5/11/87

Chart Parsing: Part II

1. Resume

Last lecture we saw how to build all well-formed constituents over an input string of length n licensed by a (restricted form) context-free grammar using an algorithm that has a time complexity $O(n^3)$ and space complexity $O(n^2)$.

These constituents were built up in a well-formed substring table or chart. The latter can be represented by a graph in which the vertices represent the positions between the words and there is an edge from vertex i to vertex j labelled C if the string of words between these points can be analysed as a constituent of category C according to the grammar.

To remove the restrictions on the form of the grammar, the notion of dotted rule was introduced. A dotted rule is a type of edge that represents a constituent that is not well-formed, or complete, but rather in the process of being built. It still lacks certain items, those which occur after the dot. When these have been found, there will be a complete consituent of the category of the lhs of the dotted rule. Since this type of edge contains within it a specification of what needs to occur after it, and thus can be given an active role in the parsing process, it is known as an **active edge**. A parser that uses a chart including active edges is known as an **active chart parser**. An edge with no indication of its start and finish position is called a **state.**

2. Top-down aspects of chart parsing

The CKY parser is essentially bottom-up, so that constituents are built regardless of the possibility of their incorporation into larger phrases. We can incorporate the dotted rule idea into a bottom-up parser. We must add an initial dotted rule for each rule in the grammar to each entry along the diagonal of the chart. The diagonal of the chart represents zero length phrases, phrases that span from vertex i to vertex i for any i. So these initial dotted rules correspond to constituents of which no sub-consituents have yet been found. They can be thought of as hypotheses. The dotted rule (in Winograd's notation):

$({}_4\,\mathrm{vp} \rightarrow_4 \mathrm{v\ np\ pp}\,)$

embodies the hypothesis that there is a vp starting at position 4 that will include a v, np and pp, though none of these have yet been found.

In entering initial dotted rules of every type at every point, we have been rather lavish with our hypotheses. In fact, at the start of parsing, we have only a single reasonable hypothesis. This is the characteristic hypothesis of top-down parsers, that there is a sentence starting at position 0. Actually, we need a hypothesis for each way of building a sentence.

Since it will be the active edges that dictate the course of parsing, and the only active edges that are in the initial state of the chart are those that we know will be useful, the number of useless constituents that are built will be greatly reduced from those built by the CKY algorithm.

5/4/87

3. Earley's algorithm

Earley defined his algorithm in terms of three cases that can obtain when an edge is put into the chart. These are as follows:

(1) Predicting: The edge that is being entered is active, but there is no edge for it to combine with already in the chart. In this case, the required edge is predicted, by entering an initial dotted rule for each way that it can be built.

(2) Scanning: The edge that is being entered is active, and there are edges already in the chart with which it can combine. For each of these edges the multiplication rule is applied. This gives rise to further edges.

(3) Completing: The edge that is being entered is complete. For each of the active edges that it can combine with, the multiplication rule is applied, giving rise to further edges. Note that there must be such active edges, for the complete edge to have been predicted in the first place.

When a number of new edges are created by one of these operations, there is some choice about how we deal with them. We can add each of them before we add any other edge. Alternatively, we can add the first and then any new edges that that operation gives rise to, before considering the rest. The first of these approaches gives rise to a breadth first strategy and the second to a depth first one. This point is developed further below.

Another choice is how we start the chart off. Do we predict the distinguished symbol or do we add the lexical edges first. In the former strategy, we will only enter a complete edge after we have made all predictions up to the start of that edge. This means that the conditions for scanning will never arise. All non-diagonal edges will arise as a result of completing. Hence we will only access the chart by end vertex, retrieving the set of edges that end at a certain point. If efficiency was a primary concern, this possiblity might be worth exploiting. In any case, this will be how the algorithm will be presented.

For the sake of exposition we will assume that the chart is a three-dimensional matrix of booleans. The second and third dimensions are the start and finish vertices of edges, and the first dimension ranges over the possible states. In a grammar with rules:

s → np vp
np → det n
vp → v np np

the possible states are:

.s	s → np . vp	s.		
.np	np → det . n	np.		
.vp	vp → v . np np	vp → v np . np	vp.	
.det	det.	.n	n.	.v	v.

```
procedure chartparse
  add_chart({.s},0,0)
  for j <- 1 to n do
    for state ∈ {A | A → $word_j$ } do
      enter_edge(state,j,j-1)

procedure enter_edge(state1,i,j)
  if chart(state1,i,j) = false then
    chart(state1,i,j) <- true
```

5/11/87

```
    for state2 ∈ right_sisters_of(state1) do
        enter_edge(initial_state_of(state2),j,j)
    complete(state1,i,j)

procedure complete(state1,k,j)
  for state2 ∈ left_sisters_of(state1) do
    for i in chart(state2,i,k) do
        enter_edge(state2*state1,i,j)
```

Note that before we enter an edge, we must check that an identical edge has not been entered before. Although this algorithm is top-down, doing only what is necessary, this occurs check avoids the problem that the Prolog interpreter has with left- recursive grammars.

The rest of procedure enter_edge is the heart of the algorithm. If the edge that we are entering is an active one, we find the constituents that it could combine with to its right by calling right_sisters_of. Then we predict their initial states. If the edge is complete, then right_sisters_of will just return the empty set, and procedure complete will be called directly.

Complete will find the constituents that an edge could combine with to its left by calling left_sisters_of. If the edge is active, this will be the empty set. If the edge is complete, however, then each of the appropriate active edges that end at the start of the complete one will be combined with it.

4. The Agenda

We can make this algorithm somewhat more flexible by the use of a data structure called an agenda. Instead of entering an edge directly into the chart when we have constructed it (by completing) or determined that it is useful (in the prediction step), we add it to the agenda. We start the algorithm off by putting the initial edges (the lexical edges and (.s,0,0)) on the agenda. Then we continue to perform the loop:

```
while there are edges on the agenda do
  choose an edge from the agenda
  enter the edge into the chart
```

The rest of the algorithm is the same as before, except that the calls to enter_edge from enter_edge and complete are replaced by calls to add the edge to the agenda.

Now we can change the order in which things get done merely by adopting different strategies for choosing an edge from the agenda. If the agenda is treated as a stack, so that the last edge added to it is the first one to be chosen for entering in the chart, then we will be realising a depth-first parser. On the other hand, if the agenda is a queue, then the strategy will be breadth-first. We can even adopt more sophisticated criteria for choosing an edge from the agenda, such as choosing the edge which progresses furthest through the sentence (with the highest numbered end vertex).

There are other variations on the basic algorithm, such as adding lookahead. In general, active chart parsing provides a framework in which all sorts of ideas about efficient and/or psychologically real parsing strategies can be explored.