Chart Parsing

Doug Arnold
doug@essex.ac.uk

1 The Problems

- Inefficiency of backtracking parsers;
- Inadequacy of trees as representations for parsing when there are local ambiguities;
- Inadequacy of trees as representations for parsing when there are incomplete structures.

1.1 Parsing inefficiency

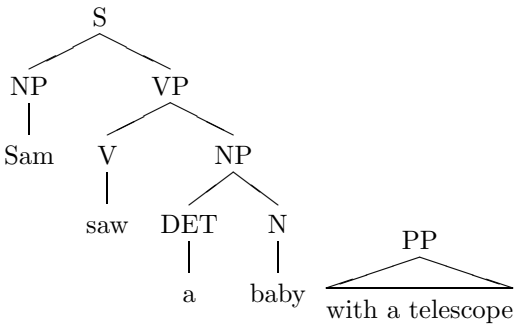
A backtracking parser may perform many redundant actions:

- rediscovering phrases it has already successfully found; and
- re–exploring hypotheses that have already failed.

(1) Have the [$_{NP}$ students that we talked about] take the exam.
(2) Have the [$_{NP}$ students that we talked about] taken the exam?
(3) The cherry blossoms [$_{PP}$ in the garden],
are lovely.
(4) The cherry blossoms [$_{PP}$ in the garden],
and a rose blooms by the house.
(5) Several cars raced [$_{PP}$ at the the most dangerous race track in the world]
were tested for problems.

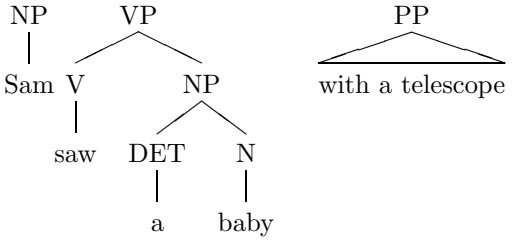
1.2 Inadequacy of Trees (1)

Trees do not provide a neat way of representing local ambiguity. The PP below may attach to the NP, the VP, or the S. Representing each possibility requires a separate tree. It is not possible to collapse the trees so as to share the common information:



1.3 Inadequacy of Trees (2)

Trees do not provide a neat way of representing incomplete structures (which may arise because the input cannot be completely parsed). The following is not a tree (no root):



The use of a chart offers three advantages:

- it avoids multiplication of effort;
- it provides a compact representation for 'local ambiguity';
- it provides a representation for 'partial parses'.

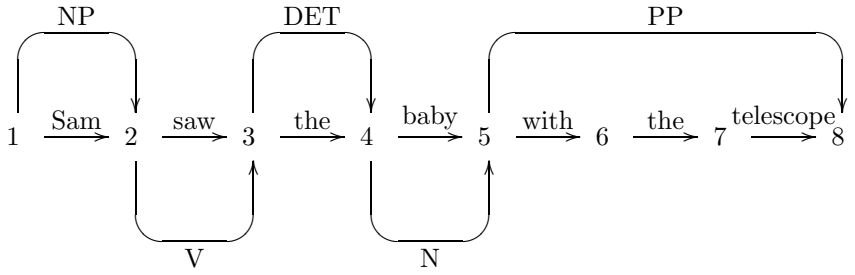
Moreover, *('active') chart parsing*, that is, parsing where the chart itself drives the parsing process, provides a general framework in which alternative parsing and search strategies can be compared.

2 Charts: Well-formed Sub-String Tables

A *chart* is a form of well-formed sub-string table (wfst).

It consists of a collection of *vertices*, one between each word of the input, connected by *edges*, labelled with grammatical information.

Chart 1



2.1 Avoiding duplication of work

As parts of the input are successfully parsed, they are entered in the chart. The idea is that before trying to parse any part of the input as (say) a PP, we look in the chart to see if we have parsed it as a PP before, (and if so, don't parse it again). Thus, the sub-string *with a telescope* is only parsed once as a PP.

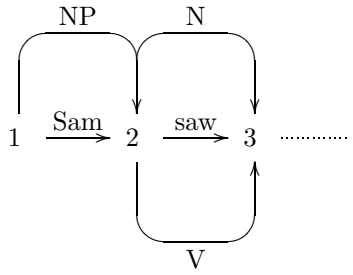
This technique is often called *memoization*.

2.2 Local Ambiguity

The chart provides a compact representation of local ambiguity:

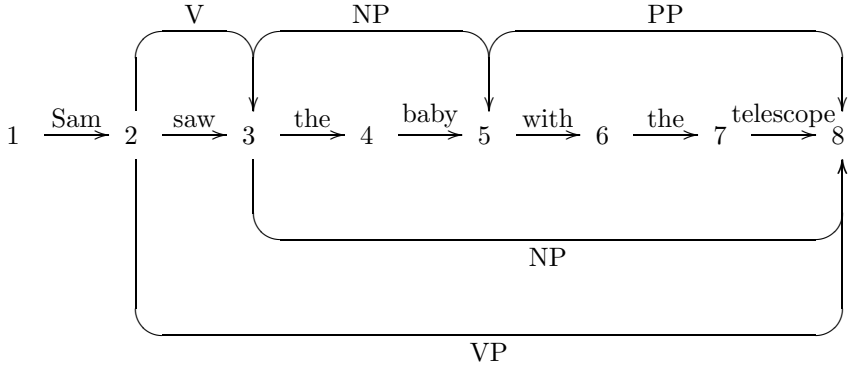
The N/V ambiguity of *saw*:

Chart 2



Notice the following chart contains a single VP edge from 2 to 8, expressing the possibility *saw the baby with the telescope* is a VP, abstracting away from the issue of whether the PP is a daughter of the VP or a daughter of the NP (i.e. whether the VPs structure is [V,NP,PP] or [V,NP]):

Chart 3



The Basic Principle: Avoid duplication: Represent everything, but only represent it *once*.

(Of course, this kind of 'packing' may be more difficult if entries in the chart contain more than information about categories, and start and end points of substrings).

2.3 Partial Parsing

Even if the overall parsing fails (i.e. one cannot get a single edge spanning the whole input — so one cannot build a representation of the whole input), the chart still contains useful information.

One could, for example, recover the smallest sequence of edges sufficient to cover the whole input (there may be several).

3 Formalization

Formally, Charts are directed graphs (i.e. like trees), consisting of:

- Vertices (cf nodes), connected by
- Edges (cf branches).
- Edges can be labelled.

In charts, unlike trees:

- Several edges may point to one vertex.
- Cycles are allowed.

4 (Active) Chart Parsing

An active chart parser consists of

- A chart, which may contain active (i.e. incomplete) edges.
- An agenda, which dictates an order in which tasks are carried out.

4.1 Active edges (Dotted rules)

Dotted rules are used to represent stages in the parsing process. They look like are normal PS rules, with a 'dot' somewhere in the rhs. Corresponding to the rule:

(6) $NP \rightarrow DET\ N\ PP$

There are the following dotted rules:

(7) a. $NP \rightarrow \circ\ DET\ N\ PP$
b. $NP \rightarrow DET \circ\ N\ PP$
c. $NP \rightarrow DET\ N \circ\ PP$
d. $NP \rightarrow DET\ N\ PP \circ$

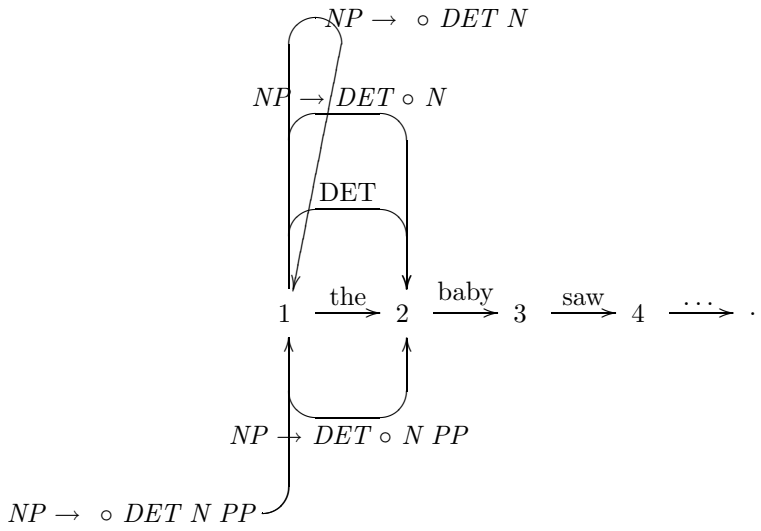
The part after the dot is called the *remainder*.

Intuitively, (8) indicates a situation where part of an NP has been found (the DET and the N), and a PP remains to be found to complete the NP.

(8) $NP \rightarrow DET\ N \circ\ PP$

Active edges are incomplete:

Chart 4



Notation:

`<StartV, EndV, Label, Found, Remainder, Representation>`

the `Label` corresponds to the lhs of the dotted rule, `Found` corresponds to the part before the dot, `Remainder` corresponds to the part after the dot, the `Representation` contains whatever information we want to record (parse tree, semantics, ...).

```
<1, 2, DET,  [], [], ...>     <1, 1, S,   [],  [NP,VP], ...>
<1, 2, the,  [], [], ...>     <1, 2, NP,  [],  [N], ...>
<2, 3, N,    [], [], ...>     <1, 2, NP,  [],  [N PP], ...>
<2, 3, baby, [], [], ...>
<3, 4, saw,  [], [], ...>
etc.
```

4.2 The Fundamental Rule

When an active edge meets an inactive edge:

if the first thing in the remainder of the active edge matches the label of the inactive edge, then create a new edge which:

- starts where the active edge started
- ends where the inactive edge ended
- has the same label as the active edge
- has the remainder of the active edge, minus the label of the inactive edge (which is moved 'before the dot' — into `Found`)
- and has an appropriate representation (a matter of taste).

But: do nothing twice — hence only add this edge if it is not already in the chart.

Given:

`<1, 2, DET, [], [], ...>`

```
<2, 3, N,   [], [], ...>
<1, 1, S,   [], [NP, VP], ...>
<1, 2, NP,  [DET], [N], ...>
<1, 2, NP,  [DET], [N PP], ...>
```

The Fundamental Rule:

```
<1, 2, NP, [DET], [N], ...>  +
<2, 3, N,  [], [], ...>           =>    <1, 3, NP, [DET, N], [], ...>

<1, 2, NP, [DET], [N, PP], ...>  +
<2, 3, N,  [], [], ...>           =>    <1, 3, NP, [DET, N] [PP], ...>
```

5 Processing

- Initialization ('scan', 'predict')
- Processing ('complete', 'scan', 'predict')

5.1 Initialization

5.1.1 Lexical Lookup ('scan')

For each item in the input, and each lexical entry of the item, add an edge that:

- starts before the item,
- ends after it,
- is labelled with the category of the item,
- has an emtpy remainder,
- has some representation

(Alternatively, we can interleave scanning with other operations).

5.1.2 Top down ('predict')

Choose a goal category (e.g. S)

- For every rule that has the goal category as its lhs, add an edge to the chart of the form:
  `<0, 0, GoalCat, RHS, []>`
  where RHS is the rhs of the rule: e.g.
  ```
  <0, 0, S, [], [NP, VP], [], ...>       % S -> . NP VP
  <0, 0, S, [], [NP, VP, PP], [], ...>   % S -> . NP VP PP
  <0, 0, S, [], [PP, NP, VP], [], ...>   % S -> . PP NP VP
  etc.
  ```
- For each edge added, take the first item in the remainder, and treat it as a goal category. In this way, given:
  `<0, 0, PP, [], [PP, NP, VP], [], ...>   % S -> . PP NP VP`
  we add the edge:
  `<0, 0, PP, [], [P, NP], [], ...>`

But: do nothing twice — only add edges if they are not already in the chart.

5.1.3 Bottom Up ('predict')

For each inactive edge that starts at 0:

- find every rule that has the label of the edge as its left corner (i.e. the first thing on its rhs)
- add an empty active edge correpsonding to that rule, i.e. an edge:
  - that goes from 0 to 0
  - whose label is the lhs of the rule
  - whose remainder is the rhs of the rule
- Repeat this process using the labels of the active edges so added

For example, given:

```
<0, 1, DET, [], [], ...>
```

We get:

```
<0, 0, NP, [], [DET N], ...>        % NP -> . DET N
<0, 1, S,  [], [NP, VP], ...>       % S ->  . NP VP
<0, 1, S,  [], [NP, VP, PP], ...>   % S ->  . NP VP PP
etc.
```

5.2 Parsing

Mostly this is just a question of invoking the fundamental rule (cf. 'completing').

However, there are times when we also need to look at grammar rules; and we need some way of deciding in what order to do various things.

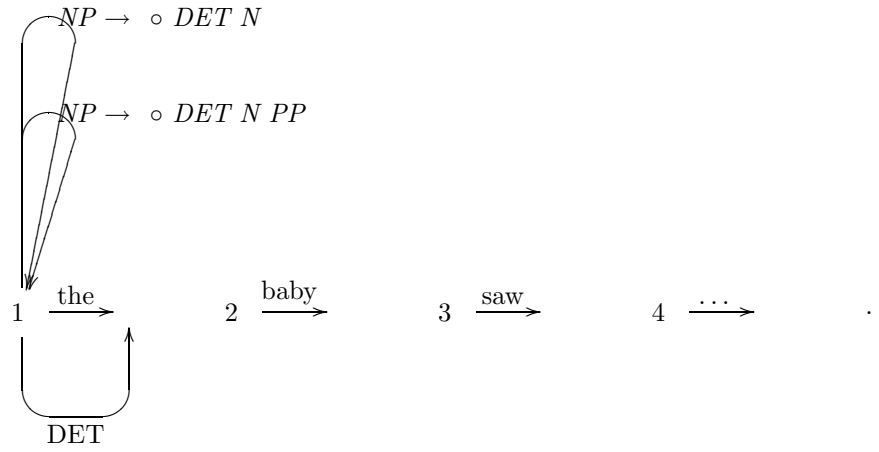
- The agenda determines in what order possibilities are tried (see below).
- When the result of combining two edges is an active edge, it will sometimes be necessary to 'predict': add further, new empty active edges to be added, to complete the parse. This will require reference to the grammar as when initializing. E.g. when the following edges are combined:

  ```
  <n, m,   S,  [],       [NP, VP], ...> +
  <m, m+1, NP, [DET, N], [], ...>
  =>
  <n, m+1, S, [NP], [VP], ...>
  ```

  The rules must be consulted again to add edges that will allow a VP to be parsed (Top-down or Bottom-up). The way we consult the rules determines whether the overall strategy is bottom up or top down.
  (If lexical look-up is done at initialization, this will only be necessary when the first thing in the remainder of the new active edge is not a lexical category. An alternative strategy would have us perform all possible 'predict' and 'complete' actions at given point, and then invoke 'scan' to add the next word of input to the chart).

5.3 The Agenda

Chart 5



- Do we try to extend `<1, 2, NP, [], [DET, N], ...>` before `<1, 2, NP, [], [DET, N, PP]>`, or vice versa?
- Suppose we extend `<2, 2, NP, [], [DET, N], ...>`, this gives a new edge `<2, 3, NP, [DET], [N], ...>`. Do we try to extend this, or do we try to extend `<2, 2, NP, [], [DET, N, PP]>`?

One view of the agenda is simply as a set of edges waiting to be added to the chart: when a new edge is created, rather than adding it to the chart immediately, we first put it on the agenda.

The agenda determines in what order edges are added to the chart.

Suppose tasks are removed from the front of the agenda.

- Stack agenda: every time an edge is added, it is placed on the front of the agenda.
- Queue agenda: every time an edge is added, it is placed on the end of the agenda.

The choice between these determines the search strategy (depth first vs. breadth first)

Stack organization gives depth first search; queue organization gives breadth first.

One attraction of chart parsing is that it provides a general framework in which alternative parsing strategies can be considered and compared.

6 Attractions

It provides a general framework in which alternative parsing strategies can be considered and compared.

- Flexibility
- A way of avoiding the problems of local ambiguity — no duplication of effort.
- No problem with left recursion, even top-down.

7 Disadvantages

Everything possible is done (much of it useless).

[This is an exaggeration: in parsing *The baby ...*, a chart parser working top down will add an edge corresponding to $NP \rightarrow \circ$ *ProperNoun*, which would not be added bottom up. Parsing *Flies are a nuisance*, a bottom up parser may add edges corresponding to $VP \rightarrow \circ V...$, because *flies* can be a verb, a top down parser might not add such edges, because sentences cannot begin with a finite verb.]

8 Variations, Extensions

- Middle Out Parsing;
- Mixtures of Top-down and Bottom-up;
- 'Packing' so as to share information among edges, and reduce duplication further.

9 Reading

Many introductory books contain good discussion of chart parsing, e.g. Gazdar and Mellish (1989), Winograd (1983).

The invention of chart parsing as an approach is generally credited to Martin Kay; Kay (1980, 1986) discusses it as a basis for comparing different parsing schemes. It bears more than a passing similarity to Earley's Algorithm for parsing (the efficiency of Earley's algorithm arises from its use of what is in essence a chart Earley (1970, 1986). Pereira and Warren's work on 'Earley deduction' Pereira and Warren (1983) is a generalization of chart parsing that gets away from the particular datastructure of a chart.

Beyond this, there is a considerable literature on variations and applications of chart parsing.

References

Jay Earley. An efficient context-free parsing algorithm. In Grosz et al. (1986), pages 25–33.

G. Gazdar and C. Mellish. *Natural Language Processing in Prolog.* Addison Wesley, Wokingham, 1989.

Barbara J. Grosz, Karen Sparck-Jones, and Bonnie Lynn Webber, editors. *Readings in Natural Language Processing.* Morgan Kaufmann, Los Altos, 1986.

Martin Kay. Algorithm schemata and data structures in syntactic processing. In Grosz et al. (1986), pages 35–70.

Fernando C. N. Pereira and David Warren. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, MIT, Cambridge, Massachusetts, 1983.

T. Winograd. *Language as a Cognitive Process.* Addison Welesley, Reading, Mass., 1983.