Distributed Memory: A Basis for Chart Parsing

Jon M. Slack

Human Cognition Research Laboratory
Open University
Milton Keynes, MK7 6AA
ENGLAND

Abstract

The properties of distributed representations and memory systems are explored as a potential basis for non-deterministic parsing mechanisms. The structure of a distributed chart parsing representation is outlined. Such a representation encodes both immediate-dominance and terminal projection information on a single composite memory vector. A parsing architecture is described which uses a permanent store of context-free rule patterns encoded as split composite vectors, and two interacting working memory units. These latter two units encode vectors which correspond to the active and inactive edges of an active chart parsing scheme. This type of virtual parsing mechanism is compatible with both a macro-level implementation based on standard sequential processing and a micro-level implementation using a massively parallel architecture.

A lot of recent research has focused on the problem of building psychologically feasible models of natural language comprehension. Much of this work has been based on the connectionist paradigm of Feldman and Ballard (1982) and other massively parallel architectures (Fahlman, Hinton and Sejnowski, 1983). For example, Waltz and Pollack (1984) have devised a model of word-sense and syntactic disambiguation, and Cottrell (1985) has proposed a neural network style model of parsing. Originally, such systems were limited in thier capability to handle tasks involving rule-based processing. For example, the Waltz and Pollack model uses the output of a conventional chart parser to derive a structure for the syntactic parse of a sentence. However, more recent connectionist parsing systems (e.g., Selman and Hirst, 1985) are more suited to handling sequential rule-based parsing. In this type of model grammar rules are represented by means of a small set of connectionist primitives. The syntactic categories of the grammar are represented in a localist manner, that is, by a computational unit in a network. The interconnections of the units within the network are determined by the grammar rules. Such localised representations are obviously useful in the construction of connectionist models of rule-based processing, but they suffer from an inherent capacity limitation and are usually non-adaptive.

The research to be discussed here differs from previous work in that it explores the properties of *distributed representations* as a basis for constructing parallel parsing architectures. Rather than being represented by localised networks of processing units, the grammar rules are encoded as patterns which have their effect through simple, yet well-specified forms of interaction. The aim of the research is to devise a virtual machine for parsing context-free languages based on the mutual interaction of relatively simple memory components.

1. DISTRIBUTED MEMORY SYSTEMS

Constraints on space make it impossible to describe distributed memory systems in detail, and this section merely outlines their salient properties. For a more detailed description of their properties and operation see Slack (1984a, b).

In distributed memory systems items of information are encoded as k-element *vectors*, where each element ranges over some specified interval, such as [-1,+1]. Such vectors might correspond to the pattern of activity over a set of neurons, or the pattern of activation levels of a collection of processing units within a connectionist model. Irrespective of the manifest form of vectors, the information they encode can be manipulated by means of three basic operations; *association* (denoted by *), *concatenation* (+), and *retrieval* (#). The association operator creates a composite memory vector which encodes the association of two items of information (denoting memory vectors by angular brackets, the association of items <A> and <B> is denoted <A>*<B>). The concatenation operator creates a composite memory vector encoding individual items or vectors (<A>+<B>). Thus, a single composite vector can encode large numbers of items, and associations between items. The individual elements encoded on a composite vector are accessible through the retrieval operator. When a *retrieval key vector* is input to a composite vector the retrieval operator produces a new composite vector which encodes those items of information which were associated with the key vector on the original composite trace. An important property of the retrieval and assocaition operators is that they are both *distributive* over concatenation. However, only the association operator is associative.

These basic properties enable distributive memory systems to encode large amounts of knowledge on a single composite memory vector. The capacity limitations of such systems are determined by the noise levels implicit in the output of the retrieval operation. The generated noise is a function of the number of traces encoded on a vector and the size of the vector.

Two classes of distributive memory can be distinguished, *permanent* and *working*. The former provide permanent storage of information, while the latter have no permanent contents but are used to build temporary representations of inputs. An important feature of working distributed memories is that the traces encoded on them decay. Associated with each vector encoded on a working memory is a decay, or strength, index. The decay function is event dependent, rather than time dependent, in that the indices are adjusted on the occurence of a new input. This property provides a useful form of event indexing which is crucial to the chart parsing scheme.

2. DISTRIBUTED CHART PARSING REPRESENTATION

The aim of chart parsing using distributed representations is to derive a composite memory vector which encodes the type of structural information shown in figure 1. The figure shows the constituent structure (C-structure) built for the sentence *The man hit the boy.*

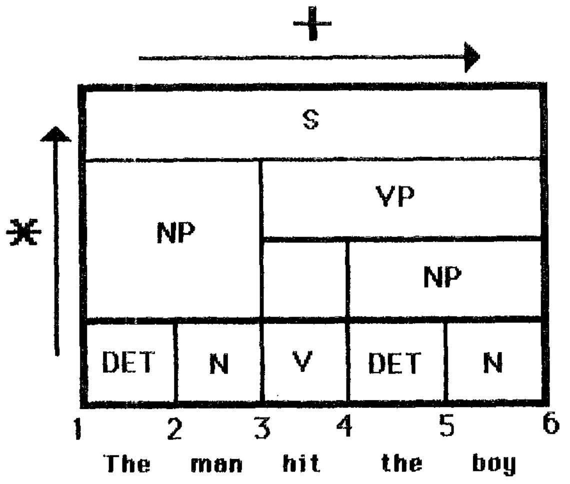

Figure 1

This structure incorporates features of different notations used to describe chart parsing schemes (Earley, 1970; Kay, 1980). The structure represents a passive lattice consisting of one cell, or an equivalent edge, per constituent. It embodies structural information relating to the immediate-dominance of constituents, as well as information about the range of terminals spanned by each constituent. This structure can be collapsed under the operations of association and concatenation to derive a composite memory vector representation which preserves both types of structural information. Each constituent category is encoded as a random memory vector; <S>, <NP>, <DET> and so on. The vertices of the chart (1-6 in figure 1) are mapped onto the set of decay indices registered by the working memory systems at each input point. Moving from left to right through the table shown in figure 1 the vectors are combined by the concatenation operator. The dominance structure exhibited within the table is preserved by the association operator going from bottom to top. These procedures produce the following composite vector which encodes the information represented in figure 1:

<S>*<1-6>*{<NP>*<1-3>*{<DET>*<1-2>+<N><2-3>}
+ <VP>*<3-6>*{<V>*<3-4> +
<NP>*<4-6>*{<DET>*<4-5>+<N>*<5-6>}}

Each component of the pattern consists of a vector encoding a constituent label associated with the input indices defining the left and right edges of the constituent. This form of representation is similar to the idea of assertion sets which have been shown to have useful properties as parsing representations (Barton and Berwick, 1985). One of the major advantages of the representation is that the two types of structural information are jointly accessible using the appropriate retrieval vector, that is category label vector. For example, if the composite vector is accessed using the retrieval vector <VP>, then the retrieved pattern encodes both the categories dominated by VP and the range of input terminals covered by VP.

3. PARSING ARCHITECTURE

A possible parsing architecture for realising the above scheme is shown in figure 2. The parsing mechanism consists of three distributed memory units, one permanent unit which stores context-free rule patterns, and two working memory units which encode temporary constituent structures. In figure 2, the *stored rules pattern* memory adds retrieved lists of rule constituents to the *active patterns* working memory. The double arrows on the lines connecting to the *inactive patterns* memory depict the two-way interactions between the inactive patterns and both the stored rule patterns and active patterns.

The input to the parsing mechanism comprises the syntactic categories of each constituent of the input string. This input is received by the inactive patterns working memory and each new input triggers the indexing of patterns in accordance with the decay function. Thus, while a category label vector is held on the inactive patterns unit its decay index is continually up-dated.

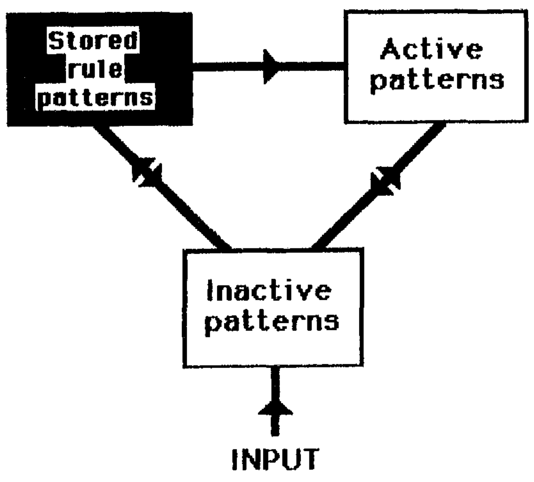

Figure 2

Representing the rules of the CFG by the following notation:

label: Constituent category ---> List of constituents

or

L_i:Cat_i ---> C_{ij}'s

Each element of a rule is encoded as a memory vector, and the rules are stored in the permanent memory unit in terms of the following patterns:

$<C_{i1}>*(<L_i>*<Cat_i>//<C_{i2}>*.......*<C_{ik}>*<L_i>)$

The vector $<C_{i1}>$ encodes the first constituent on the RHS of the CF rule labeled $<L_i>$. When this constituent is input to the rule patterns store the split pattern with which it is associated is retrieved as output. The retrieved pattern is split in that the two halves of the pattern are output over different lines. The first half of the pattern, $<L_i>*<Cat_i>$, is output to the inactive patterns unit, and the second half, $<C_{i2}>*....*<C_{ik}>*<L_i>$, is output to the active patterns memory unit. This retrieval process is equivalent to building an active edge in active chart parsing (Kay, 1980; Thompson, 1983). A major difference, however, is that the vector encoding the active edge is split over the two working memory units.

The active patterns unit now encodes the list of remaining constituents which must occur for the RHS of rule $<L_i>$ to be satisfied. Meanwhile, the inactive patterns unit encodes the category of the hypothesised, or active, edge. If the list of necessary constituents is matched by later inptus then the label of the satisfied rule is retrieved on the active patterns unit and output to the inactive patterns store. This new input retrieves the constituent category associated with the rule label on the inactive patterns unit. This retrieval process is equivalent to building an inactive edge in standard active chart parsing. Each time a new inactive edge pattern is created it is output to both the other units to determine whether it satisfies any of the constituents of a stored rule, or any remaining active patterns. Those active patterns which do not have their constituent elements matched rapidly decay, and fade out of the composite traces held on the two working memory units.

When a rule is satisfied and an inactive pattern, or edge, is built the index range spanned by the rule is retrieved at the same time as the rule's category vector. Thus, an inactive pattern is encoded as $<Cat_i>*<m\text{-}q>$, where $<m\text{-}q>$ encodes the range of input indices spanned by the category $<Cat_i>$.

Within this scheme the alternative CF rules for a particular constituent category have to be encoded as separate rule patterns. For example, the rules for building NPs would include the following:

1: NP ---> DET NP_2
2: NP ---> NP_2
3: NP ---> NP PP
4: NP_2 ---> N
5: NP_2 ---> ADJ NP_2

To simulate a top-down parsing scheme the active elements encoded on the inactive patterns unit can function as retrieval vectors to the stored rules unit. This feedback loop enables a retrieved category to retrieve further rule patterns; those for which it is the first constituent. In this way, all the rule patterns which are applicable at a particular point in the parse are retrieved and held as active edges split across the two working memory units. On each cylce of feedback the newly retrieved rule categories are associated with the active elements which retrieved them to form patterns such as $<L_a>*<S>*(<L_b>*<NP>)$.

On successfully parsing a sentence the inactive patterns unit holds a collection of inactive edge patterns which form a composite vector of the type described in section 2. All active patterns, and those inactive edges which are incompatible with the final parse rapidly decay leaving only the derived constituent structure in working memory. If an input sentence is syntactically ambiguous then all possible C-structures are encoded on the final composite vector. That is, as it stands the parsing scheme does not embody a theory of syntactic closure, or preference.

4. A PARSING EXAMPLE

The parsing architecture described above has been implemented on standard sequential processing machinery. To achieve this it was necessary to decide on computational functions for the three memory operators, and to build an agenda mechanism to schedule the ordering of the memory unit input/output processes. The association and retrieval operators were accurately simulated using *convolution* and *correlation* functions, respectively. The concatenation operator was simulated through a combination of *vector addition* and a *normalisation* function. All syntactic categories, rule labels, and input indices were encoded as randomly assigned 1000-element vectors.

The agenda mechanism imposes the following sequences on input-output operations:

[A] The input lines to the inactive patterns unit are processed in the order - constituent input, input from stored rule patterns unit, and finally, input from active patterns unit.

[B] The retrieval and output operations for the units are cycled through in the order - activation of stored rule patterns, including top-down feedback; matching of active patterns, and building of inactive edge patterns.

[C] The final operation is to increment the input indices, before accepting the next constituent input.

To illustrate the operation of the parsing mechanism it is useful to consider an example. As a simple example, assume that the stored rule patterns unit holds the following context free rules:

1: NP ---> DET NP_2	6: S ---> NP VP
2: NP ---> NP_2	7: VP ---> V
3: NP ---> NP PP	8: VP ---> V NP
4: NP_2 ---> N	9: VP ---> VP PP
5: NP_2 ---> ADJ NP_2	10: PP ---> Prep NP

In parsing the sentence *The old man the boats* the lexical ambiguity associated with the words *old* and *man* produces the input string -

<Det> <Adj+N> <N+V> <Det> <N>

The first input vector, <Det>, retrieves rule 1, setting up <NP_2> on the active patterns unit and <1>*<NP>*(<Det>) on the inactive patterns unit (the input indices have been omitted to simplify the notation). Through feedback, rules 3 and 6 are also retrieved creating the pattern <NP_2>*<1>+<PP>*<3>+<VP>*<6> on the active patterns unit, and the pattern <6>*<S>*((<1>*<NP>*(<Det>)+(<6>*<NP>*(<1>*<NP>*(<Det>))) on the inactive patterns unit.

On input of the second word which covers the categories *Adj* and *N*, the composite vector <Adj+N> retrieves rules 4 and 5, and through feedback rules 2, 3 and 6. The <N> pattern component of the input vector satisfies the <1>*<NP_2> pattern held on the active patterns unit which leads to the creation of the first inactive edge pattern, <6>*<S>*(<NP>*(<Det>+<N>)).

The third input is also ambiguous and triggers a large number of rules, including rules 7 and 8. Again, the <N> pattern in the input vector triggers the construction of an NP inactive pattern -

<6>*<S>*(<NP>*(<Det>+<Adj>+<N>)).

In addition, the retrieval of rule 7 produces the inactive edge pattern <S>*(<NP>*(<Det>+<N>)+<VP>*<V>)), corresponding to a premature interpretation of the input as *The old(N) man(V)*. However, this pattern rapidly decays as new input is received.

The final two inputs set up another <NP> inactive edge which when output to the active patterns unit retrieves the label vector associated with the constituents of rule 8. This produces a <VP> inactive edge which through feedback to the active patterns unit retrieves the label vector <6>, as it completes the <S> rule. In turn, an inactive edge pattern for <S> is created.

To complete the parse, the period symbol, or corresponding vector, can be used to trigger an operation which refreshes only those inactive edges which contain the final <S> pattern. All other patterns decay rapidly leaving only the constituent structure pattern on the inactive patterns unit.

This parsing example is not concordant with most people's reading of the sentence *The old man the boats*. In the first instance, the sentence tends to be parsed as two consecutive NPs, *(The old man)(the boats)*, Such phenomena would tend to imply that the human parser is more deterministic then the present model suggests. However, the model is not a complete comprehension mechanism and other factors influence the final outcome of the parse. In particular, many of the phenomena associated with closure and ambiguity can be explained in terms of lexical items having preferred interpretations (Kaplan and Bresnan, 1982). In the present example, the preferred categories for *old* and *man* are *Adj* and *N*, respectively. Such an idea is easily accommodated within the present scheme in terms of the concept of *pattern strength* (see Slack, 1984b, for details).

5. MACRO- AND MICRO-LEVEL DESCRIPTIONS

The parsing architecture described in section 3 specifies a virtual machine which is compatible with both traditional sequential processing based on Von Neumann architecture, and massively parallel processing using architectures based on connectionist principles. The parsing scheme outlined in the last section represents a *macro*-level implementation of the distributed representation parsing mechanism. The memory operators and agenda control structures that determine the sequence of states of the mechanism are well-specified, and the parallelism implicit in the system is buried deep in the representation. However, the concept of a memory vector, and its theoretical operators, can also be mapped onto connectionist concepts to provide a *micro*-level implementation of the system.

A connectionist system would employ three layers, or collections, of elementary processing units. These sets of units correspond to the three distributed memory units, and the gross connections between them would be the same. Within each set of units an item of information is encoded as a distributed representation, that is, as a different pattern of activation over the units. The memory operations are modeled in terms of excitatory and inhibitory processes between units of the same and different layers. As with all connectionist models, it is necessary to delineate the principles which determine the settings of the input weights and unit thresholds. However, no global sequencing mechanism is required as the activation processes have their influence over a number of interative cycles. Each new input pattern stimulates the activity of the system, and through their mutual interactions the three sets of units gradually converge on an equilibrium state. Providing the connection weights and unit thresholds are set correctly, the two working memory layers should encode the types of active and inactive patterns described previously.

The major advantage of the distributed representation parsing scheme is that it obviates the need for special-purpose binding units as used in other connectionist parsing systems (Selman and Hirst, 1985; Cottrell, 1985). The function of such units can be achieved through the appropriate use of the association operator.

6. TOWARD A COMPLETE LANGUAGE INPUT ANALYSER

The main goal of this research is to establish the relationship between language understanding and memory. This involves building a complete language analysis system based on a homogenous set of memory processing principles. These principles would seem to include, (a) the interaction of mutual constraints, (b) a high degree of parallel processing, and (c) the necessity for distributed representations to facilitate the simple concatenation of multiple constraints. Such principles need to be matched against linguistic models and constructs to derive the most integrated account of linguistic phenomena.

Within these goals, the present chart parsing scheme is not allied to any particular grammatical formalism. However, distributed memory systems have previously been used as the basis for a parsing mechanism based on the principles of *lexical functional grammar* (Slack, 1984a). Originally, this system incorporated a conventional CFG module, but this can now be simulated using distributed memory systems. Thus, the processing underlying the language analyser is based on a more homogenous set of principles, and more importantly, memory principles.

As a component of the language analyser the CFG module generates a constituent structure comprising an augmented phrase-structure tree. This structure is derived using augmented CF rules of the form:

$$6:\ S \longrightarrow \underset{(\uparrow \text{Subj}=\downarrow)}{NP}\quad VP$$

These augmented rules are easily assimilated into the present chart parsing scheme as follows: The functional equations {e.g., ($\uparrow$Subj=$\downarrow$)} are replaced by grammatical function vectors; <SUBJ>, <OBJ>, and so on. These vectors are encoded on the stored CF rule patterns as a third form of output associated with sub-patterns such as (<6>*<S>*<NP>). The function vectors are output to a working distributive memory system which encodes the functional structure of an input sentence. As long as its associated sub-pattern is active within the CFG module the appropriate function vector will be output. This means that a particular grammatical function vector will only be active in the system while the rule and constituent with which it is associated are also active.

7. REFERENCES

Barton, G.E. and **Berwick, R.C.** (1985) Parsing with assertion sets and information monotonicity. In *Proceedings of IJCAI-85*, Los Angeles.

Cottrell, G.W. (1985) Connectionist Parsing. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, Irvine, California, 201-212.

Earley, J. (1970) An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery, 13,* 94-102.

Fahlman, S.E., Hinton, G.E., and **Sejnowski, T.J.** (1983) Massively parallel architectures for AI: NETL, Thistle, and Boltzmann machines. In *Proceedings of AAAI*, Washington, 109-113.

Feldman, J.A. and **Ballard, D.** (1982) Connectionist models and their properties. *Cognitive Science, 6,* 205-254.

Kaplan, R. and **Bresnan, J.** (1982) Lexical Functional Grammar: A formal system for grammatical representation. In Bresnan, J. (ed.), *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Mass..

Kay, M (1980) Algorithm schemata and data structures in syntactic processing. In *Proceedings of the Symposium on Text Processing.* Nobel Academy.

Pollack, J. and **Waltz, D.** (1985) Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science, 9,* 51-74.

Selman, B. and **Hirst, G.** (1985) A rule-based connectionist parsing system. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society,* Irvine, California.

Slack, J.M. (1984a) A parsing architecture based on distributed memory machines. In *Proceedings of COLING-84,* Stanford.

Slack, J.M. (1984b) The role of distributed memory in natural language parsing. In *Proceedings of ECAI-84,* Pisa.

Thompson, H.S. (1983) MCHART: A flexible, modular chart parsing system. In *Proceedings of AAAI,* Washington.