better way to state these results is that the system found 4/5ths of the relations, and that 4/5ths of those were correct in all respects. Most of the deficits in precision were due to failing to find a value for a field, rather than filling it with an incorrect value; the number of relations with an actual mistake in one field was 6% (36/597).

Roughly three man months went into preparing the grammar.[3] The development corpus was the articles from

[2] Note that this is now "semantics" in the sense of finding the denotation of a formula (English phrase) in some model, not in the sense of the choice of labels in a "semantic grammar". For example, when the parser identifies an NP that is labeled as a "company", that labeling is syntactic and restricts how the NP can be composed into larger phrases. The denotation of that NP, which SPARSER constructs on the basis of its rules of interpretation, is the particular individual company that the NP refers to, or more precisely, the representation of that company in SPARSER's internal data base.

[3] It is not very informative to report that there were 2,092 rewrite rules in the grammar on the day of the test, since this number includes the definition of 40 individual years, the 12 months and their abbreviations, upper and lowercase forms of most of the words, etc. The number also omits the grammar for proper names and for numbers, since these are organized on a quite different basis. To give some idea of its relevant size, we can point out that it supported 12 topic-specific verb subcategorization frames and 31 topic-specific verbs, 25 title heads and 30 title modifiers, and that about 25% of the 244 mistakes counted in the test could be attributed missing some

December 1990 and from the first half of February. Probably an additional two months would have been required to bring the grammar up to full competence on that corpus; entire classes of constructions that were known to be relevant were not implemented at the time of the test, including definite descriptions of titles acting as people (*five vice presidents were* ...), and conjuncts that did not have objects of identical type directly on each side of the conjunction. The grammar overall does, however, have reasonable competence in definite references and pronouns, participles, relative clauses, and appositives.

Its accuracy, especially in such areas as pp-attachment, stems from its use of semantic rather than strictly syntactic terms in its rules. This means that a non-trivial amount of extension is required for each new topic area that a grammar is written for, though much of what is needed will be analogous to what is already in place, and the syntactic base, with its treatments auxiliaries, determiners, relative pronouns, etc. can be carried forward unchanged.

As a program, Sparser is quite robust. In other applications the system has been run continuously without error for more than 30 hours, and it has handled magazine articles more than forty thousand words long.

We will now look at Sparser's algorithm for phrase structure parsing. We begin by introducing the rationale for the algorithm by comparing it with other common phrase structure parsing methods. Then in §4 we look at the tokenizer, the chart, and the phrase structure rules, finally moving to the details of the algorithm and examples of its use. Unfortunately space does not permit more than a passing mention of the other parsing algorithms SPARSER uses in conjunction with phrase structure parsing; a brief prècis of these companion parsing techniques can be found in McDonald (1990).

3. Placing the phrase structure algorithm in context

Sparser forms its analysis in one pass through the text from left to right (beginning to end). The backbone of the analysis is a set of context free and context sensitive phrase structure rewrite rules of the usual sort. These rules are applied to the text to form edges (parse nodes) over the terminal words and other edges–their daughter constituents. The final set of maximal, connected edges constitutes the parser's analysis of the text's form and linguistic relations. A parallel set of projected denotations for these edges in the designated domain model constitutes Sparser's analysis of the text's meaning, as briefly sketched in §1.3.

3.1 Standard phrase structure algorithms

Phrase structure parsing can be seen as a kind of search. One looks for the best analysis of the text by searching the space of possible analyses permitted by the grammar to see which one best describes the derivation of the text. To be sure of arriving at the correct analysis, the search must be thorough enough to ensure that no valid analysis is missed. At the same time, the search space should be as small as possible to ensure efficiency.

In considering efficiency, we must trade off the simplicity of the control structure against the amount or complexity of the state information that the algorithm calls for. The simplest control algorithm is probably the nested loops of the CKY algorithm (see, e.g., Aho & Ullman 1972). This algorithm searches for parse nodes of all possible word lengths and starting positions. It looks through all legal values of three indices, $0 \leq i < j < k \leq n$ (where n is the length of the input text), to determine whether two adjacent candidate daughters, one spanning the text from index i to index j and the other from j to k, can be combined to form a new node from i to k. This algorithm takes only a few lines to write, but since it is driven by the space of index values, it necessarily requires On^3 time to complete its search, along with potentially $n^2/2$ storage cells to record its intermediate results.

Other familiar algorithms reduce the search space by, in effect, only looking at those points in the space where there is guaranteed to be something to see. They pay for this in a more elaborate control structure. Using Earley's algorithm (1970) as the model, we can summarize their procedures as typically first *predicting*, top-down, what constituents could legally occur given the rules of the grammar. Then, as they sequentially *scan* the terminals of the text, either from the left end or the right, they incrementally confirm some of these predictions by *completing* hypothesized rules bottom up as all of a rule's daughter constituents are found. With common grammars (bounded state), Earley's algorithm runs in order of n time, but at a cost in storage potentially as great as G^2, where G is the number of rules in the grammar.

The bulk of this storage cost in Earley's algorithm is due to its representation of the predictions, i.e. a listing of all of the potentially completable rules that is modified as each terminal is scanned and edges are completed. An active-edge chart algorithm (Kay 1980, Kaplan 1973; a good textbook treatment can be found in Winograd 1983) has a comparable storage cost because it also maintains an online representation of the production rules that are relevant to the analysis, though the particulars of how it represents these partially instantiated rules are quite different from Earley's given the differences in their control structures.

In a parser designed for unrestricted, multi-paragraph text, like Sparser, there are problems with using any explicit runtime representation of potentially completable rules. The near-inevitability that the sentences will be broken up by unknown words means that one cannot assume that all of the root edges in the final forest will be labeled with "S". As a result, one must include in the set of starting labels for the predictions essentially all of the lefthand-side labels in the grammar's rule set. (The treatment in Martin, Church & Patil 1981 handles this "all predictions at all vertexes" problem very elegantly.) Given that Sparser presently contains approximately 300 non-terminal labels in its semantic grammar, any algorithm with an order of G storage cost would be prohibitively expensive.

rewrite rule and 20% to missing vocabulary (8% to the single case *hold <position>*).

3.2 Introduction to SPARSER's algorithm

To predict everything, however, is to constrain nothing, and so the natural alternative is of course to form phrases bottom up, using only the "scan" and "complete" aspects of the basic algorithm. SPARSER uses a bottom-up parsing algorithm for its phrase structure rules. All of the edges in its chart are what would be called "inactive" edges in the above approaches—they all represent actual constituents in the text rather hypothesized ones.

Without the constraint provided by prediction that every edge will be used in the final analysis, a conventional bottom-up algorithm suffers from two kinds of problems.

- Locally correct but globally misaligned edges can result in additional, unconnected edges that will not be part of the final, maximal analysis.
- The very same combination of constituents may be parsed in several different orders, resulting in multiple, "spurious" edges covering the same span and with the same meaning, where only one is needed.

Sparser addresses the problem of misalignment by forcing its its initial, "segment by segment" parsing to conform to a linguistically-motivated "grid" that is formed from phrase boundary information taken from the location of closed-class words; see §4.4.

Sparser addresses the spurious edge problem by drastically restricting its search space of adjacent edges. When a new edge is entered and checked against its adjacent neighbor edges for possible completion of a new edge, only the single "topmost" neighbor edge at a position is checked, rather than all of the edges that have accumulated at that position as is customary in a bottom-up algorithm. This topmost edge will be the one most recently entered into the chart, and it will be the longest thus far to start/end at that position.

This reduction in the search space by checking against only topmost edges can dramatically lower the number of checks made and edges entered. The exact amounts vary with the grammar and the text. The greatest savings comes in conjunctions or in cases where a head labeled with a recursive category can take several complements from either its left or right (e.g. a verb phrase taking auxiliaries to its left and optional adjuncts to its right)—constructions that are ubiquitous in news articles. The savings is multiplicative: If there are m complements to the left of the head and n complements to the right, and if each composition of a complement and its accumulated head+complements neighbor phrase yields a new edge with the same label as the head, then the number of edges formed if all neighbors are checked is $m*n$. If only the top neighbor is checked the number is $m+n$.

A further reduction in the search space is achieved through Sparser's use of a semantic grammar. Each preterminal edge for a open class word will have a semantic classification (label) corresponding to the kind of thing it denotes (see §1.3). If a word is ambiguous it will introduce multiple edges, one for each interpretation. By writing the rules for phrasal composition in terms of these classifications, we insure that *no phrase will be formed by the parser unless it has a semantic interpretation.* This cuts down dramatically on the amount of structural ambiguity that the parser must cope with; indeed, in nearly all cases examined so far, polysemous words have been disambiguated by the first rule that applies to them to form a larger phrase. Prepositional attachment, in particular, has not proved a problem since one is not adding a PP, waiting for a later semantic interpretation process to rule on whether the combination makes sense, but instead adding a phrase labeled, e.g., "for-company", whose rules of combination are markedly more specific.

4. The Details of the Algorithm

We will now look at particulars of the scan routine, the chart, and the phrase structure rules, and then move on to describe the phrase structure parsing algorithm in the context of a short example. Overall, Sparser is a transducer taking as input a stream of ascii characters and producing as output (a) a recycled chart of completed edges with their denotations in the domain model, and (b) a sequence of user-specified actions triggered at hooks embedded within the core algorithms such as the completion of an edge with a given label. One use of these hooks/actions has been to collect, e.g., the maximal job-event edges in an article so that they can be readout into the data base. We will not discuss them further in this paper.

4.1 Operations over terminals

At the base of the parser's operation is a scan operation that identifies (delimits) minimal tokens within the input character stream, consuming it in the process, and adding edges to the chart. Phrase boundary brackets are also entered into the chart when function words are scanned, as described in §4.4.

As each token is delimited within the stream, it is looked up in the word list. If it is known, the predefined object that represents it (a "word") is entered into the chart, along with any edges dictated by the grammar. If it is unknown—new to the parser—a word object is constructed for it, and its string examined to characterize its morphological and capitalization properties. Tokens are minimal sequences of the same character type, e.g. the sequence "$43.3 million" is seen as six tokens: "$", "43", ".", "2", <one space>, "million". All larger combinations are formed through phrase structure or other sorts of rules.

In addition to the introduction of preterminal edges, non-phrase structure rules of various sorts may be associated with tokens and are executed as the tokens are scanned. These include

- simple "polyword" rules that interpret a sequence of terminals as a single, inseparable entity (e.g. "holding company", "Wall Street Journal");
- rules for forming constituents on the basis of paired punctuation such as parentheses or brackets, or for special conventions such as SGML tags;
- complex rules for the formation of constituents with "flat", Kleene star -style internal structures, in particular proper names; and

- arbitrary actions outside the parser's scope, e.g. to do word counts, or to feed a topic detection algorithm that does not use Sparser's later stages.

A proper discussion of the algorithms for these operations and their integration into the parsing algorithm as a whole is beyond the scope of this paper. Suffice it to say that once triggered they execute as self-contained processes, and that their results are always recorded as one or more edges that they add to the chart, spanning the appropriate amount of text.

4.2 The Chart

Sparser's chart is comprised of three kinds of data structures: positions, edges, and edge-vectors. Positions provide indices to record the sequence of the terminals and indicate the spans of the edges. They correspond to the "vertices" between edges as used in other chart algorithms, but here they are first class objects with their own primary definition in terms of the sequence of terminals, rather than being dependent on the notion of edges. Following the usual convention, positions are located between the terminals.

From the point of view of the parsing algorithm, there is unlimited stream of positions, starting with the position with index zero that precedes the dummy terminal representing the start of the text, and continuing terminal by terminal until the tokenizer has exhausted the input character stream. The implementation is actually in terms of a fixed length array filled with position objects. This array grounds the notion of successive positions. The Scan operation will make the array wrap around and write over earlier position objects as needed when the length of the text exceeds the length of the array. The utility of this fixed, recycled resource is that it allows SPARSER to handle texts of arbitrary length, so long as the array is longer than the longest span of terminals over which some adjacency-driven phrase structure rule is expected to apply. A length of 250 has proved more than adequate in the Who's News domain.

Edges represent the completion of rules, or mark the presence of terminals that are mentioned as literals in some rule. An edge has a label, which will be the lefthand-side term of the corresponding rule, and records the constituent edges (or single terminal/edge) that it spans. An edge's daughter edges can be readout recursively as a parse tree marking the sequence (derivation) of rules that constitutes the grammar's analysis of the sequence of terminals the edge spans. Like positions, edges are implemented as a recycled resource; the customary number of edge objects is 500.

Edge-vectors link positions to edges. Each position has two edge vectors, one recording the edges that end at that position, the other the edges that begin at that position. The edges in each vector are sorted historically: the first edge in a vector will be the first edge to have been introduced in to the chart that ended/started at that position; the last edge will be the most recent. The most recently introduced edge is referred to at the "top" edge to end/start at the position; this edge is pointed to directly by the edge vector object because of its importance to the parsing algorithm. Given the nature of the parsing algorithm it will also be the longest edge to end/start at the position.

4.3 The phrase structure grammar

The phrase structure grammar consists of a set of rewrite rules. The rules define patterns of labeled adjacent immediate constituents in the usual manner. The labels are either literal words (or any other sort of token such as punctuation or in some cases even whitespace), or they are atomic category symbols.

Shown below are some of the rules used in the analysis of the sample article, given in the usual notation as terms on the left and righthand-sides of an arrow. The righthand side terms are the labels on immediate constituents; the lefthand term will be their parent, labeling any edge formed by the completion of that rule.

```
(1)  head-of-subsidiary-phrase  ->  "unit"
(2)  head-of-subsidiary-phrase  ->
        company  head-of-subsidiary-phrase
(3)  subsidiary-company   ->
        "the"  head-of-subsidiary-phrase
(4)  company-possessive  ->
        company  apostrophe-s
(5)  name  ->  company
        /   company-possessive  ____
(6)  for-company  ->  "for"  company
```

As part of not needing to support an "active edge" representation of partially complete rules during runtime, Sparser's basic operation, which we can call "check", is defined over a pair of adjacent edges. A table is consulted to see if there is some rule (or dotted expansion of a rule, see below) that lists the labels of those two edges, in order, as its righthand side. If there is such a rule, a new edge is constructed and entered into the chart. If a context free rule is involved, then the edge will span both daughter edges and is labeled with the lefthand side term of the rule. If it is a context sensitive rule, then the designated daughter edge will be respanned and given that label.

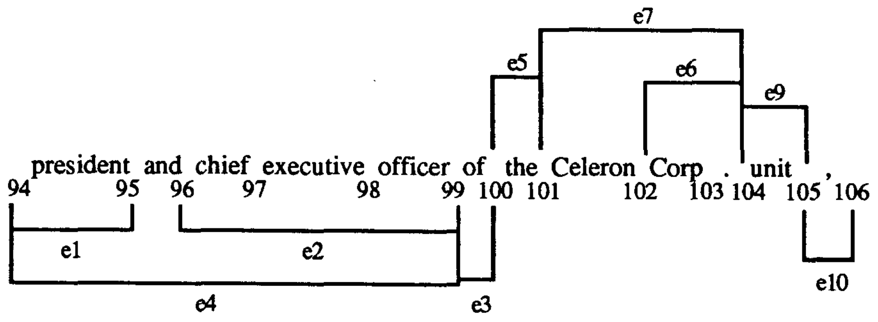
Sparser supports rules with more than two righthand side terms by converting them to a kind of Chomsky Normal Form using a dotted rule rule convention as described by Martin, Church & Patil (1981).

4.4 The parsing algorithm

The phrase structure algorithm divides logically into three processes: (1) delimiting the next segment, (2) parsing the new edges within that segment, and (3) parsing edges across segment boundaries. The control structure treats these as independent processes that signal events, and switches between them as the events dictate. We describe each of these processes in turn, using as our example the portion of the example article excerpted below.

... president and chief executive officer of the Celeron Corp. unit, a holding company for Goodyear's All American Pipeline.

The notion of a "segment" in SPARSER is a sequence of terminals between a matching set of phrase boundary brackets that are introduced into the chart by closed class words or by known open-class words from the domain vocabulary.


e7
e5
e6
e9
president and chief executive officer of the Celeron Corp . unit ,
94 95 96 97 98 99 100 101 102 103 104 105 106
e1
e2
e4
e3
e10

Just below is the excerpt with its brackets. The initial bracket was introduced by the just-preceding known verb, "become", the other brackets were introduced by the function words/punctuation/affixes: "and", "of", "the", ",", "a", "for", "'s", and ".".

[*president*] *and* [*chief executive officer*] *of* [*the Celeron Corp. unit*] , [*a holding company*] *for* [*Goodyear*] *'s* [*All American Pipeline*] .]

The idea of segmenting a text on the basis of its closed class words is an old one. A recent, comparably systematic system where closed class words are used is described by O'Shaughnessy (1989). And it appears that something like this scheme is used in Hindle's FIDDITCH parser (partially described in Hindle 1983).

The segment delimiter starts at the last position where a segment terminated (or initially at position 0). It makes successive calls to Scan, adding words and their immediate pre-terminal edges to the chart and running any of the non-phrase structure parsing processes that the words trigger. This processes stops when a word is scanned that introduces a close bracket ("]"). At this point control is passed the second process, to form whatever constituents may be found within the new segment by looking for combinations of the pre-terminal edges.

When using a normal "all edges" bottom-up algorithm, the criteria for which of the many trees to select is usually to choose the combination that provides the longest consistent account of the text and strands the fewest unattached edges. We mimic that selection criteria online, by having the parser first respect the linguistically motivated boundaries provided by closed class and other known words—parsing within a segment before combining any edges across a segment boundary. And second by respecting the possibility of that the rightmost edge in a segment may be extended by some not-yet-formed edge to its right in the adjacent segments—the algorithm does not allow a rightmost edge to be combined with an edge to its left if the resulting edge would not have the same label and consequently does not have the same possibilities for rightward extensions.

In terms of the interaction of the three processes, this means first that within-segment parsing is constrained not to permit any combinations of the segment's rightmost edge and its immediate neighbor edge to its left if that would change the possibilities for extending that rightmost edge later through a combination with some edge to its right. (This is a trivial check against the grammar tables.)

Once the within-segment parsing has finished, the resulting rightmost edge is similarly examined: If it permits rightward combinations then we return to the segment-delimiting process, and from that to the within-segment parsing process. Once there is finally a segment whose rightmost edge does not have a possible rightward extension, then the across-segment parsing process is allowed to start operating, beginning with the then rightmost edge in the chart overall.[4] As this third process moves leftwards forming successively larger edges, the possibility of rightward extensions is continually checked for, and the segment-delimiting process re-entered as needed.

We can see this control structure loop in action by walking through the excerpted text. Let us assume that have reached the point where the segment containing "*the Celeron Corp. unit*" has just been delimited. The chart will be as shown below. Positions are indicated by their index numbers between and below each of the words. Edges are indicated by half rectangles connecting the positions. The numbers on the edges (e.g. "e1") are for expository purposes only; they reflect the order in which each edge was introduced. The edge labels are not shown. For clarity the edges in the just delimited segment are shown above the text, and those of earlier and later segments below.

The within-segment parsing process will look for combinations of the edges between position 100 and 105, working rightwards from edge9. Edge9, the preterminal edge over the word "*unit*", is labeled "head-of-subsidiary-phrase" in this grammar. There are no rules in the grammar that would extend that edge into a larger edge to its right, and so the process is allowed to look for leftward combinations.

There are two edges adjacent to the left of edge9. Following the restricted search space convention of the algorithm, only the more recent of these, edge7, is checked. According to rule number two of the set listed earlier, edge7 and edge9 combine to form a new edge, which will then combined with edge5 according to rule three. There is now one edge spanning all of the segment; if there was a gap, say due to the presence of unknown words in the segment, then heuristic rules would be attempted, as briefly mentioned at the end of §1.1.

4 In some cases this can mean that an entire sentence is scanned before any across-segment edges are formed. This assumes, of course, that one does not write a grammar rule where period is not the left term of some rule, in which case the scan would continue.

The new topmost edge over the segment, labeled "subsidiary-company", does participate in rules that could combine it with an edge to its right, and so the delimiting process is resumed to scan until the next segment is terminated. That segment will contain the words "*a holding company*". Within-segment parsing will span the segment with a phrase labeled "company-description", which in the present grammar takes rightwards extensions and so the the delimiting process is run again. This iterates until the period after "pipeline" is reached, at which point across-segment parsing is finally begun. It rolls up the accumulated edges one after the other from the right. The penultimate composition in this example is the title phrase (edge4) and a "subsidiary-company" phrase spanning all the way from position 99 to position 115 just before the period.

5. Conclusions: Why is this efficient ?

Given two parsers that employ the same algorithms, the more efficient one will be the one with the most carefully designed and optimized implementation. The two parsers will carry out the same steps (at the algorithmic level), but one will do them more quickly, consuming less storage, etc. From this mechanical point of view Sparser comes off well as compared with other parsing systems that the author is familiar with: A Lisp program, it uses only preallocated storage, which led to a three-fold increase in speed relative to its prior implementation.

Holding the quality of the implementation constant (and of course the choice of machine on which any tests are made), the greatest increase in efficiency comes from improving the algorithm so that fewer steps are taken. We achieved this in two ways.

First, we employed a particular technique for reducing the search space through which the parser searched, thereby reducing the number of checks make against the grammar to see whether two edge could be combined, and also reducing the number of edges ever entered into the chart. While we have not yet made a systematic comparison, this technique of checking only the topmost edges at a position appears to result in three to ten times fewer edges ever being formed (depending on the article and the grammar) when compared to an earlier variant of Sparser's algorithm that checking all of the edges.

Second, we employed a grammar with semantically labeled terms, thereby ensuring that only edges that could receive a valid semantic interpretation would ever be formed. This does not cut down on the number of edges checked against the grammar, but it has a dramatic effect on the number of edges ever allowed to be formed in the first place. While again we do not have systematic counts (which would effectively require having an entirely new grammar that used only syntactic labels), our impression is that the reduction in ambiguity that the semantic labels brought about had a more significant effect on the number of edges and checks than any variation in the algorithm.

6. References

Aho, Alfred V. & Ullman, Jeffrey D. (1972) *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall.

Earley, J. (1970) "An Efficient Context-Free Parsing Algorithm", Communications of the ACM, 13(2), February 1970.

Hindle, Don (1983) "Deterministic Parsing of Syntactic Non-fluencies", Proc. 21st Annual Meeting of the Association for Computational Linguistics, June 15-17 1983, MIT, pp. 123-128.

Kaplan, Ronald M. (1973) "A General Syntactic Parser" in Rustin (ed.) *Natural Language Processing,* Algorithmics Press, New York, pp.193-242.

Kay, Martin (1980) "Algorithm Schemata and Data Structures in Syntactic Processing", Xerox PARC Technical Report CSL-80-12; reprinted in Grosz, Sparck Jones and Webber (eds) *Readings in Natural Language Processing,* Morgan Kaufmann. 1986.

Martin, William A., Kenneth W. Church, Ramesh S. Patil (1981) "Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results" MIT Laboratory for Computer Science Technical Report #261.

McDonald, David D. (1990) "Robust Partial-Parsing through Incremental, Multi-level Processing: rationales and biases", AAAI Spring Symposium paper reprinted in Jacobs (ed.) "Text-Based Intelligent Systems: Current Research in Text Analysis, Information Extraction, and Retrieval, GE R&D Center technical report 90CRD198, Schenectady, NY.

O'Shaughnessey, Douglas D. (1989) "Parsing with a Small Dictionary for Applications such as Text to Speech" Computational Linguistics 15(2), June 1989, pp.97-108.

Winograd, Terry (1983) *Language as a Cognitive Process,* Addison Wesley.