Connectionist learning of regular graph grammars

Peter Fletcher
School of Computing and Mathematics
Keele University
Keele
Staffordshire
ST5 5BG
U.K.
tel: 01782 733260 (UK), +44 1782 733260 (overseas)
fax: 01782 734268 (UK), +44 1782 734268 (overseas)
e-mail: p.fletcher@maths.keele.ac.uk

Running heading: Learning graph grammars.

Keywords: graph grammars, grammatical inference, parallel parsing, regular grammars, stochastic grammars, neural networks, symbol processing, unsupervised learning.

Published in 2001 in *Connection Science*, **13**, no. 2, 127–188.

Connectionist learning of regular graph grammars

Abstract

This paper presents a new connectionist approach to grammatical inference. Using only positive examples, the algorithm learns regular graph grammars, representing two-dimensional iterative structures drawn on a discrete Cartesian grid. This work is intended as a case study in connectionist symbol processing and geometric concept-formation.

A grammar is represented by a self-configuring connectionist network that is analogous to a transition diagram except that it can deal with graph grammars as easily as string grammars. Learning starts with a trivial grammar, expressing no grammatical knowledge, which is then refined, by a process of successive node splitting and merging, into a grammar adequate to describe the population of input patterns.

In conclusion, I argue that the connectionist style of computation is, in some ways, better suited than sequential computation to the task of representing and manipulating recursive structures.

1. Introduction

Connectionism is conventionally seen as standing in opposition to traditional symbol processing, where 'symbol processing' in this context means representing recursive structures and manipulating them according to their structural composition (Smolensky 1988, Fodor and Pylyshyn 1988, Garfield 1997). Connectionist networks and symbol processing systems are often considered to have complementary strengths and weaknesses (Horgan and Tienson 1996, Hadley 1999). Many attempts have been made to combine the virtues of connectionism and symbol processing in a single architecture (Hadley and Hayward 1997, Browne 1998, Hadley and Cardei 1999), but it turns out to be very difficult to mimic the systematic computational competences afforded by dynamic recursive data structures in the conventional kind of connectionist network, with its fixed architecture and weighted-sum activation functions (Haselager and van Rappard 1998, Marcus 1998, Phillips 1999); this is known as the 'variable-binding problem' (Barnden and Pollack 1991, Dinsmore 1992, Sougné 1998).

However, if one takes a broader view of connectionism (as advocated and formalised in Fletcher (2000)), these difficulties can be overcome; connectionist computation is in some ways *better* suited than sequential computation to symbolic processing. This paper is a case study to illustrate this proposition. It takes a traditional research topic from artificial intelligence, the learning of structured concepts from example patterns, and investigates how a connectionist network could solve it. The first step is to choose the problem domain, that is, the population of input patterns and the system of concepts available for the network to choose from. The problem domain must be very carefully chosen if the exercise is to be meaningful: the requirements for a suitable problem domain are as follows.

(1) The problem domain should be as simple as possible, so that one can understand and analyse the network's behaviour. However, it should not be so simple that a solution can be found by exhaustive or random search; the problem should be sufficiently non-trivial to allow a distinction to be drawn between intelligent and unintelligent solutions. Ideally the domain should enable us to state difficult problems in a few symbols. (Eventually we shall have to investigate how the network's performance scales with problem size, but there is no point in doing this until we have a network that behaves sensibly on small examples.)

(2) The concepts of the problem domain should involve some form of recursion or iteration, in view of the importance ascribed to recursive structures by the theoretical tradition based on the 'language of thought' hypothesis (Fodor 1975) and the 'physical symbol system' hypothesis (Newell and Simon 1976).

(3) The problem domain should have a rich intuitive structure, so that we can tell when a concept learned by the network is a significant discovery and when it is merely an accidental construction that happens to fit the data. Ideally we would have a formal criterion for evaluating the significance of the concepts found by the network, but at present we do not. Such a criterion may emerge as a *result* of artificial intelligence research when it is at a mature stage; it cannot be a *prerequisite* for research otherwise we could never get started. So at the present stage we must evaluate concepts on a case-by-case basis, using problem domains in which we can recognise which ones are significant.

The first two of these requirements lead me to *grammatical inference*, the problem of learning a grammar to represent a given set of sentences. Grammatical inference involves the learning of recursive structures and is a very difficult problem even for quite small grammars; the difficulty of the problem can be adjusted by imposing various restrictions on the grammars. Hence the requirements (1) and (2) are met. To meet requirement (3) I choose *geometric* patterns, line-drawings of two-dimensional shapes such as lattices, staircases and tessellations, drawn on a discrete image grid (see section 9 for examples). Geometry has a conceptually rich, highly organised theory, developed over thousands of years; geometric concepts also involve invariance under transformations such as translations, which is computationally very difficult to cope with even when the image grid is small. This makes it suitable from the point of view of all three requirements.

Hence the problem of learning to recognise simple geometric structures on a small image grid is ideal as a case study for symbol processing. A research programme to do this was outlined in Fletcher (1993), loosely inspired by Klein's Erlangen Programme (Tuller 1967), under which geometric concepts are to be learned in conjunction with their associated invariances, with topological concepts learned first, followed by affine concepts and then metric concepts. In this paper I shall only consider a small part of this programme: supposing we have already learned the relations of *horizontal* and *vertical* connection between neighbouring pixels in the image grid, the task is to express geometric shapes in terms of combinations of horizontal and vertical connections.

The geometric patterns will be represented as graphs, with edges labelled as horizontal or vertical. The networks used will be unsupervised self-configuring networks, in which grammatical knowledge is represented in the network structure, in a generalisation of the way that regular string grammars are represented using transition diagrams. The network is constructed during the course of learning, each learning step being driven by statistical correlations in the patterns (hence the grammar is stochastic). Since this is a connectionist approach, the network is not merely a knowledge representation but is also a computational system that performs its own parsing and learning.

This paper is the result of a complete rethink of my earlier work on self-configuring networks (Fletcher 1991, 1992); these networks were able to learn non-recursive grammars to represent hierarchical structure in the input patterns but were unable to handle recursion or geometric invariances. One important theme however will persist from the earlier work: the network needs a clear global semantics (so that one can speak of the 'correctness' of the knowledge represented in the network), and a clear division of labour between the nodes of the network (so that the local processing of each node is correct with respect to the global semantics). The semantics guides the learning process and avoids the need for a purely combinatorial *search* for an architecture that solves the problem.

The rest of this paper is organised as follows. Section 2 is a survey of other approaches to grammatical inference, in string and graph grammars, including connectionist approaches. Section 3 defines formally the class of networks to be used and the important concept of a *homomorphism* between networks, which is fundamental to the theory of parsing and learning; networks and homomorphisms are used to represent regular stochastic graph grammars and to state the parsing and learning problems. Section 4 describes the connectionist parsing algorithm. Section 5 describes informally how the network learns by successively *refining* its architecture: refinement is essentially a process of splitting nodes, and is defined formally in terms of homomorphisms; the algorithm for this is derived in section 6. The inverse of refinement is *merging*, a process described in section 7. The learning algorithm as a whole is stated in section 8, and example simulations are described in section 9. Conclusions are drawn in section 10.

2. Survey of related work

There is no previous work using connectionist networks to learn graph grammars, so I shall survey relevant work in three areas separately: (a) grammatical inference for string grammars, (b) graph grammars, and (c) connectionist grammatical processing.

Grammatical inference is the task of finding a grammar that generates a language, given as input 'positive' examples (that is, strings belonging to the language) and usually also 'negative' examples (strings not belonging to the language). The problem of inferring string grammars has been studied since the 1960s; see Fu and Booth (1975) for a survey of early work, and Sakakibara (1997) for more recent work. There are various ways of framing the grammatical inference problem, according to the input data and the criterion of success chosen. In Gold's (1967) model, the user supplies the algorithm with an infinite sequence of example strings; the algorithm guesses a grammar and successively improves it as further patterns arrive; the algorithm is considered successful if it eventually reaches a grammar that generates the language and it does not change the grammar thereafter. This formulation of the problem is called *identification in the limit*.

Valiant (1984) introduced a different learning problem, known as the *probably approximately correct* learning model. This was formulated originally for learning propositional functions but it is easily adapted to grammatical inference. The input is a finite sample of positive examples; in addition, the algorithm may nominate its own examples and ask whether they are in the language. From these data the algorithm must find a grammar that matches the language to any required degree of approximation. More precisely, given any real numbers δ and ε in $(0, 1)$ the algorithm must find, with probability at least $1 - \varepsilon$, a grammar with accuracy at least $1 - \delta$. The algorithm is required to halt (unlike Gold's, which runs for ever), within a time polynomial in $1/\delta$, $1/\varepsilon$, the size of the sample, and the size of the grammar.

A third version of the problem is due to Angluin (1988), who allows a more elaborate interaction between the algorithm and the environment. The algorithm attempts to identify the language L by asking questions of the form:

- is the string s in L?
- does the grammar G generate L?

The environment answers 'yes' or 'no'; in the case of a 'no' answer to the second question the environment also provides a counter-example, that is, a string that is in L but is not generated by G or that is generated by G but is not in L. The algorithm must find a grammar that generates L and then halt. This procedure is known as *learning with queries*.

A variety of classes of string grammar are used in grammatical inference. Most algorithms can only handle subclasses of the regular grammars, such as

- regular grammars corresponding to *deterministic* finite automata (Tomita 1982, Lang 1992, Mäkinen 1994),
- k-testable and k-piecewise testable languages (García and Vidal 1990, Ruiz and García 1996),
- terminal-distinguishable regular languages (Radhakrishnan and Nagaraja 1987).

Some work has been done with context-free grammars; however, in this case inference is so difficult that it is common to provide the algorithm with extra information in the form of structural descriptions (that is, unlabelled derivation trees) for the positive examples (Levy and Joshi 1978, Dányi 1993).

The problem of inferring *stochastic* regular grammars is roughly equivalent to that of inferring hidden Markov models. Gregor and Thomason (1996) describe a method for inferring non-recursive Markov models. Inferring Markov models with recursion is much harder, but given a model structure the transition probabilities can be estimated using the Forward-Backward algorithm (Sakakibara, 1997), a dynamic programming technique involving expectation maximisation; the Inside-Outside algorithm is a generalisation of this to context-free grammars (Lari and Young 1990).

Theoretical studies suggest that grammatical inference is a very hard problem. Gold (1967) showed that deterministic finite automata cannot be identified in the limit from only positive examples; using both positive and negative examples, the problem of finding a minimal deterministic finite automaton consistent with a given sample is NP-hard (Gold 1978, Angluin 1978). However, using Angluin's query learning, deterministic finite automata can be identified in polynomial time (Angluin 1987). Learning of context-free grammars is computationally hard even with query learning (Angluin and Kharitonov 1991). For further details of the theoretical background see Angluin (1992).

These pessimistic results led, Lucas (1993) suggests, to a 'general stagnation in the growth of new algorithms'. However, their relevance to the practicability of grammatical inference is debatable. Gold's and Angluin's learning models demand exact identification of the language, whereas in practice we might be satisfied with a grammar that approximately represents the language (Valiant's model of course allows for this). The theoretical results are based on worst-case performance, that is, inference of the most perverse grammar in the class, whereas in practice we might be more concerned with typical or naturally-occurring grammars. Lang (1992) demonstrates a clear-cut case in which worst-grammar performance is no guide to average-grammar performance, and identification to a very high degree of accuracy is much easier than perfect identification. In this paper I am concerned with identification in the limit of regular (graph) grammars from positive examples; the above considerations suggest that this is a difficult task but not necessarily a hopeless one, provided one does not adopt too perfectionist a standard of success.

Next we turn to a classification of learning algorithms for string grammars. Algorithms may be classified as *incremental*, where an initial grammar is constructed and then successively improved, or *non-incremental*, where a single grammar is produced (e.g. Bhattacharyya and Nagaraja 1993). I am primarily concerned with incremental methods; these may be subdivided into types, according to the way in which the grammar is modified at each step.

Hill-climbing methods (Tomita 1982) randomly mutate the grammar in search of one that performs better on the sample. There is no predictable theoretical relation between the languages generated by the mutated and the unmutated grammar; the mutation is simply accepted if it does not decrease the number of true positives minus the number of false positives. This method works quite well on small deterministic finite automata, but there is no explanation of why it works.

Enumerative methods work by adding production rules to the grammar in response to misclassification of example strings. Learning is thus treated as a process of acquiring more grammatical constructions as time goes on. The language therefore grows monotonically over time (Naumann and Schrepp 1993).

Merging methods also enlarge the language monotonically. They begin with a maximal grammar and successively merge pairs of non-terminals into one; learning is thus viewed as a process of *erasing grammatical distinctions* that are deemed to be insignificant. Fu and Booth (1975) construct a 'canonical definite finite-state grammar', representing a given positive sample of strings, and then form a 'derived grammar' by merging non-terminals. Lang (1992) and Corbí *et al.* (1993) construct a prefix-tree acceptor (a tree-like automaton that accepts precisely the positive examples) and then merge states to give a simpler automaton. By this method Lang is able to learn very large (500 state) deterministic finite automata to a very high degree of accuracy. It should be noted however that his automata contain very little recursion, due to the random way their connections are chosen; in essence, his method is concerned with classifying alternative substrings rather than recognising iteration. Merging methods have become very popular recently and have been applied to stochastic grammars (Stolcke and Omohundro 1994, de la Higuera 1998) and tree grammars (Carrasco *et al.* 1998).

An opposite approach to merging is *splitting*, in which one begins with a tiny grammar that can generate any string, and then refines it by a process of successively splitting a non-terminal (or a state of the corresponding automaton) into two. This is a process of *learning to make grammatical distinctions*. Grammatical knowledge increases over time, as each grammar contains all the knowledge of the earlier grammars; the language generated decreases over time. This method has the merit of avoiding the enormous space requirements of the merging method, where the initial automaton is similar in size to the entire string sample. It is also better suited to dealing with an infinite sequence of strings. Examples of this approach are Bell *et al.* (1990) and Ron *et al.* (1994), in which each non-terminal corresponds to a suffix of a string and splitting a non-terminal corresponds to extending the suffix.

A hybrid of splitting and merging could be expressed within the *version space* approach to learning (Mitchell 1978). A rudimentary example of this is Giordano's (1994) algorithm. A *lower set* of grammars is formed (each generating a subset of the language), together with an *upper set* of grammars (each generating a superset of the language). The grammars are refined through specialisation and generalisation operations, to converge on a correct grammar. The implemented algorithm just uses two specialisation operations, substitution of the non-terminal on the left-hand side of a production rule and deletion of a production rule, but clearly more operations could be added within this framework. Another hybrid algorithm is Dupont's (1996), which uses a combination of splitting and merging to adjust a regular grammar to each positive or negative example as it arrives; the grammar has to be kept consistent with all the patterns seen previously, so the algorithm has to store the sets of positive and negative examples seen so far (the positive examples being stored in the form of a prefix-tree acceptor).

In this paper I am thinking of grammatical inference as a process of concept formation, that is, progressive gain in knowledge by building on existing knowledge. Hence splitting operations are of most relevance, though merging will also occasionally be used to simplify the grammars produced. Only positive examples will be used, and past patterns will not be stored: only the current pattern will be available to the algorithm at each stage.

So far I have only considered string grammars. The fundamental ideas extend to graph grammars, although there is a variety of ways of generalising the concept of a production rule. The most general version is known as the *set-theoretic approach* or the *expression approach* (Nagl 1987): production rules are of the form $G \to H$, where G and H are graphs; to apply such a rule to a graph, an isomorphic copy of G is removed from the graph, together with all its incident edges, and a copy of H is inserted in its place, together with new edges linking it to the rest of the graph. The new edges are chosen according to an *embedding transformation*.

Common special cases of this are *node-label controlled* grammars, where G consists of a single node with a certain label (Englefriet and Rozenberg 1991), and *edge-label controlled* grammars, where G consists of an edge with a certain label and its two incident nodes (Main and Rozenberg 1987). In node-label and edge-label controlled grammars the embedding transformation is specified by a *connection relation*.

A theoretically more tractable framework is known as *hyperedge replacement*. Whereas an edge is incident to two nodes, a *hyperedge* is a more general entity that may be incident to any fixed number of nodes; a *hypergraph* is a generalisation of a graph consisting of nodes and hyperedges rather than nodes and edges. A production rule involves replacing one hyperedge by any hypergraph (with the same number of connecting points). These production rules may be used to generate languages of graphs (Drewes and Kreowski 1991) or languages of hypergraphs (Habel 1992). This type of grammar has many desirable theoretical properties that make it a natural generalisation of *context-free* string grammars (Habel 1992).

The desirable properties of hyperedge replacement grammars are abstracted and generalised further in the *algebraic* or *category-theoretic* approach (Ehrig 1979, Ehrig *et al.* 1991), in which application of production rules is expressed in an elegant way in terms of a pair of push-outs. (More recently, Bauderon (1996) has shown how to encode the connection relation of node-label controlled grammars using pullbacks.) A useful survey of recent extensions to all these types of graph grammar is provided by Rozenberg (1997).

The parsing problem is harder for graph grammars than string grammars. As with string grammars, the class of grammars needs to be restricted in order to make the parsing problem decidable (Ehrig 1979, Main and Rozenberg 1987). Bartsch-Spörl's (1983) parsing algorithm works by exhaustive search. Most other algorithms work by imposing a sequential ordering on the graph: *chain code* methods convert the graph into a string by traversing it (Lucas and Damper 1990); Bunke and Haller (1992) scan a plex structure, constructing all possible ways of deriving the part of the structure seen so far; Lichtblau (1991) 'sequentializes' the graph using an ordered spanning tree; Flasiński (1993) also assigns a linear ordering to the nodes as a preliminary to parsing. However, the imposition of a sequential ordering is somewhat artificial and seems ill-suited to graphs, whose structure is inherently non-one-dimensional. Brandenburg and Skodinis (1996) relax the condition of strict sequentiality by using *graph automata*, which scan the graph using several control heads (instead of one, as in a conventional finite automaton), thus introducing a degree of parallelism. In my algorithm (see section 4) I abandon the concept of traversal altogether, producing a fully parallel procedure in which the whole of the graph is parsed at once.

There are few known algorithms for inference of graph grammars. In terms of the classification of string grammars above, Bartsch-Spörl's (1983) is an enumerative method for a limited class of context-sensitive graph grammars, while Carrasco *et al.*'s (1998) is a merging method for regular tree grammars and Jeltsch and Kreowski's (1991) is a merging method for hyperedge replacement grammars. Jeltsch and Kreowski propose four operations for transforming grammars: INIT, which constructs a grammar to represent a given finite set of graphs; DECOMPOSE, which splits a production rule into smaller rules (without altering the language); RENAME, which renames nonterminals; and REDUCE, which removes redundant production rules. The RENAME operation is the key one since it may merge non-terminals and thereby enlarge the language generated. Jeltsch and Kreowski do not propose any way of determining the sequence of operations to derive a correct grammar from examples; their 'algorithm' is more a formal statement of the problem than a solution.

Turning now to *connectionist* grammatical processing, the problem of representing, parsing and learning grammars in a connectionist network involves additional computational constraints: the storage and processing of the patterns and grammars must be distributed across the network and subject to capacity bounds (Fletcher 2000). This rules out algorithms such as those of Lang (1992) and Jeltsch and Kreowski (1991), which begin by constructing a huge grammar similar in size to the entire pattern set.

Attempts have been made to apply a standard three-layer back-propagation network to inference of natural language grammar (Hanson and Kegl 1987). However, for the purpose of learning regular string grammars it is more natural to use recurrent networks (which are essentially trainable deterministic finite automata). Various architectures have been used: simple first-order recurrent networks (Elman 1990, Jordan 1988); more complex first-order networks (Williams and Zipser 1989, Fahlman 1991); and second-order recurrent networks (Giles *et al.* 1992). Elman (1992) has also applied recurrent networks to context-free grammars and found that they can represent up to about three levels of recursive embedding; other authors (Kwasny and Faisal 1990, Das *et al.* 1993, Zeng *et al.* 1994) deal with context-free grammars by using a neural network in conjunction with a stack, or by using a simple recurrent network to generate representation vectors for a recursive auto-associative memory (RAAM) (Reilly 1991).

The question arises of how to relate the internal representations of a recurrent network to conventional representations of the grammar in terms of production rules or finite automata. Giles and Omlin (1992) and Das *et al.* (1993) have shown how to insert rules into the network before learning begins, while Castaño *et al.* (1995) describe several methods for converting the network's learned internal representation system into a finite automaton.

Some neural networks learn to parse sentences, without attempting to learn the grammar (Ho and Chan 1997, 1999). The parse tree may be encoded using a RAAM, or it may be converted into a sequential form by pre-order traversal and encoded using a sequential RAAM or simple recurrent network. The sentence to be parsed is encoded using a sequential RAAM or simple recurrent network. The parsing problem is then a matter of transforming one connectionist encoding to another; alternatively, the learning of the two encodings may be coupled to make them identical, thus dispensing with the transformation stage. Ho and Chan (1999) test the error-resilience of these methods using a regular string grammar; the sentences used appear to be quite short and have little or no recursion. Ho and Chan are pessimistic about the chances of scaling the method up to larger sentences and grammars.

Other connectionist parsing algorithms use specially-structured networks with the grammar pre-programmed into them (Fanty 1985, Waltz and Pollack 1985, Selman 1985, Cottrell 1989, Charniak and Santos 1991). For example, Charniak and Santos' network is a rectangular grid with the input sentence passing from right to left across the bottom row and the parse tree being built up above it; each column of the grid holds a branch of the parse tree. A few other networks are able to learn the grammar by adapting their structure (Lucas and Damper 1990, Fletcher 1991, 1992, Roques 1994), but this is only possible so far for very simple classes of grammar without recursion.

The objective of this paper is to extend my previous work to recursive graph grammars. The class of grammars used will be the regular stochastic graph grammars, defined in section 3; they will be represented, parsed and learned in a wholly connectionist way.

3. Networks, homomorphisms and languages

This section sets up the theoretical framework for representing regular graph grammars as networks and stating the parsing and learning problems; subsequent sections will solve these problems.

3.1 Parsing without traversal

Let's start with a regular *string* grammar, as represented by a transition diagram (figure 1). To parse a given sentence, *abbbcd*, one traverses the sentence from left to right, simultaneously traversing the transition diagram and matching the symbols encountered in the sentence, $a, b, b, \ldots$, against the symbols encountered in the transition diagram. The dashed lines in figure 1 show which symbol in the sentence is matched against which symbol in the transition diagram; the states of the sentence (the circles) are also matched against the states of the transition diagram. The sentence is accepted as grammatical if and only if the traversals of the sentence and the transition diagram finish simultaneously.

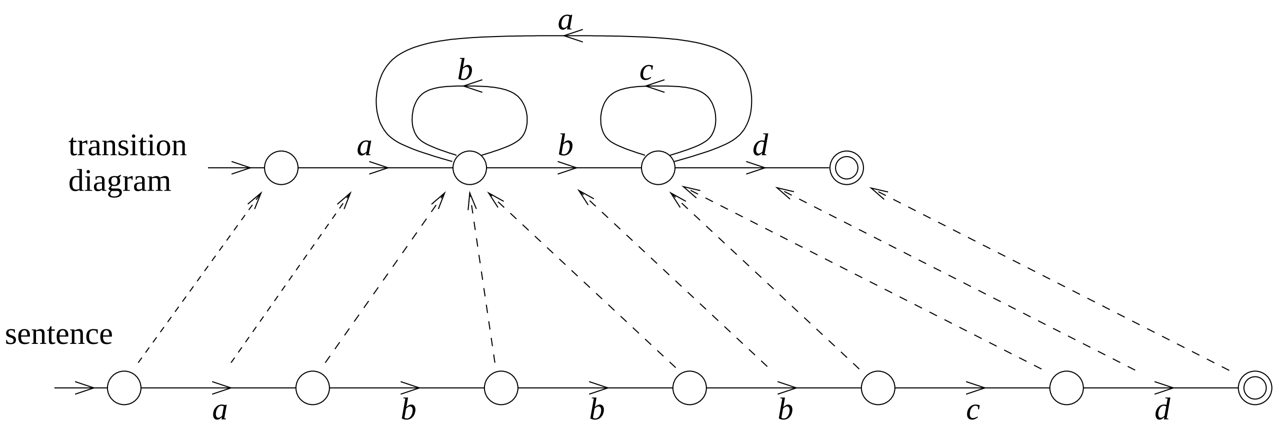


Figure 1. Parsing a sentence *abbbcd* using a transition diagram. The dashed arrows show the correspondence between the sentence's nodes and edges and the transition diagram's nodes and edges (not all arrows are shown).

Now, both the sentence and the transition diagram may be regarded as *networks*, since they both consist of nodes connected by directed edges. However, from a connectionist point of view, what is incongruous about the parsing process just described is the notion of *traversal*, which imposes an unnecessary sequential ordering on the process. Parsing can be rephrased without reference to traversal: the task is simply to find a mapping from the nodes and edges of the sentence to the nodes and edges of the transition diagram such that:

- nodes map to nodes and edges map to edges;
- the direction and symbol label (a, b, c or d) of the edges are preserved under the mapping;
- the end-points (initial and final nodes) of the sentence map to the end-points of the transition diagram;
- the mapping preserves incidence: if an edge is incident to a node in the sentence then they remain incident when the mapping is applied to both.

Let us call such a mapping a *homomorphism* (this is defined formally below). Then the parsing problem is simply to find a homomorphism from the sentence to the transition diagram.

This reformulation of the problem has several advantages. First, by removing the sequential notion of traversal we have made it possible to parse the whole sentence in parallel; for very long sentences this may be quicker than sequential parsing. Secondly, such parallel parsing may be more error-tolerant than sequential parsing. If a sequential parser encounters a grammatical error (think of a compiler scanning a source program, for example) it is liable to misunderstand the rest of the sentence and generate spurious error messages; whereas a parallel parser would parse on both sides of the error and thereby, perhaps, would be better able to locate and correct it. (However, the parsing algorithm in this paper does not attempt any error correction.)

The third, and most important, advantage of parallel parsing is that it is easily extended from string grammars to graph grammars. Let us redraw figure 1 in a slightly different way. Suppose each node (except for the end-points) has two *hooks*, labelled 'in' and 'out', and that all incoming edges are connected to the 'in' hook and all outgoing edges to the 'out' hook. We may now rub out the arrows on the edges, as they are redundant. (See figure 2.)

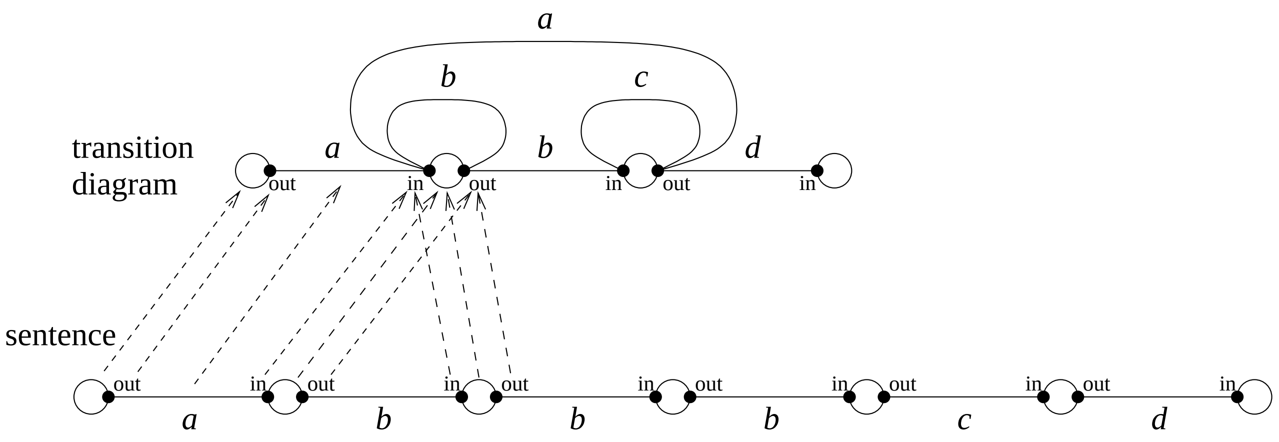


Figure 2. A re-drawing of figure 1, without using traversal. The dashed arrows indicate (part of) the homomorphism. The hooks are indicated by small filled circles.

Note that now a homomorphism maps nodes to nodes, hooks to hooks, and edges to edges. Now, to generalise this to graph grammars we simply rub out the labels 'in' and 'out' (since these are the last remnants of the obsolete notion of traversal) and permit any number of hooks at each node. Figure 3 shows a sentence (which I shall call a *pattern* from now on), the generalised transition diagram (which I shall call a *regular graph grammar*, or just a *grammar*), and the homomorphism between them. The edge labels H and V indicate horizontal and vertical edges. Note that this sort of pattern is very awkward to handle using string grammars (as done in the 'chain code' approach of Brandenburg and Chytel (1991), for example), as it is a non-Eulerian graph and would have to be traversed in several sweeps, which would obscure its geometric structure.

There is a possible problem here. By removing the notion of traversal there is a danger that patterns may be parsed back-to-front: thus, in figure 2, the initial state of the pattern may be mapped to the final state of the grammar, and vice versa; the

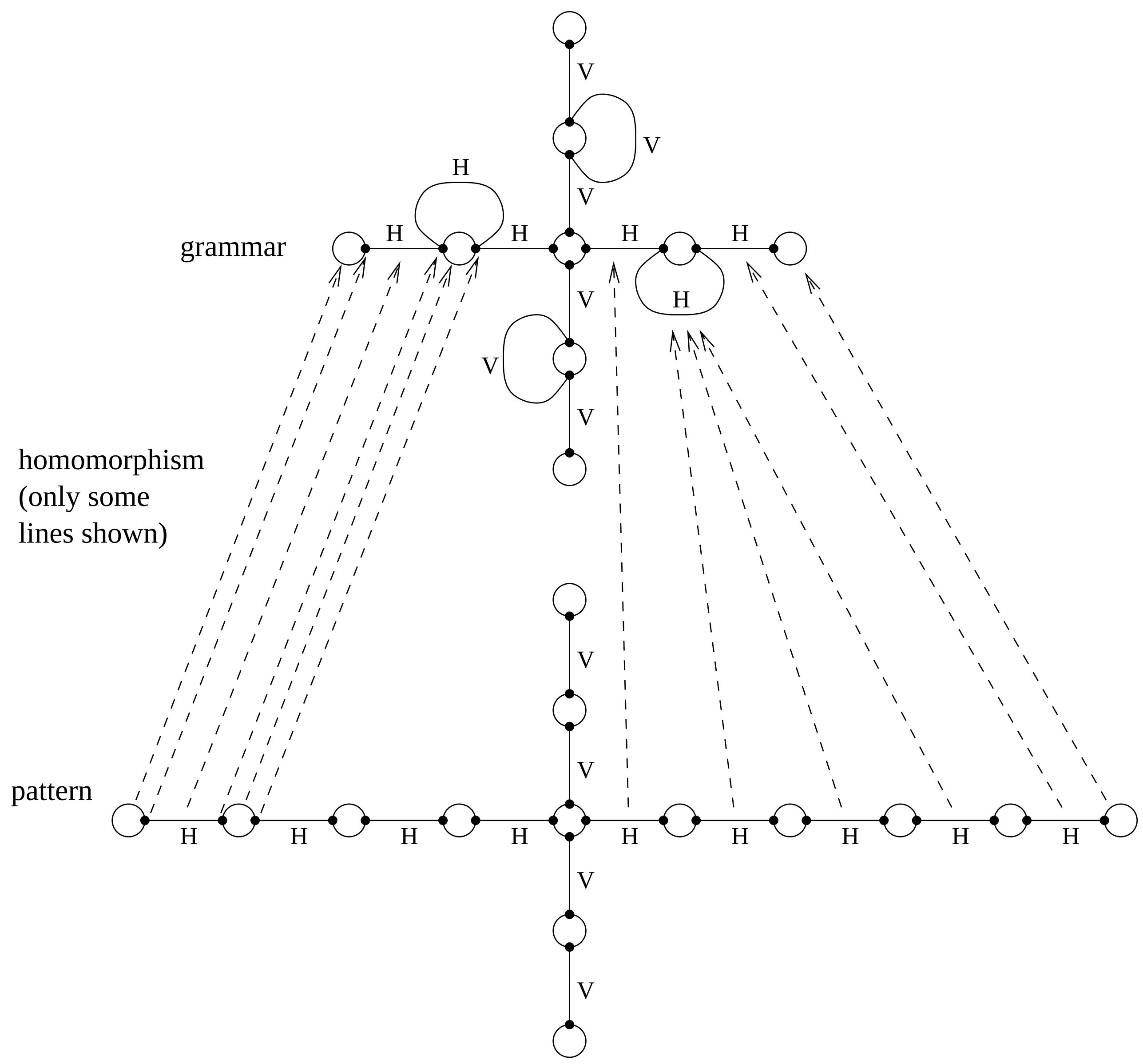


Figure 3. Parsing using a graph grammar.

same problem arises in a more severe form with graph grammars, since a pattern node with k hooks can be mapped to a node in the grammar with the same number of hooks in $k!$ ways. This is called the *direction ambiguity* problem. To help deal with this problem we give each edge in the pattern and the grammar an arbitrary orientation; that is, we designate one of the edge's ends as 'first' and the other as 'second'. In diagrams it is convenient to represent this by drawing an arrow from the 'first' end to the 'second' end. Note that this is *not* a re-introduction of the concept of traversal; it is just a diagrammatic convention for distinguishing one end of an edge from the other. The direction ambiguity problem will be solved by the parsing algorithm in section 4.

This convention is illustrated in figure 4, which shows the *image grid* on which the pattern resides. A pattern, such as the one in figure 3, is drawn by activating a subset of the nodes, hooks and edges of the image grid. The horizontal and vertical connections in the image grid represent the geometric structure of the image space; as mentioned in section 1, these should really be learned from example patterns, but in this paper I am assuming this learning has already been done. The image grid should be thought of as unbounded, though in simulations a grid of finite width and height is used. See figures 11–15 in section 9 for some further examples of grammars, patterns and parses.

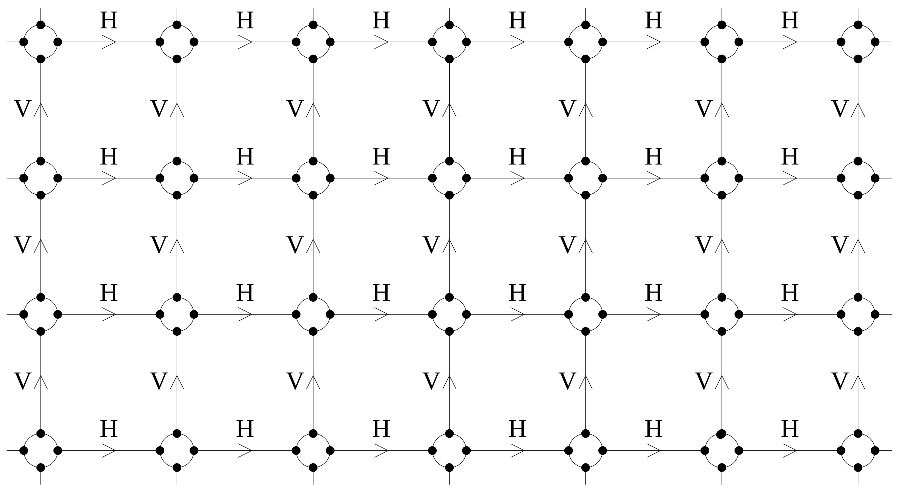


Figure 4. The image grid, showing nodes, each with four hooks, connected by horizontal and vertical edges. The edges are oriented (arbitrarily) rightwards and upwards.

3.2 Formal definition of networks and homomorphisms

In order to define the foregoing notions formally we shall need the following standard notation for logic, sets and functions.

NOTATION.

$\wedge$, $\vee$, $\Rightarrow$, $\forall$ and $\exists$ mean 'and', 'or', 'implies', 'for all' and 'there exists'.
$A \times B$ is the Cartesian product of the sets A and B: $A \times B = \{ (x,y) \mid x \in A \wedge y \in B \}$.
$\biguplus_{i \in I} S_i$ is the disjoint union of the sets S_i: $\biguplus_{i \in I} S_i = \{ (i,x) \mid i \in I \wedge x \in S_i \}$.
$|A|$ is the number of elements in the set A.
$f{:}A \to B$ means that f is a function mapping the set A into the set B.
$f(S)$ is the image of the set S under f: $f(S) = \{ f(x) \mid x \in S \}$.
$f^{-1}(S)$ is the pre-image of the set S under f: $f^{-1}(S) = \{ x \mid f(x) \in S \}$.
$f|_S$ is the function obtained by restricting the domain of f to S.
$f \circ g$ is the composition of the functions f and g: $(f \circ g)(x) = f(g(x))$.

DEFINITION. A *network* is an octuple (N,H,E,L,A,F,S,M), where N, H, E and L are disjoint sets, $A{:}H \to N$, $F{:}E \to H$, $S{:}E \to H$ and $M{:}E \to L$, such that $F(E) \cup S(E) = H$.

The elements of N, H, E and L are called *nodes*, *hooks*, *edges* and *labels*, respectively. If $A(h) = n$ we say that the hook h is *attached to*, or *incident to*, the node n (for example, in figure 3 all the nodes have one, two or four hooks attached to them). If $F(e) = h_1$ and

$S(e) = h_2$ we say that the edge e is *connected to*, or *incident to*, h_1 and h_2; we also say e is *incident to* the nodes $A(h_1)$ and $A(h_2)$; h_1 is called the *first hook* and h_2 the *second hook* of e. We call $M(e)$ the *label of* the edge e. The image grid, the patterns and the grammar are all networks. Formally, a *pattern* is a network (N,H,E,L,A,F,S,M) such that for each $h \in H$ there is a unique $e \in F^{-1}(\{h\}) \cup S^{-1}(\{h\})$. The pattern is said to be *drawn in the image grid*, $(N_G,H_G,E_G,L_G,A_G,F_G,S_G,M_G)$, iff $N \subseteq N_G$, $H \subseteq H_G$, $E \subseteq E_G$, $L \subseteq L_G$, $A = A_G|_H$, $F = F_G|_E$, $S = S_G|_E$ and $M = M_G|_E$. The *pattern population* is the set of patterns drawn in the grid that are presented by the environment.

In what follows, $\mathcal{N}_1 = (N_1,H_1,E_1,L_1,A_1,F_1,S_1,M_1)$ and $\mathcal{N}_2 = (N_2,H_2,E_2,L_2,A_2,F_2,S_2,M_2)$ are networks.

DEFINITION. A *homomorphism* $f{:}\mathcal{N}_1 \to \mathcal{N}_2$ is a function from $N_1 \cup H_1 \cup E_1 \cup L_1$ into $N_2 \cup H_2 \cup E_2 \cup L_2$ such that

$$\begin{aligned}
&f|_{N_1}{:}N_1 \to N_2,\\
&\forall n \in N_1 \quad f|_{A_1{}^{-1}(\{n\})}{:}A_1{}^{-1}(\{n\}) \to A_2{}^{-1}(\{f(n)\}) \text{ is a bijection,}\\
&f|_{E_1}{:}E_1 \to E_2, \qquad F_2 \circ f = f \circ F_1, \qquad S_2 \circ f = f \circ S_1,\\
&f|_{L_1}{:}L_1 \to L_2, \qquad M_2 \circ f = f \circ M_1.
\end{aligned}$$

PROPOSITION 1. If $f{:}\mathcal{N}_1 \to \mathcal{N}_2$ is a homomorphism then

$$f|_{H_1}{:}H_1 \to H_2, \qquad A_2 \circ f = f \circ A_1, \qquad f(H_1) = A_2{}^{-1}(f(N_1)).$$

DEFINITION. An *isomorphism* is a homomorphism $f{:}\mathcal{N}_1 \to \mathcal{N}_2$ that is a bijection from $N_1 \cup H_1 \cup E_1 \cup L_1$ to $N_2 \cup H_2 \cup E_2 \cup L_2$.

PROPOSITION 2. If $f{:}\mathcal{N}_1 \to \mathcal{N}_2$ is an isomorphism then so is the inverse function $f^{-1}{:}\mathcal{N}_2 \to \mathcal{N}_1$.

DEFINITION. A network $\mathcal{N}_1$ is a *subnetwork* of a network $\mathcal{N}_2$ iff

$$\begin{aligned}
&N_1 \subseteq N_2, \qquad H_1 = A_2{}^{-1}(N_1), \qquad E_1 \subseteq E_2, \qquad L_1 \subseteq L_2,\\
&A_1 = A_2|_{H_1}, \qquad F_1 = F_2|_{E_1}, \qquad S_1 = S_2|_{E_1}, \qquad M_1 = M_2|_{E_1}.
\end{aligned}$$

PROPOSITION 3.

(a) If $\mathcal{N}_1$ is a subnetwork of $\mathcal{N}_2$ then the inclusion function $i{:}N_1 \cup H_1 \cup E_1 \cup L_1 \to N_2 \cup H_2 \cup E_2 \cup L_2$, defined by $\forall x\ i(x) = x$, is a homomorphism from $\mathcal{N}_1$ to $\mathcal{N}_2$.

(b) If $f{:}\mathcal{N}_1 \to \mathcal{N}_2$ is a homomorphism then the image network $f(\mathcal{N}_1) = (f(N_1), f(H_1), f(E_1), f(L_1), A_2|_{f(H_1)}, F_2|_{f(E_1)}, S_2|_{f(E_1)}, M_2|_{f(E_1)})$ is a subnetwork of $\mathcal{N}_2$, and there is a unique homomorphism $f'{:}\mathcal{N}_1 \to f(\mathcal{N}_1)$ such that $f = i \circ f'$, where $i{:}f(\mathcal{N}_1) \to \mathcal{N}_2$ is the inclusion homomorphism.

The formal definition of homomorphism is consistent with the informal description given earlier, but is slightly more general in that it allows $\mathcal{N}_1$ (representing the pattern) to have a different set of labels from $\mathcal{N}_2$ (representing the grammar). This is useful in a wider context of geometric concept learning (see Fletcher 1993, especially figure 4), but for the purposes of this paper we shall only deal with homomorphisms for which $L_1 = L_2$ and the label mapping $f|_{L_1}$ is the identity.

3.3 Languages generated by networks

The *language $L(\mathcal{N})$ generated by a network $\mathcal{N}$* is defined as the set of finite connected patterns $\mathcal{P}$ for which there exists a homomorphism $p: \mathcal{P} \to \mathcal{N}$; p is called a *parse* of $\mathcal{P}$. The *grid language $GL(\mathcal{N})$ generated by $\mathcal{N}$* is the set of patterns drawn in the image grid that are also in $L(\mathcal{N})$.

The grammar network $\mathcal{N}$ will also contain stochastic information about the probability distribution of patterns. The network contains a real number n_e for each edge e and a real number n_v for each node v. A probability distribution *prob* over $L(\mathcal{N})$ is said to be *consistent* with these numbers iff

$$\begin{aligned} \forall e \in E \quad n_e &= \mathrm{Exp}_{\mathcal{P}}\ |p^{-1}(\{e\})| \\ \forall v \in N \quad n_v &= \mathrm{Exp}_{\mathcal{P}}\ |p^{-1}(\{v\})| \end{aligned} \tag{1}$$

where $\mathcal{N} = (N, H, E, L, A, F, S, M)$, $\mathrm{Exp}_{\mathcal{P}}$ is the expectation operator over all patterns, defined by

$$\mathrm{Exp}_{\mathcal{P}}\ X = \sum_{\mathcal{P}} prob(\mathcal{P}) X,$$

and $p: \mathcal{P} \to \mathcal{N}$ is the parse of $\mathcal{P}$ (assumed unique). The numbers $|p^{-1}(\{e\})|$ and $|p^{-1}(\{v\})|$ are called the *multiplicities* of e and v in the pattern $\mathcal{P}$, and the numbers n_e and n_v are called *mean multiplicities* of e and v. The *stochastic language generated by $\mathcal{N}$* is $L(\mathcal{N})$ with the maximum-entropy probability distribution over $L(\mathcal{N})$ consistent with the mean multiplicities; the *stochastic grid language generated by $\mathcal{N}$* is defined analogously. The stochastic aspect of the grammar will be important in sections 5–6, but I shall disregard it for the rest of this section.

A network $\mathcal{N}$ is said to be *unambiguous* iff for every pattern $\mathcal{P}$ there is at most one homomorphism from $\mathcal{P}$ to $\mathcal{N}$. It is said to be *simple* iff $\forall e, e' \in E\ (F(e) = F(e') \wedge S(e) = S(e') \wedge M(e) = M(e') \Rightarrow e = e')$ and $\forall e \in E\ F(e) \neq S(e)$. We shall largely be concerned with grammars that are both simple and unambiguous.

The approach taken here, in which grammars are represented as networks, is different from the conventional notion of a graph grammar (Rozenberg 1997), in which patterns are derived from a start symbol by a sequential application of production rules. The relation between the two approaches can be seen by analysing the parsing homomorphism p into simpler homomorphisms, as follows. A simple type of homomorphism is illustrated in figure 5. In this construction we start with an arbitrary network $\mathcal{N}$ and split one node n into two nodes, n_+ and n_-, to give a network $\mathcal{N}'$. $\mathcal{N}'$ has one hook for every hook of $\mathcal{N}$, except in the case of the hooks attached to n, for each of which there are two hooks in $\mathcal{N}'$; for example, corresponding to hook h in $\mathcal{N}$ are two hooks, h_+ and h_-, in $\mathcal{N}'$. Edges are a little more complicated. For each of the edges u, v, x, y, z in $\mathcal{N}$ there are two edges $u_+, u_-, v_+, v_-, x_+, x_-, y_+, y_-, z_+, z_-$ in $\mathcal{N}'$; this is because these edges are connected at one end to a hook of n, which has split into two. If an edge were connected to hooks of n at *both* ends then it would

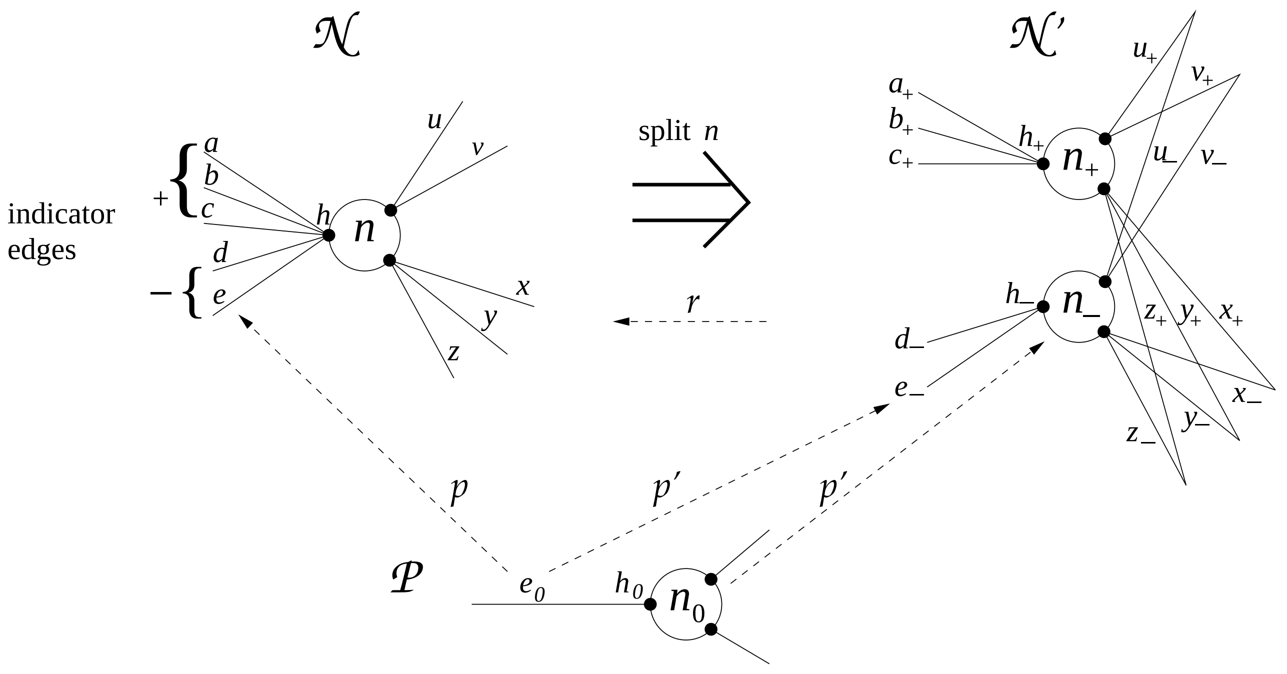


Figure 5. A node split and the associated homomorphism, r. (The letters $a, b, c, d, \ldots$ are used to identify the edges uniquely: they are not edge labels.)

need to split into four in $\mathcal{N}'$, since the hooks at each end would have split into two; such an edge is called a *re-entrant* edge.

However, notice from figure 5 that the edges a, b, c, d, e connected to h have been treated differently: for each of these there is only one corresponding edge in $\mathcal{N}'$, connected to either h_+ or h_-. Hook h is called the *governing* hook, and its edges a, b, c, d, e are called *indicator* edges, with a, b, c being *positive* and d, e *negative*. The operation that takes us from $\mathcal{N}$ to $\mathcal{N}'$ is called a *node split*, and can be specified formally by the homomorphism $r: \mathcal{N}' \to \mathcal{N}$ that maps each node, hook, edge and label of $\mathcal{N}'$ back to the one it came from in $\mathcal{N}$ (for example, $r(n_\pm) = n$, $r(u_\pm) = u$). (I shall not give the formal construction of node splitting in general here, as it is a special case of the splitting operation defined in §6.2 below.)

The reason for the special treatment of the governing hook h and the indicator edges is revealed by the following proposition.

PROPOSITION 4. Let $r: \mathcal{N}' \to \mathcal{N}$ be a homomorphism representing a node split. For any pattern $\mathcal{P}$ and parse $p: \mathcal{P} \to \mathcal{N}$ there is a unique parse $p': \mathcal{P} \to \mathcal{N}'$ such that $p = r \circ p'$.

Proof. Given $\mathcal{P}$ and p, we can construct p' in only one way, which I shall illustrate with the example in figure 5. Each node n_0 in $\mathcal{P}$ such that $p(n_0) = n$ must have a hook h_0 such that $p(h_0) = h$, and h_0 must have a unique incident edge e_0, which must map under p to one of the edges a, b, c, d, e. If $p(e_0)$ is a, b or c then we define $p'(e_0)$ as a_+, b_+ or c_+, respectively, and hence $p'(h_0) = h_+$, $p'(n_0) = n_+$ and so on for all other hooks and edges incident to n_0. Whereas if $p(e_0)$ is d or e instead then we define $p'(e_0)$ as d_- or e_-, and hence $p'(h_0) = h_-$, $p'(n_0) = n_-$ and so on for all other hooks and edges incident to n_0. The key point is that it is the edge e_0 incident to hook h_0 that determines whether n_0 maps to n_+ or n_- under p'. We then extend p' to the whole of $\mathcal{P}$ by defining $p'(x) = p(x)$ for all other nodes, hooks, edges and labels, x. ▮

PROPOSITION 5. Let $r: \mathcal{N}' \to \mathcal{N}$ be a homomorphism representing a node split.
(a) $L(\mathcal{N}') = L(\mathcal{N})$ and $GL(\mathcal{N}') = GL(\mathcal{N})$.
(b) $\mathcal{N}'$ is unambiguous if $\mathcal{N}$ is.
(c) $\mathcal{N}'$ is simple if $\mathcal{N}$ is.

Proof. (a) If a pattern $\mathcal{P}$ has a parse $p: \mathcal{P} \to \mathcal{N}$ then, by the previous proposition, it has a parse $p': \mathcal{P} \to \mathcal{N}'$; conversely, if it has a parse $p': \mathcal{P} \to \mathcal{N}'$ then it has a parse $r \circ p': \mathcal{P} \to \mathcal{N}$. This establishes that $L(\mathcal{N}') = L(\mathcal{N})$ and hence $GL(\mathcal{N}') = GL(\mathcal{N})$.

(b) For any parse of a pattern, $p': \mathcal{P} \to \mathcal{N}'$, we can produce a parse $p = r \circ p': \mathcal{P} \to \mathcal{N}$. But p is unique if $\mathcal{N}$ is unambiguous, so by proposition 4 p' is unique.

(c) is straightforward. ▮

A second simple type of homomorphism corresponds to pruning unwanted parts of the network. Given any network $\mathcal{N}$ we can remove nodes, hooks, edges (and possibly labels) in such a way as to give a subnetwork $\mathcal{N}'$. The pruning operation may be represented formally by the inclusion homomorphism $i: \mathcal{N}' \to \mathcal{N}$. These two simple types of homomorphism are sufficient to generate all parses.

PROPOSITION 6. Any parse $p: \mathcal{P} \to \mathcal{N}$ can be expressed (up to isomorphism) as a composition of homomorphisms representing node splits and prunings.

Proof. Start with the network $\mathcal{N}$, prune any edges and nodes (and their hooks) that have multiplicity 0 under the parse p, giving a subnetwork $p(\mathcal{P})$, an inclusion homomorphism $i: p(\mathcal{P}) \to \mathcal{N}$, and a parse $p': \mathcal{P} \to p(\mathcal{P})$ (by proposition 3). Next, choose a node n in $p(\mathcal{P})$ of multiplicity greater than 1 (if there is one) and split n into two nodes, n_+ and n_-, giving a new network $\mathcal{N}'$, a homomorphism $r: \mathcal{N}' \to p(\mathcal{P})$, and a parse $p'': \mathcal{P} \to \mathcal{N}'$ (by proposition 4). Then we have

$$r \circ p'' = p', \qquad i \circ p' = p, \qquad \text{and hence } i \circ r \circ p'' = p.$$

Some of the newly created edges in $\mathcal{N}'$ may have multiplicity 0, so continue the sequence of alternate prunings and node splits until one obtains a network $\mathcal{N}^*$ in which all nodes and edges have multiplicity 1 (this must happen eventually since the multiplicities are reducing at each step). Hence $\mathcal{N}^*$ is isomorphic to $\mathcal{P}$. Thus p has been analysed as a composition of node splitting and pruning homomorphisms. ▮

The process described in proposition 6 may be regarded as a grammatical derivation of the pattern $\mathcal{P}$. The start symbol is $\mathcal{N}$; figure 5 depicts a production rule, with n and its incident hooks and edges as left-hand side, and n_+, n_- and their incident hooks and edges as right-hand side; application of a production rule consists of carrying out a node split or a pruning. This gives us a kind of node-replacement graph grammar.

In general, networks and homomorphisms form a mathematical system called a *category*, since every network has an identity homomorphism and composition of homomorphisms is associative (see Goldblatt (1984) for an introduction to category theory). This category is the search space for the learning problem; it is essentially a generalisation of Dupont *et al.*'s (1994) representation of the search space for string grammatical inference as a lattice of automata. The category has products and pullbacks but not equalisers or a terminal object. The pullback construction is particularly useful as it provides a Church-Rosser property for these grammars (cf Rozenberg 1997: 173).

PROPOSITION 7. Given networks $\mathcal{N}_0, \mathcal{N}_1, \mathcal{N}_2$ and homomorphisms $r_1 : \mathcal{N}_1 \to \mathcal{N}_0$ and $r_2 : \mathcal{N}_2 \to \mathcal{N}_0$, there exist a network $\mathcal{N}$ and homomorphisms $\pi_1 : \mathcal{N} \to \mathcal{N}_1$ and $\pi_2 : \mathcal{N} \to \mathcal{N}_2$ such that the following hold.

(a) If $\mathcal{N}_1$ and $\mathcal{N}_2$ are simple then so is $\mathcal{N}$.
(b) $r_1 \circ \pi_1 = r_2 \circ \pi_2$.
(c) For any network $\mathcal{N}^*$ and homomorphisms $\pi_1^* : \mathcal{N}^* \to \mathcal{N}_1$ and $\pi_2^* : \mathcal{N}^* \to \mathcal{N}_2$ such that $r_1 \circ \pi_1^* = r_2 \circ \pi_2^*$, there is a unique homomorphism $k : \mathcal{N}^* \to \mathcal{N}$ such that $\pi_1 \circ k = \pi_1^*$ and $\pi_2 \circ k = \pi_2^*$.
(d) If $\mathcal{N}_0$ is unambiguous then $L(\mathcal{N}) = L(\mathcal{N}_1) \cap L(\mathcal{N}_2)$ and hence $GL(\mathcal{N}) = GL(\mathcal{N}_1) \cap GL(\mathcal{N}_2)$.
(e) If $\mathcal{N}_1$ and $\mathcal{N}_2$ are unambiguous then so is $\mathcal{N}$.
(f) If r_1 and r_2 represent node splits or prunings then π_1 and π_2 represent node splits or prunings, or the compositions of two node splits or prunings.

Proof. Let $\mathcal{N}_0 = (N_0, H_0, E_0, L_0, A_0, F_0, S_0, M_0)$, let $\mathcal{N}_1 = (N_1, H_1, E_1, L_1, A_1, F_1, S_1, M_1)$, and let $\mathcal{N}_2 = (N_2, H_2, E_2, L_2, A_2, F_2, S_2, M_2)$.

Define $N = \{\, (n_1, n_2) \in N_1 \times N_2 \mid r_1(n_1) = r_2(n_2) \,\}$.
Define $H = \{\, (h_1, h_2) \in H_1 \times H_2 \mid r_1(h_1) = r_2(h_2) \,\}$.
Define $E = \{\, (e_1, e_2) \in E_1 \times E_2 \mid r_1(e_1) = r_2(e_2) \,\}$.
Define $L = \{\, (l_1, l_2) \in L_1 \times L_2 \mid r_1(l_1) = r_2(l_2) \,\}$.
Define $A : H \to N$ by $\forall (h_1, h_2) \in H \; A(h_1, h_2) = (A_1(h_1), A_2(h_2))$.
Define $F : E \to H$ by $\forall (e_1, e_2) \in E \; F(e_1, e_2) = (F_1(e_1), F_2(e_2))$.
Define $S : E \to H$ by $\forall (e_1, e_2) \in E \; S(e_1, e_2) = (S_1(e_1), S_2(e_2))$.
Define $M : E \to L$ by $\forall (e_1, e_2) \in E \; M(e_1, e_2) = (M_1(e_1), M_2(e_2))$.

The obvious way to define the network $\mathcal{N}$ would be as (N, H, E, L, A, F, S, M). However, this would be unsatisfactory, as H may contain hooks that have no incident edges. Such hooks need to be removed from H; moreover, when a hook is removed its node must also be removed, as must all the node's hooks and edges; this may deprive some other hooks of all their edges, so that they need to be removed, and so on. This removal procedure is formally specified as follows.

Let N', H' and E' be the maximal subsets of N, H and E (respectively) such that

$$F(E') \cup S(E') = H' = A^{-1}(N').$$

(These sets can be constructed by considering all triples (N',H',E') satisfying the above equations and then taking the union of all the N' sets, the union of all the H' sets, and the union of all the E' sets.) Then the pullback network $\mathcal{N}$ is defined as $(N',H',E',L,A|_{H'},F|_{E'},S|_{E'},M|_{E'})$. The homomorphisms $\pi_1{:}\mathcal{N} \to \mathcal{N}_1$ and $\pi_2{:}\mathcal{N} \to \mathcal{N}_2$ are defined by

$$\begin{aligned}
&\forall (n_1,n_2) \in N' \quad \pi_1(n_1,n_2) = n_1 \quad \pi_2(n_1,n_2) = n_2\\
&\forall (h_1,h_2) \in H' \quad \pi_1(h_1,h_2) = h_1 \quad \pi_2(h_1,h_2) = h_2\\
&\forall (e_1,e_2) \in E' \quad \pi_1(e_1,e_2) = e_1 \quad \pi_2(e_1,e_2) = e_2\\
&\forall (l_1,l_2) \in L \quad \pi_1(l_1,l_2) = l_1 \quad \pi_2(l_1,l_2) = l_2.
\end{aligned}$$

It is routine to verify that this construction satisfies the proposition. ▮

Hence, by part (f), if the homomorphisms r_1 and r_2 each represent application of a production rule then π_1 and π_2 each represent the application of one or two production rules. In fact, π_1 simply represents application of the analogous production rule(s) on $\mathcal{N}_1$ to that of r_2 on $\mathcal{N}_0$; and similarly for π_2. The composite homomorphism $r_1 \circ \pi_1$ may be regarded as representing a parallel application of the two production rules represented by r_1 and r_2 (cf Rozenberg 1997: 175). Proposition 7 shows the context-free nature of this class of graph grammar.

The operations of node splitting and pruning, together with propositions 3–7, which I have introduced in the context of interpreting networks as graph grammars, will be useful in a quite different way in section 5 in the theory of learning.

The outcome of this section is a statement of the parsing and learning problems: the parsing problem is to find a homomorphism from a given pattern to a given grammar; the learning problem is to find a grammar that generates a grid language (or stochastic grid language) equal to a given pattern population.

4. Parsing

This section describes how patterns are parsed using a given graph grammar. Both the pattern and the grammar are networks, as defined in section 3, and the task of parsing is to find a homomorphism from the pattern to the grammar. I shall introduce the algorithm informally with an example before stating it formally.

4.1 Informal account of parsing

The idea is to build up a parse of the pattern by piecing together parse-fragments. A *parse-fragment* is a copy of a part of the pattern together with a homomorphic mapping to the grammar. We begin with parse-fragments consisting of single edges (plus the hooks, and possibly the node, at either end). Then consecutive parse-fragments are combined in all consistent ways, until eventually we have a parse of the whole pattern (or all possible parses, if the grammar is ambiguous).

Figure 6 shows the grammar and pattern to be used in our example. The edge labels are a, b, c, d, but for ease of reference the edges in the grammar have also been given numeric subscripts and the nodes have been indexed with Greek or capital Roman letters.

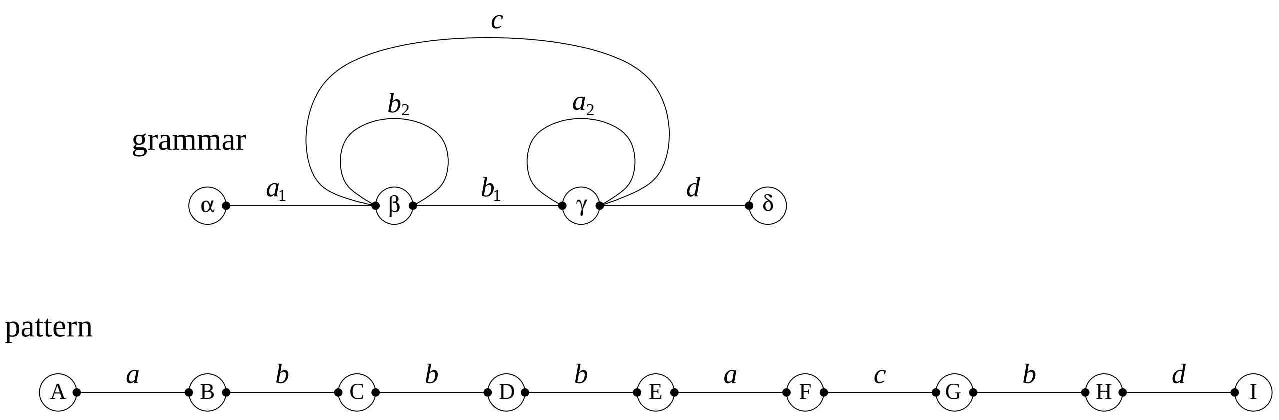


Figure 6. Parsing example: the grammar and pattern.

Step 1: copy edges.

We begin by constructing all possible parse-fragments involving one edge. There will be one such parse-fragment for every pair consisting of an edge in the pattern and an edge with the same label in the grammar. There is a function f mapping each of these parse-fragments to the corresponding edge in the grammar, and a function g mapping each parse-fragment to the corresponding edge in the pattern. In figure 7 each parse-fragment is marked with the same label and subscript as the corresponding edge in the grammar, while its horizontal position indicates which edge in the pattern it corresponds to.

Step 2: copy hooks.

Next we attach hooks to each end of the parse-fragments (see figure 7). The functions f and g are extended to map each of these hooks to the corresponding hooks in the grammar and the pattern.

Step 3: copy nodes.

Next we attach nodes to some of the parse-fragments. For each pattern node we choose one of its hooks at random and attach a node to the corresponding hooks in the parse-fragments. The functions f and g are extended to map these nodes to the corresponding nodes in the grammar and pattern; in figure 7 each node has been marked with the name of the grammar node it maps to under f.

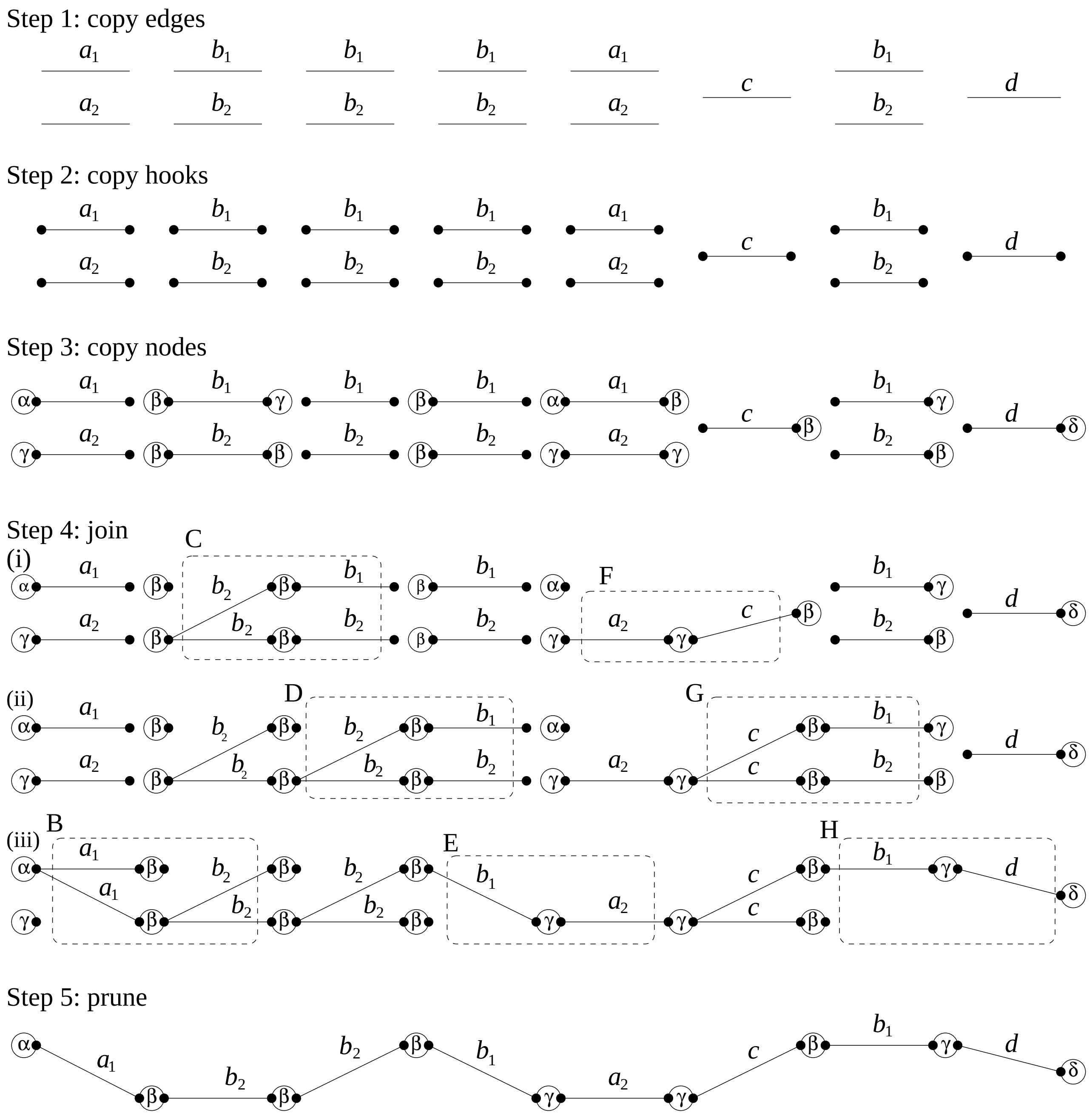


Figure 7. The parse-fragments produced while parsing the example in figure 6. At each stage of step 4, the dashed boxes indicate the nodes, hooks and edges that have changed.

Step 4: join.

Next we combine the parse-fragments into larger fragments. In stage (i) of this step (see the figure) we have chosen (arbitrarily) to begin by joining at the points corresponding to nodes C and F of the pattern (see the dashed boxes marked C and F in the figure). At the position corresponding to node C there are (after step 3) two nodes, marked γ and β, and two dangling hooks to join to them, giving four possible combinations. However, not all of these combinations are consistent with the grammar: the grammar allows β to be followed by either b_1 or b_2, but does not allow γ followed by either b_1 or b_2. So only two of the four combinations are grammatical; we create a new node for each of the two consistent combinations (with hooks and edges attached as appropriate), replacing the two nodes previously at that position and their hooks and edges. This is shown in figure 7, stage (i): the newly created nodes, hooks and edges are shown enclosed in the dashed box marked C. Note that the node marked β has been duplicated, along with its hook and incident edge marked b_2; the node marked γ has been removed, along with its hook and incident edge marked b_1.

Simultaneously, a join occurs at the position corresponding to node F. Here, there is one hook to be joined to two nodes, but the grammar only allows the hook to connect to one of the nodes (since c can follow γ, not α). So there is one new node, replacing the two nodes previously at that position. The new node is given appropriate hooks and edges; the new node, hooks and edges are shown enclosed in the dashed box marked F in the figure.

Next suppose we join at the positions corresponding to nodes D and G in the pattern, producing stage (ii) in the figure in a similar way. Again, the nodes at those positions, together with their hooks and incident edges, are removed and replaced by new ones; the new nodes, hooks and edges are shown enclosed in the dashed boxes marked D and G.

Next we join at the remaining positions, corresponding to nodes B, E and H, giving stage (iii) in the figure. All the parse-fragments have now been combined.

Step 5: prune.

At this stage the parse-fragments may contain dangling edges and nodes, which ought to be removed, as shown in the figure. Also, the parse-fragments may provide more than one complete parse, due to the direction ambiguity problem mentioned in section 3. This is certain to be the case for the very first pattern, since every node in the pattern with k hooks can map to a node with the same number of hooks in the grammar in $k!$ ways; we need to select a particular parse by breaking the symmetry. We can deal with both problems simultaneously as follows. For each pattern edge e_2 we must select one edge in $g^{-1}(\{e_2\})$ and for each pattern node n_2 we must select one node in $g^{-1}(\{n_2\})$. We assign each edge e_1 in the parse-fragments a real number $\rho(e_1)$, constrained to lie between 0 and 1, the interpretation being that $\rho(e_1) = 1$ if e_1 is selected to be part of the parse, $\rho(e_1) = 0$ if it is not selected, and $0 < \rho(e_1) < 1$ if it is not yet decided whether it is selected; the nodes n_1 are also assigned real numbers $\rho(n_1)$ between 0 and 1. Each $\rho(e_1)$ and $\rho(n_1)$ is given an initial value between 0 and 1 (with random perturbations to break the symmetry), and then we perform gradient ascent on the objective function

$$\sum_{e_0} \Big(n_{e_0} + \sum_{e_1 \in f^{-1}(\{e_0\})} \rho(e_1) \Big) \ln \Big(n_{e_0} + \sum_{e_1 \in f^{-1}(\{e_0\})} \rho(e_1) \Big)$$

(summing over all edges e_0 in the grammar), subject to the following constraints:

- for every edge e_2 in the pattern, $\sum_{e_1 \in g^{-1}(\{e_2\})} \rho(e_1) = 1$ (cf step 5.2 below);
- for every hook h_1 in a parse-fragment, $\sum_{e_1} \rho(e_1) = \rho(n_1)$, where e_1 ranges over all incident edges of h_1 and n_1 is the node to which h_1 is attached (cf steps 5.3–5.5 below).

While this is going on, edges e_1 in the parse-fragments with $\rho(e_1) = 0$ are removed, as are nodes that have a hook with no incident edges (see step 5.6 below). Eventually the function g becomes injective and the pruning process is finished. The objective function used above is consistent with that used for splitting and merging (equation (6) in section 6.1 below), taking $n_{e_0} + \sum_{e_1 \in f^{-1}(\{e_0\})} \rho(e_1)$ as an estimate of the mean multiplicity of e_0 based on past and present values of multiplicity.

Step 6: result.

We should now have a single parse-fragment, with g an isomorphism from the parse-fragment to the pattern and f a homomorphism from the parse-fragment to the grammar. Then $f \circ g^{-1}$ is the parse of the pattern. The alternative possibility that all the parse-fragments have been pruned at step 5, in which case there is no parse.

4.2 Formal statement of the parsing algorithm

The algorithm uses the following notation for assignment statements.

NOTATION.

$x := E$ means assign the value of the expression E to the variable x;
$f(X) := Y$ means make the function f map the argument X to the value Y;
$f(X) := \perp$ means make the function f undefined for argument X.

The parsing problem is specified as follows. Given a grammar, represented as a network $\mathcal{N}_0 = (N_0, H_0, E_0, L_0, A_0, F_0, S_0, M_0)$, and a pattern, represented as a network $\mathcal{P} = (N_2, H_2, E_2, L_2, A_2, F_2, S_2, M_2)$, the task is to find a homomorphism $p: \mathcal{P} \to \mathcal{N}_0$.

The procedure is to construct an intermediate network $\mathcal{N}_1 = (N_1, H_1, E_1, L_1, A_1, F_1, S_1, M_1)$, built up out of parse-fragments, and homomorphisms $f: \mathcal{N}_1 \to \mathcal{N}_0$, and $g: \mathcal{N}_1 \to \mathcal{P}$ as in the example above. There is one real parameter, η, used in the gradient ascent in step 5, for which I adopt the value 0.01.

Step 1: copy edges.

We need to create one parse-fragment for each edge $e_0 \in E_0$ and each edge $e_2 \in E_2$ with the same label. Each parse-fragment will consist of a single edge; it is mathematically convenient to represent this edge by the pair (e_0, e_2). (We may restrict attention to pairs (e_0, e_2) where the nodes incident to e_0 have the same number of hooks as the nodes incident to e_2, since only such parse-fragments can be extended to a parse of the whole pattern.) The set of all such edges is E_1. Formally, this is done as follows.

$$\begin{aligned} E_1 := \{\, (e_0, e_2) \in E_0 \times E_2 \mid\ & M_0(e_0) = M_2(e_2) \wedge \\ & |A_0{}^{-1}(\{A_0(F_0(e_0))\})| = |A_2{}^{-1}(\{A_2(F_2(e_2))\})| \wedge \\ & |A_0{}^{-1}(\{A_0(S_0(e_0))\})| = |A_2{}^{-1}(\{A_2(S_2(e_2))\})| \,\} \end{aligned}$$

$L_1 := L_2$
$\forall (e_0, e_2) \in E_1 \quad f(e_0, e_2) := e_0 \quad g(e_0, e_2) := e_2 \quad M_1(e_0, e_2) := M_2(e_2)$
$\forall l_1 \in L_1 \quad g(l_1) := l_1 \quad f(l_1) := l_1$

Step 2: copy hooks.

Next we create hooks for each end of every edge e_1 in E_1. It is mathematically convenient to represent each such hook as a pair (h_0, h_2), where h_0 and h_2 are the corresponding hooks in the grammar and the pattern, respectively; that is, h_0 is $F_0(f(e_1))$ or $S_0(f(e_1))$, and h_2 is $F_2(g(e_1))$ or $S_2(g(e_1))$. The set of all such hooks makes up H_1. Formally,

$H_1 := \{ (F_0(f(e_1)), F_2(g(e_1))) \mid e_1 \in E_1 \} \cup \{ (S_0(f(e_1)), S_2(g(e_1))) \mid e_1 \in E_1 \}$
$\forall e_1 \in E_1 \quad F_1(e_1) := (F_0(f(e_1)), F_2(g(e_1))) \quad S_1(e_1) := (S_0(f(e_1)), S_2(g(e_1)))$
$\forall (h_0, h_2) \in H_1 \quad f(h_0, h_2) := h_0 \quad g(h_0, h_2) := h_2.$

Step 3: copy nodes.

Now we create some nodes to attach to the parse-fragments; these will be represented mathematically as pairs (h_1, n_2), where $h_1 \in H_1$ and $n_2 \in N_2$, and will be collected together in the set N_1. First, set $N_1 := \emptyset$. Then, for each $n_2 \in N_2$, choose one $h_2 \in A_2^{-1}(\{n_2\})$ and carry out the following assignments:

$N_1 := N_1 \cup (g^{-1}(\{h_2\}) \times \{n_2\})$
$\forall h_1 \in g^{-1}(\{h_2\}) \quad A_1(h_1) := (h_1, n_2) \quad f(h_1, n_2) := A_0(f(h_1)) \quad g(h_1, n_2) := n_2.$

Step 4: join. (See figure 8 and accompanying text below.)

Carry out the following sequence of steps repeatedly as many times as possible.

(4.1) Choose $h_2 \in H_2$ such that $g^{-1}(\{h_2\}) \cap dom(A_1) = \emptyset$ (if there is no such h_2 then step 4 is finished).

(4.2) Define

$nodes = g^{-1}(\{A_2(h_2)\}) \qquad hooks = g^{-1}(\{h_2\})$
$newnodes = \{ (n, h) \in nodes \times hooks \mid f(n) = A_0(f(h)) \},$
$\forall (n, h) \in newnodes \; Y_{(n,h)} = {A_1}^{-1}(\{n\}) \cup \{h\}$
$oldhooks = {A_1}^{-1}(nodes) \cup hooks \qquad newhooks = \biguplus_{x \in newnodes} Y_x$
$oldedges = {F_1}^{-1}(oldhooks) \cup {S_1}^{-1}(oldhooks)$
$newedges = \biguplus_{x \in newnodes} {F_1}^{-1}(Y_x) \cup {S_1}^{-1}(Y_x)$

and carry out the following assignments

$N_1 := (N_1 \setminus nodes) \cup newnodes$
$H_1 := (H_1 \setminus oldhooks) \cup newhooks$
$E_1 := (E_1 \setminus oldedges) \cup newedges.$

(4.3) Update the incidence functions and homomorphisms accordingly:

$\forall (n, h) \in newnodes \quad f(n, h) := f(n) \quad g(n, h) := g(n)$
$\forall n \in nodes \quad f(n) := \perp \quad g(n) := \perp$
$\forall (x, h) \in newhooks \quad A_1(x, h) := x \quad f(x, h) := f(h) \quad g(x, h) := g(h)$

$\forall h \in oldhooks \quad A_1(h) := \perp \quad f(h) := \perp \quad g(h) := \perp$
$\forall (x,e) \in newedges \quad F_1(x,e) := (x, F_1(e)) \quad S_1(x,e) := (x, S_1(e))$
$\qquad M_1(x,e) := M_1(e) \quad f(x,e) := f(e) \quad g(x,e) := g(e)$
$\forall e \in oldedges \quad F_1(e) := \perp \quad S_1(e) := \perp \quad M_1(e) := \perp \quad f(e) := \perp \quad g(e) := \perp$.

Step 5: prune.

Assign initial values as follows:

$\forall e_1 \in E_1 \quad \rho(e_1) := random(0.99, 1.01)/|g^{-1}(\{g(e_1)\})|$
$\forall e_0 \in E_0 \quad \rho(e_0) := random(0.7, 1.4) \times (n_{e_0} + \sum_{e_1 \in f^{-1}(\{e_0\})} \rho(e_1))$

where $random(a,b)$ is a random number chosen from a uniform probability distribution on the interval $[a,b]$. Then repeat the following sequence of steps until g is injective.

(5.1) $\forall e_1 \in E_1 \quad \rho(e_1) := \rho(e_1) + \eta(\ln \rho(f(e_1)) + 1)$.

(5.2) $\forall e_2 \in E_2 \ \forall e_1 \in g^{-1}(\{e_2\}) \quad \rho(e_1) := \max(\rho(e_1) - \delta, 0)$

$$\text{where } \delta = \max\left(\frac{\sum_{e'_1 \in g^{-1}(\{e_2\})} \rho(e'_1) - 1}{|\{\, e'_1 \in g^{-1}(\{e_2\}) \mid \rho(e'_1) > 0 \,\}|}, \max_{e'_1 \in g^{-1}(\{e_2\})} \rho(e'_1) - 1 \right).$$

(5.3) $\forall h_1 \in H_1 \quad \rho(h_1) := \sum_{e_1 \in F_1{}^{-1}(\{h_1\}) \cup S_1{}^{-1}(\{h_1\})} \rho(e_1)$

(5.4) $\forall n_1 \in N_1 \quad \rho(n_1) := \left(\sum_{h_1 \in A_1{}^{-1}(\{n_1\})} \rho(h_1)\right) / |A_1{}^{-1}(\{n_1\})|$

(5.5) $\forall n_1 \in N_1 \ \forall h_1 \in A_1{}^{-1}(\{n_1\}) \ \forall e_1 \in F_1{}^{-1}(\{h_1\}) \cup S_1{}^{-1}(\{h_1\})$
$\rho(e_1) := \max(\rho(e_1) - \delta_{h_1}, 0)$
where $\delta_{h_1} =$

$$\begin{cases} (\rho(h_1) - \rho(n_1)) / \left|F_1{}^{-1}(\{h_1\}) \cup S_1{}^{-1}(\{h_1\})\right| & \text{if } \rho(h_1) \le \rho(n_1) \\ (\rho(h_1) - \rho(n_1)) / \left|\{\, e_1 \in F_1{}^{-1}(\{h_1\}) \cup S_1{}^{-1}(\{h_1\}) \mid \rho(e_1) > 0 \,\}\right| & \text{if } \rho(h_1) > \rho(n_1) \end{cases}$$

(5.6) Choose an $n_1 \in N_1$ such that $\exists h_1 \in A_1{}^{-1}(\{n_1\}) \ F_1{}^{-1}(\{h_1\}) \cup S_1{}^{-1}(\{h_1\}) = \emptyset$ and carry out the following assignments:

$N_1 := N_1 \setminus \{n_1\}$
$H_1 := H_1 \setminus A_1^{-1}(\{n_1\})$
$E_1 := E_1 \setminus (F_1^{-1}(A_1^{-1}(\{n_1\})) \cup S_1^{-1}(A_1^{-1}(\{n_1\})))$
$f(n_1) := \perp \qquad g(n_1) := \perp$
$\forall h_1 \in A_1{}^{-1}(\{n_1\}) \quad A_1(h_1) := \perp \quad f(h_1) := \perp \quad g(h_1) := \perp$
$\forall e_1 \in F_1{}^{-1}(A_1{}^{-1}(\{n_1\})) \cup S_1{}^{-1}(A_1{}^{-1}(\{n_1\}))$
$\qquad F_1(e_1) := \perp \quad S_1(e_1) := \perp \quad M_1(e_1) := \perp \quad f(e_1) := \perp \quad g(e_1) := \perp$.

Alternatively, if there is no such n_1, then choose an $e_1 \in E_1$ such that $\rho(e_1) = 0$ (if there is one) and carry out the following assignments:

$E_1 := E_1 \setminus \{e_1\}$
$F_1(e_1) := \perp \quad S_1(e_1) := \perp \quad M_1(e_1) := \perp \quad f(e_1) := \perp \quad g(e_1) := \perp$.

(5.7) $\forall e_0 \in E_0 \quad \rho(e_0) := n_{e_0} + \sum_{e_1 \in f^{-1}(\{e_0\})} \rho(e_1)$.

Step 6: result.

If the homomorphism g is a bijection, let $p = f \circ g^{-1} : \mathcal{P} \to \mathcal{N}_0$: this is the parse of the pattern. If g is not a bijection then the parse has failed.

This completes the algorithm. See figure 8 for assistance in understanding step 4. The aim is to combine the parse-fragments on the left of the diagram with all the parse-fragments on the right; this means attaching all the hooks in the set *hooks* to all the nodes in the set *nodes* in all ways consistent with the grammar, taking copies where necessary. The set of all such consistent combinations is *newnodes*, and this set is inserted in N_1, replacing *nodes*. To each new node $x \in newnodes$ we need to attach copies of the relevant hooks (in Y_x) and all their incident edges.

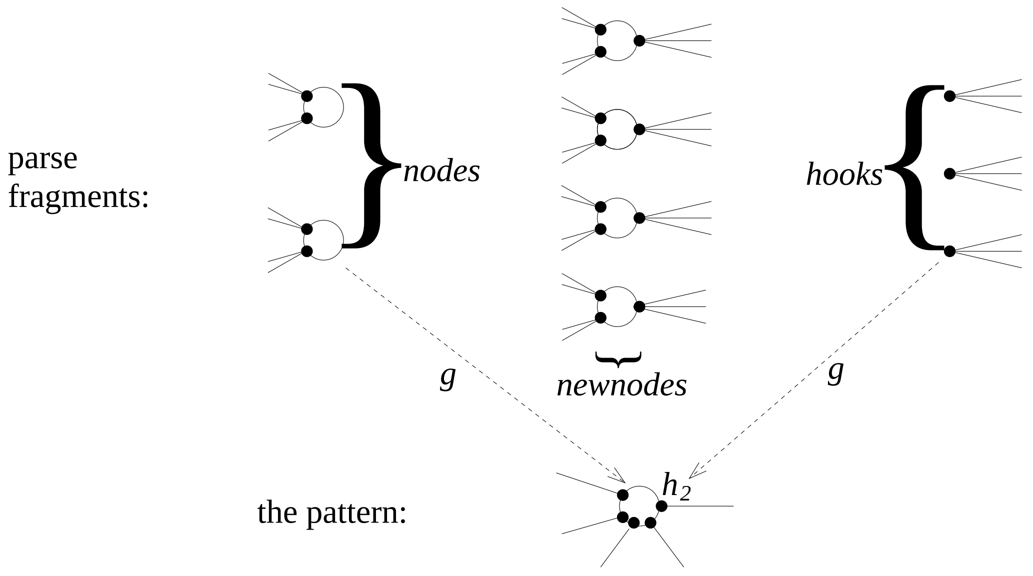


Figure 8. Step 4 of the parsing algorithm: we combine all the parse-fragments terminating in *nodes* with all the parse-fragments terminating in *hooks* in all ways consistent with the grammar; *newnodes* is the resulting set of combinations.

4.3 Comments on the computational complexity of parsing

Most of the work in the parsing algorithm is in step 4; the time complexity of this step can be reduced by carrying out as many as possible of the 'join' operations in parallel. To be precise, if two hooks h_2 and h_2' in the pattern belong to nodes that are not neighbours (i.e. have no common incident edge), then the 'join' operations on h_2 and h_2' can be executed in parallel. We can partition the nodes of the image grid into two classes such that no two nodes in the same class are neighbours; then we can apply 'join' operations simultaneously to one hook from every node in one class. Thus all the 'join' operations can be completed in at most six parallel steps, regardless of the size of the image grid, the pattern, and the grammar.

Since the parsing algorithm works by exhaustively exploring all parse-fragments, it may appear that a very large number of parse-fragments is generated at the 'copy edges' step and the number is multiplied at every 'join' operation. However, in reality the proliferation of parse-fragments is limited tightly by grammatical constraints. Moreover, there is scope for improving the efficiency of the algorithm by reducing the number of edges generated at the 'copy edges' step. Before generating an edge (e_0, e_2), we can check whether, for every edge e_2' incident to the same node as e_2, there is an edge e_0' incident to the same node as e_0 (on the same side) with the same label and direction as e_2': if the answer is no then there is no need to generate the edge (e_0, e_2), since it cannot be extended to a larger parse-fragment. I shall refer to this as the *parsimonious* version of the algorithm.

The total size of the parse-fragments at any stage of parsing can be measured by the number of edges in E_1. In practice this number declines monotonically during step 4, for unambiguous grammars. Table 1 shows the number of E_1 edges after step 1 and after step 4, for both the original algorithm and the parsimonious version, using the 'carpet' patterns in section 9.5; m and n are the numbers of zigzags horizontally and vertically. For purposes of comparison, the table also shows the number of edges in the pattern of each size. Observe that the numbers of E_1 edges increase in proportion to the number of pattern edges.

		pattern	original version		parsimonious version	
m	n	edges	after step 1	after step 4	after step 1	after step 4
5	5	200	2402	304	519	242
20	20	740	9242	1219	1989	902
40	40	1460	18362	2439	3949	1782
60	60	2180	27482	3659	5909	2662
80	80	2900	36602	4879	7869	3542
100	100	3620	45722	6099	9829	4422

Table 1. The numbers of edges in the parse-fragments, after step 1 and after step 4, for two versions of the algorithm, when parsing carpet patterns.

5. Learning by splitting: informal account

We have seen in section 3 that homomorphisms are useful for specifying the parsing problem without artificial sequentiality. They are also essential to describing the way the grammar is learned. This section describes the principles of the learning method informally; the formal algorithm is derived in the next section. Consider the example in figure 9.

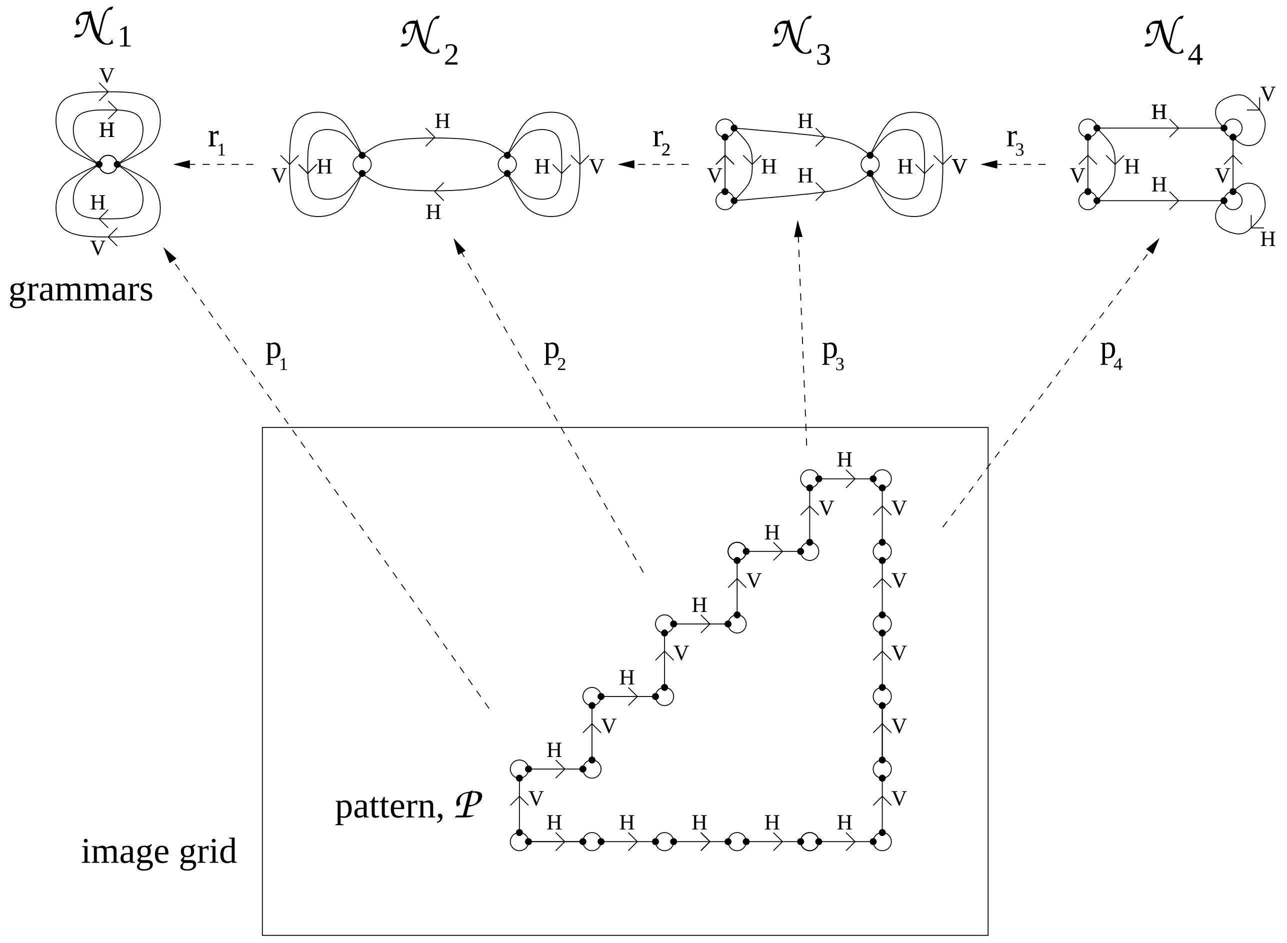


Figure 9. An example of how a grammar is learned, through a sequence of steps $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4$. The homomorphisms p_1, p_2, p_3, p_4 are parses of a typical pattern, $\mathcal{P}$; the homomorphisms r_1, r_2, r_3 describe the refinement steps.

The patterns are isosceles right-angled 'triangles' consisting of a vertical segment, a horizontal segment, and a 'diagonal' segment made of alternate horizontals and verticals. These patterns are presented one at a time in the image grid and may occur at any position and be of any size. The task is to learn a grammar to represent them. We begin with an *initial* grammar $\mathcal{N}_1$, which is consistent with any pattern that is a simple closed curve and hence embodies little or no grammatical knowledge. We proceed to *refine* the grammar through a sequence of steps, producing $\mathcal{N}_2$, $\mathcal{N}_3$, $\mathcal{N}_4$, the last of which describes the pattern population precisely. Each network $\mathcal{N}_{i+1}$ is obtained from $\mathcal{N}_i$ by node splitting followed by pruning of unwanted edges (see section 3.3).

The parses are represented by homomorphisms p_1, p_2, p_3, p_4 from the pattern network $\mathcal{P}$ to the grammar networks $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4$ respectively. There are also homomorphisms $r_i : \mathcal{N}_{i+1} \to \mathcal{N}_i$ representing the refinement operations, for $i = 1, 2, 3$. The diagram commutes: that is, $p_1 = r_1 \circ p_2$, $p_2 = r_2 \circ p_3$, and $p_3 = r_3 \circ p_4$. A network $\mathcal{N}'$ is said to be a *refinement* of a network $\mathcal{N}$ iff there is a homomorphism $r : \mathcal{N}' \to \mathcal{N}$; if $\mathcal{N}'$ is a refinement of $\mathcal{N}$ then $L(\mathcal{N}') \subseteq L(\mathcal{N})$ and $GL(\mathcal{N}') \subseteq GL(\mathcal{N})$, since for every parse $p' : \mathcal{P} \to \mathcal{N}'$ there is a parse $r \circ p' : \mathcal{P} \to \mathcal{N}$.

Now we can restate the learning problem as follows: given an initial network $\mathcal{N}_1$, find a sequence of refinements $\mathcal{N}_1 \xleftarrow{r_1} \mathcal{N}_2 \xleftarrow{r_2} \mathcal{N}_3 \xleftarrow{r_3} \cdots \xleftarrow{r_{n-1}} \mathcal{N}_n$ such that $GL(\mathcal{N}_n)$ equals the pattern population. The initial network $\mathcal{N}_1$ is defined as follows. If the patterns being considered have nodes with allowed numbers of hooks $k_1, k_2, \ldots k_m$ then $\mathcal{N}_1$ will consist of m nodes, with $k_1, k_2, \ldots k_m$ hooks respectively, and with an edge of every possible label and direction connecting every hook to every other hook.

Proposition 7 of section 3.3 may be used to shed light on the tractability of the learning problem. Suppose that at some stage of learning we have a grammar network $\mathcal{N}$ and that we *ought* to refine it, in one or more steps, to a grammar network $\mathcal{N}'$ that represents the pattern population better, but due to a mistake in the learning process we *actually* refine $\mathcal{N}$ in one or more steps to another network, $\mathcal{N}''$. The question is, can we recover from our mistake by refining $\mathcal{N}''$ to $\mathcal{N}'$ or to some equally good network? Or is $\mathcal{N}''$ a dead end, from which we cannot escape by further refinements? Proposition 7 provides us with a canonical way of recovering, namely by refining $\mathcal{N}''$ to the pullback network $\mathcal{N}'''$, which is also a refinement of $\mathcal{N}'$. In the learning process we shall restrict ourselves to simple unambiguous grammar networks whose generated languages are a superset of the given pattern population. The pullback grammar $\mathcal{N}'''$ will also be simple and unambiguous, and will represent the pattern population at least as well as $\mathcal{N}'$ and $\mathcal{N}''$ (by part (d) of the proposition).

This shows that, in principle, learning can proceed directly to a solution by a sequence of refinements, without need for backtracking in case of error.

In the algorithm to be used in this paper, the refinement steps will all consist of splitting of a node, or of a larger portion of the network, combined with pruning of redundant edges produced by the split. The main question is how, given a network $\mathcal{N}$ at some stage of learning, to choose the next refinement step to $\mathcal{N}'$. I have pointed out that, in general, $\mathcal{N}'$ will generate a smaller language than $\mathcal{N}$ and hence embody more grammatical knowledge. However, this is not the full story. Consider again the example node-splitting refinement shown in figure 5 in section 3.3. Recall from section 3.3 that the grammars are stochastic, containing mean multiplicities n_e and n_v for each edge e and node v, satisfying equations (1). The mean multiplicities are related by equations such as

$$\begin{aligned} n_n &= n_u + n_v = n_x + n_y + n_z = n_a + n_b + n_c + n_d + n_e \\ n_{n_+} &= n_{u_+} + n_{v_+} = n_{x_+} + n_{y_+} + n_{z_+} = n_{a_+} + n_{b_+} + n_{c_+} . \end{aligned} \tag{2}$$

These equations hold because any pattern node mapping to n must have an incident edge mapping to either u or v, another edge mapping to one of x, y, z, and a third edge mapping to one of a, b, c, d, e; and similarly for n_+. The mean multiplicities of $\mathcal{N}$ are related to those of $\mathcal{N}'$ by equations such as

$$n_{u_+} + n_{u_-} = n_u, \quad n_{v_+} + n_{v_-} = n_v, \quad n_{n_+} + n_{n_-} = n_n, \quad n_{a_+} = n_a. \tag{3}$$

The mean multiplicities for $\mathcal{N}$ are derivable from those of $\mathcal{N}'$, but those of $\mathcal{N}'$ contain extra information. This extra information is new grammatical knowledge represented in $\mathcal{N}'$ but not in $\mathcal{N}$.

These considerations can be used to choose the next refinement step. Suppose we have the grammar $\mathcal{N}$, and we wish to choose a node to split, the governing hook, and the positive and negative indicator edges. We want to choose the split to maximise the 'extra information' produced; this extra information can be understood in terms of *correlations* between mapping of edges, as follows.

We have seen that any pattern node mapping to n must have three incident edges, one of which has a choice between mapping to a, b, c, d or e, while another has to choose between mapping to u or v. The mean multiplicities $n_a, n_b, \ldots$ record the frequencies with which each of these choices is made, but they do not record the *correlations* between the choices. In the absence of information about correlations let us assume, by default, that the choices are uncorrelated. On this assumption we would estimate that, if n is split as in figure 5, we would have

$$\begin{aligned}
n_{u_+} &= \frac{n_{n_+} n_u}{n_n}, & n_{u_-} &= \frac{n_{n_-} n_u}{n_n}, & n_{v_+} &= \frac{n_{n_+} n_v}{n_n}, & n_{v_-} &= \frac{n_{n_-} n_v}{n_n}, \\
n_{x_+} &= \frac{n_{n_+} n_x}{n_n}, & n_{x_-} &= \frac{n_{n_-} n_x}{n_n}, & n_{y_+} &= \frac{n_{n_+} n_y}{n_n}, & n_{y_-} &= \frac{n_{n_-} n_y}{n_n}, \\
n_{z_+} &= \frac{n_{n_+} n_z}{n_n}, & n_{z_-} &= \frac{n_{n_-} n_z}{n_n}.
\end{aligned} \tag{4}$$

since n_{u_+}, for example, is the frequency with which a node has an incident edge mapping to a, b or c *and* an edge mapping to u, under p. Now, the refined network $\mathcal{N}'$ is able to record the *actual* values for $n_{u_\pm}, \ldots n_{z_\pm}$. If the actual values agree with the estimated values, that is, if there really is no correlation, then there is no point in splitting in this way, since refining $\mathcal{N}$ to $\mathcal{N}'$ produces no extra grammatical knowledge. If the actual values differ markedly from the estimated values then there is some benefit in the split, as it brings to light statistical grammatical regularities not observable in $\mathcal{N}$. The size of the correlation gives a measure of the *fissility* of n, i.e. its suitability for splitting. We want to find the way of splitting $\mathcal{N}$ that maximises the fissility measure.

Exactly how to define the measure of correlation is not immediately clear. We want to measure the extent to which the equations (4) are violated, and we need a measure that makes a fair comparison between different candidate splits on different nodes involving different numbers of hooks, different numbers of incident edges, and different magnitudes of mean multiplicities.

A principled and consistent way to do this is to define a numerical objective function *Obj* on networks, such that our aim is to maximise *Obj*($\mathcal{N}$) while minimising the complexity of $\mathcal{N}$, as measured by a cost function *Cost*($\mathcal{N}$). Then we can define the fissility of a possible split by

$$\text{fissility} = \frac{\mathit{Obj}(\mathcal{N}') - \mathit{Obj}(\mathcal{N})}{\mathit{Cost}(\mathcal{N}') - \mathit{Cost}(\mathcal{N})}. \tag{5}$$

Fissility should always be non-negative and should be zero when and only when the equations (4) hold. Defining fissility in this way gives coherence to the whole learning process, since it means the algorithm is pursing the same goal at each learning step, namely to maximise the ratio $(Obj(\mathcal{N}) - Obj(\mathcal{N}_1))/(Cost(\mathcal{N}) - Cost(\mathcal{N}_1))$, where $\mathcal{N}_1$ is the initial network. The *Obj* and *Cost* functions will be defined at the start of the next section.

So far we have considered the simplest possible type of refinement, in which one node is split into two. It is sometimes desirable to split a larger portion of the network than a single node. An example of this is given in figure 10, which shows part of a grammar network $\mathcal{N}$ and a corresponding part of a pattern $\mathcal{P}$. When $\mathcal{P}$ is parsed using $\mathcal{N}$, $\mathcal{P}$ must have an edge mapping to a or b, followed by zero or more edges mapping to c, then an edge mapping to d or e; this can be written using regular expression notation as $(a|b)c^*(d|e)$. Now suppose that the choice between a and b is highly correlated with the choice between d and e: that is, the combinations ac^*d and bc^*e are much more common than the combinations ac^*e and bc^*d. Then, to represent this correlation, we should split the grammar to $\mathcal{N}'$ as shown in figure 10. The portion of $\mathcal{N}$ consisting of node n and edge c has been duplicated as a unit (instead of just duplicating a single node, as we did in the previous example); h is the governing hook, a is a positive indicator edge and b a negative indicator edge; the edges d and e are not part of the portion but they have been duplicated as a consequence of the split. (If the correlation is perfect then we can prune the new edges d_- and e_+, as n_{d_-} and n_{e_+} will be zero.)

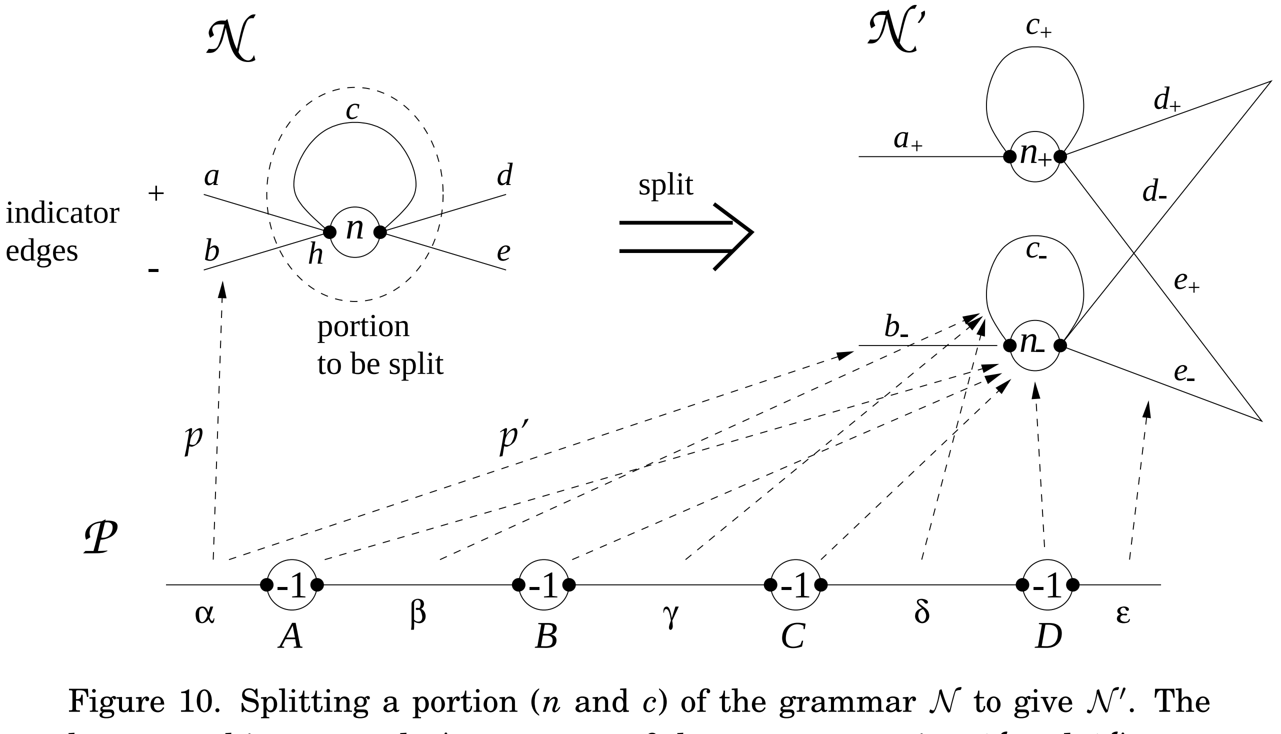


Figure 10. Splitting a portion (n and c) of the grammar $\mathcal{N}$ to give $\mathcal{N}'$. The homomorphisms p and p' are parses of the pattern $\mathcal{P}$ using $\mathcal{N}$ and $\mathcal{N}'$.

How does the network $\mathcal{N}$ determine the fissility of such a split? One way would be to calculate $Obj(\mathcal{N})$ and $Cost(\mathcal{N})$, carry out the split to $\mathcal{N}'$, and then calculate $Obj(\mathcal{N}')$

and $Cost(\mathcal{N}')$ and apply equation (5). However, it would be computationally expensive to do this for every possible way of splitting $\mathcal{N}$; we want to evaluate many possible splits of $\mathcal{N}$ in parallel. So we need a way whereby $\mathcal{N}$ can estimate $Obj(\mathcal{N}')-Obj(\mathcal{N})$ and $Cost(\mathcal{N}')-Cost(\mathcal{N})$ without actually performing the split to $\mathcal{N}'$. Consider the example in figure 10 again. A pattern arrives and is parsed using $\mathcal{N}$, with $\alpha, A, \beta, B, \gamma, C, \delta, D, \varepsilon$ mapping to $b, n, c, n, c, n, c, n, e$ respectively. $\mathcal{N}$ has to imagine what the parse would be if it were split into $\mathcal{N}'$; would γ map to c_+ or c_-, for example? Now, there is only one possibility for α: it must map to b_-. Hence A must map to n_-, hence β must map to c_-, hence B must map to n_-, γ must map to c_-, C must map to n_-, δ must map to c_-, D must map to n_-, and finally ε must map to e_-. We can record these conclusions by putting '-1' in the relevant nodes A, B, C, D of the pattern, to indicate that this part of the pattern maps to the negative copy of the portion in $\mathcal{N}'$. If α had mapped to a instead of b in $\mathcal{N}$ then $A, \beta, B, \gamma, C, \delta, D, \varepsilon$ would have had to map to the positive copy of the portion in $\mathcal{N}'$, so we would have put '$+1$' in the nodes A, B, C, D.

The numbers '-1' and '$+1$' express the additional information necessary to turn a parse using $\mathcal{N}$ into a parse using $\mathcal{N}'$. It is helpful to think of these numbers as two 'colours' and to think of the assignment of the ±1 numbers as a graph-colouring problem. The colour at A is determined directly by whether α maps to a or b; then the colour is propagated along β, γ, δ to B, C, D. Call the edges β, γ, δ *transparent* because colour can propagate along them from node to node: in general an edge is called transparent if and only if it is in the portion to be split or is mapped under p to an edge in the portion (thus, c, β, γ, δ are transparent, $a, b, d, e, \alpha, \varepsilon$ are not). Colour always propagates in a consistent direction: away from the indicator edges. Thus for example the colour of B arrives through β rather than γ. This is specified by designating one of the hooks of each node in the portion as a *governing* hook (h in this case), and the hooks in the pattern that map to a governing hook are also called governing hooks. Thus, the colour of a pattern node is always obtained through its governing hook.

By this graph-colouring process $\mathcal{N}$ is able to simulate parsing using $\mathcal{N}'$. Each node and edge in $\mathcal{N}$ keeps a record of the *mean* colour over all nodes or edges mapping to it, over all patterns. From these mean colours, together with its mean multiplicities $n_a, n_b, n_c, n_d, \ldots$ it can calculate the mean multiplicities $n_{a_+}, n_{b_-}, n_{c_\pm}, n_{d_\pm}, \ldots$ for $\mathcal{N}'$ and thereby calculate $Obj(\mathcal{N}') - Obj(\mathcal{N})$ and $Cost(\mathcal{N}') - Cost(\mathcal{N})$.

The full details of this algorithm are derived formally in the next section. I should like to stress three aspects of this process here:

- it enables the fissility of the split from $\mathcal{N}$ to $\mathcal{N}'$ to be calculated without actually carrying out the split;
- it only requires a modest amount of extra computation (the propagation of colours in the pattern, the updating of mean colours in $\mathcal{N}$) in addition to the parsing that $\mathcal{N}$ has to do anyway;
- fissility values for many possible splits can be calculated in parallel and hence the best $\mathcal{N}'$ can be found with a minimum of searching.

6. Derivation of the splitting algorithm

In this section the informal discussion of splitting in the previous section is turned into a precise algorithm, called *Choose-Split*, by which a grammar network $\mathcal{N}$ determines the best way of splitting. A summary of the Choose-Split algorithm is given at the end of this section. It has to determine the portion of the grammar to be split, the positive and negative indicator edges, and the governing hooks for all nodes in the portion.

6.1 Global objective and cost functions

The learning process is guided by an objective function and a cost function. For any grammar network $\mathcal{N} = (N, H, E, L, A, F, S, M)$, we define

$$\begin{aligned} Obj(\mathcal{N}) &= \sum_{e \in E} n_e \ln n_e - \sum_{n \in N} (k_n - 1) n_n \ln n_n \\ Cost(\mathcal{N}) &= -\sum_{n \in N} (k_n - 1) n_n \ln n_n . \end{aligned} \tag{6}$$

where n_e and n_n are the mean multiplicities defined in equations (1) of section 3.3, and k_n is $|A^{-1}(\{n\})|$, the number of hooks of n. We adopt the usual convention that $0 \ln 0 = 0$. Values of the objective and cost functions are usually negative, but this is of no significance since only differences in their values matter. The learning process seeks to refine the initial network $\mathcal{N}_1$ into a network $\mathcal{N}$ that maximises the ratio

$$\frac{Obj(\mathcal{N}) - Obj(\mathcal{N}_1)}{Cost(\mathcal{N}) - Cost(\mathcal{N}_1)}.$$

This is done by maximising the fissility ratio (see equation (5) in the previous section) at each refinement step. The justification for these definitions is that they lead to splitting criteria that conform to the qualitative arguments in the previous section. (As a matter of interest, it can be shown that, in the case of string or tree patterns, $-Obj(\mathcal{N})$ is the entropy of the stochastic language generated by $\mathcal{N}$.)

The values n_e and n_n can be stored in the grammar network at the edge e and the node n, respectively, and calculated by a fading average of the observed multiplicities, $|p^{-1}(\{e\})|$ and $|p^{-1}(\{n\})|$; that is to say, whenever a pattern arrives and has been parsed, n_e and n_n are updated by the following rule

$$\begin{aligned} \forall e \in E \quad n_e &:= n_e + \varepsilon(|p^{-1}(\{e\})| - n_e) \\ \forall n \in N \quad n_n &:= n_n + \varepsilon(|p^{-1}(\{n\})| - n_n) \end{aligned} \tag{7}$$

where ε is a small positive constant. This is done in step 2.6 of the Choose-Split algorithm (see the end of this section).

6.2 Formal definition of splitting

To define the splitting operation we need to introduce a little more notation. It is natural to speak of an edge as having two 'ends' and to consider an edge-end as an object in its own right. For example, in section 5 I spoke of 'indicator edges', but it would be more accurate to speak of 'indicator edge-ends'. Mathematically, an edge-end is represented as a pair (e,h), where e is the edge and h is the hook to which the end is attached. Formally, an *edge-end* is a member of the set

$$EE = \{\, (e,h) \in E \times H \mid h = F(e) \vee h = S(e) \,\} = \{\, (e,F(e)) \mid e \in E \,\} \cup \{\, (e,S(e)) \mid e \in E \,\}.$$

Define a projection function $\pi{:}\, EE \to H$ by $\forall (e,h) \in EE\ \pi(e,h) = h$. Given a homomorphism $f{:}\, \mathcal{N}_1 \to \mathcal{N}_2$ between two networks, we can consider it also to include a mapping $f{:}\, EE_1 \to EE_2$ between their respective sets of edge-ends, defined by $\forall (e,h) \in EE_1\ f(e,h) = (f(e), f(h))$.

The portion of the network that is to be split is identified by specifying the nodes to be split, the governing hooks, the transparent edges, and the positive and negative indicator edge-ends. Formally, define a *portion* P of $\mathcal{N}$ as a quintuple $(N_P, H_P, E_P, I_P^{+1}, I_P^{-1})$, where $N_P \subseteq N$, $H_P \subseteq H$, $E_P \subseteq E$, $I_P^{+1} \subseteq EE$, $I_P^{-1} \subseteq EE$, such that

- for all $e \in E_P$, $A(F(e)) \in N_P$, $A(S(e)) \in N_P$, and either $F(e) \in H_P$ or $S(e) \in H_P$;
- $A|_{H_P}$ is a bijection from H_P to N_P;
- (N_P, E_P), considered as a graph, is connected;
- $I_P^{+1} \cap I_P^{-1} = \emptyset$, and $I_P^{+1} \cup I_P^{-1} = I_P$, where $I_P = \{\, (e,h) \in EE \mid e \notin E_P \wedge h \in H_P \,\}$.

Observe that every node in N_P must have a unique governing hook and every transparent edge must have a governing hook at one end; I_P is the set of all indicator edge-ends.

Now, the task is to define what it means to refine $\mathcal{N}$ by splitting the portion $(N_P, H_P, E_P, I_P^{+1}, I_P^{-1})$. I shall define a split network $\mathcal{N}' = (N', H', E', L', A', F', S', M')$, which is an isomorphic copy of $\mathcal{N}$ except where nodes, hooks and edges have been split, and a homomorphism $r{:}\, \mathcal{N}' \to \mathcal{N}$ specifying the refinement relation between $\mathcal{N}'$ and $\mathcal{N}$. Let

$$N' = N_P \times \{-1,+1\} \ \cup\ (N \setminus N_P) \times \{0\}$$

$$H' = A^{-1}(N_P) \times \{-1,+1\} \ \cup\ (H \setminus A^{-1}(N_P)) \times \{0\}$$

$$\begin{aligned} E' = {} & \{\, (e,c,c) \mid e \in E_P \wedge c \in \{-1,+1\} \,\} \cup \\ & \quad \{\, (e,c,d) \mid e \in E \setminus E_P \wedge c \in K(e,F(e)) \wedge d \in K(e,S(e)) \,\} \end{aligned}$$

$$\text{where } \forall (e,h) \in EE \quad K(e,h) = \begin{cases} \{c\} & \text{if } (e,h) \in I_P^c, \text{ for } c = \pm 1 \\ \{-1,+1\} & \text{if } (e,h) \notin I_P \text{ and } A(h) \in N_P \\ \{0\} & \text{if } A(h) \notin N_P \end{cases}$$

$$L' = L$$

$$\forall (n,c) \in N' \quad r(n,c) = n$$

$$\forall (h,c) \in H' \quad A'(h,c) = (A(h),c) \qquad r(h,c) = h$$

$$\begin{aligned}
&\forall(e,c,d)\in E' \quad F'(e,c,d)=(F(e),c) \quad S'(e,c,d)=(S(e),d)\\
&\forall(e,c,d)\in E' \quad M'(e,c,d)=M(e) \quad r(e,c,d)=e\\
&\forall l\in L' \quad r(l)=l.
\end{aligned}$$

Thus each node n in the portion is split into two nodes, $(n,-1)$ and $(n,+1)$, and each of its hooks is similarly split. For every other node n in $\mathcal{N}$ there is only one node $(n,0)$ in $\mathcal{N}'$. For every edge e in $\mathcal{N}$, going from hook h_1 to hook h_2, the corresponding edges in $\mathcal{N}'$ are of the form (e,c,d), going from the hook (h_1,c) to the hook (h_2,d), where the numbers c,d are colours ± 1, or 0 if the hook is not split. There may be one, two or four edges (e,c,d), depending on whether e is in the portion, whether e's incident nodes are in the portion, and whether either end of e is an indicator edge-end.

If $\mathcal{N}$ is simple then so is $\mathcal{N}'$. If $\mathcal{N}$ is unambiguous then $\mathcal{N}'$ will usually be as well. In what follows I shall assume that $\mathcal{N}$ and $\mathcal{N}'$ are simple and unambiguous (I shall comment on the exceptional case where splitting introduces ambiguity in section 6.8 below).

The mean multiplicities of $\mathcal{N}'$ are defined analogously to those of $\mathcal{N}$:

$$\begin{aligned}
\forall e'\in E' \quad n_{e'} &= \mathrm{Exp}_{\mathcal{P}}\ |p'^{-1}(\{e'\})|,\\
\forall n'\in N' \quad n_{n'} &= \mathrm{Exp}_{\mathcal{P}}\ |p'^{-1}(\{n'\})|,
\end{aligned}$$

where p' is the unique parse $p' : \mathcal{P}\to\mathcal{N}'$. Note that, for all $n\in N$ and $e\in E$,

$$n_e = \sum_{e'\in r^{-1}(\{e\})} n_{e'}, \qquad n_n = \sum_{n'\in r^{-1}(\{n\})} n_{n'}, \tag{8}$$

which is a generalisation of equations (3) in section 5.

6.3 The effect of a split on the objective and cost functions

Splitting $\mathcal{N}$ as specified above produces a change in the objective function, (6), of

$$\begin{aligned}
\Delta Obj &= Obj(\mathcal{N}') - Obj(\mathcal{N})\\
&= \sum_{\substack{e\in E_P\\ c\in\{-1,+1\}}} n_{(e,c,c)} \ln\frac{n_{(e,c,c)}}{n_e} + \sum_{\substack{e\in E\setminus E_P\\ c\in K(e,F(e))\\ d\in K(e,S(e))}} n_{(e,c,d)} \ln\frac{n_{(e,c,d)}}{n_e} - \sum_{\substack{n\in N_P\\ c\in\{-1,+1\}}} (k_n-1) n_{(n,c)} \ln\frac{n_{(n,c)}}{n_n}.
\end{aligned}$$

Call the summations on the right-hand side Term 1, Term 2 and Term 3; I shall take these terms in reverse order and consider how they can be computed from information available in $\mathcal{N}$.

Term 3. At each node $n\in N$ we can define a *mean colour* m_n by

$$m_n = \frac{1}{n_n} \sum_{c \text{ such that } (n,c)\in N'} c\, n_{(n,c)} = \begin{cases} (n_{(n,+1)} - n_{(n,-1)})/n_n & \text{if } n\in N_P,\\ 0 & \text{if } n\notin N_P.\end{cases} \tag{9}$$

Note that, for any $n \in N_P$, $n_{(n,\pm 1)} = n_n \frac{1 \pm m_n}{2}$. Now, if in Term 3 we hold n fixed and evaluate the sum over c, we obtain

$$\sum_{c \in \{-1,+1\}} (k_n - 1) n_{(n,c)} \ln \frac{n_{(n,c)}}{n_n} = (k_n - 1) n_n \Lambda(m_n),$$

where the function $\Lambda \colon [-1,1] \to \mathbf{R}$ is defined by $\forall x \in [-1,1]\ \Lambda(x) = \frac{1+x}{2} \ln \frac{1+x}{2} + \frac{1-x}{2} \ln \frac{1-x}{2}$.

Term 2 can be decomposed into three pieces, which I shall call Term 2.1, Term 2.2 and Term 2.3:

$$\sum_{\substack{e \in E \setminus E_P \\ c \in K(e,F(e)) \\ d \in K(e,S(e))}} n_{(e,c,d)} \ln \frac{n_{(e,c,d)}}{n_e} = \sum_{\substack{e \in E \setminus E_P \\ c \in K(e,F(e))}} n_{(e,c,\bullet)} \ln \frac{n_{(e,c,\bullet)}}{n_e} + \sum_{\substack{e \in E \setminus E_P \\ d \in K(e,S(e))}} n_{(e,\bullet,d)} \ln \frac{n_{(e,\bullet,d)}}{n_e} + \sum_{e \in E \setminus E_P} R_e$$

where, for any $e \in E$, any $c \in K(e,F(e))$, and any $d \in K(e,S(e))$,

$$n_{(e,c,\bullet)} = \sum_{d \text{ such that } (e,c,d) \in E'} n_{(e,c,d)}, \qquad n_{(e,\bullet,d)} = \sum_{c \text{ such that } (e,c,d) \in E'} n_{(e,c,d)},$$
$$R_e = \sum_{c,d \text{ such that } (e,c,d) \in E'} n_{(e,c,d)} \ln \frac{n_{(e,c,d)} n_e}{n_{(e,c,\bullet)} n_{(e,\bullet,d)}}.$$

Note that $\sum_{c \in K(e,F(e))} n_{(e,c,\bullet)} = n_e = \sum_{d \in K(e,S(e))} n_{(e,\bullet,d)}$.

Term 2.1. Define a *mean colour* m_i for each edge-end $i = (e,h)$ by

$$m_i = \begin{cases} \frac{1}{n_e} \sum_{c \in K(e,F(e))} c\, n_{(e,c,\bullet)} & \text{if } i = (e,F(e)), \\ \frac{1}{n_e} \sum_{d \in K(e,S(e))} d\, n_{(e,\bullet,d)} & \text{if } i = (e,S(e)); \end{cases} \tag{10}$$

for convenience I shall also use the notation n_i,

$$n_i = n_e \quad \text{where } i = (e,h).$$

Note that, for any node n and hook h with $A(h) = n$,

$$n_n = \sum_{i \in \pi^{-1}(\{h\})} n_i, \qquad n_n m_n = \sum_{i \in \pi^{-1}(\{h\})} n_i m_i, \tag{11}$$

which conveys the same information as equations (2) in the example in section 5.

Now, for an edge-end $i = (e,F(e))$, with $e \in E \setminus E_P$, we can evaluate the partial sum over c in Term 2.1, holding e fixed, by

$$\sum_{c \in K(e,F(e))} n_{(e,c,\bullet)} \ln \frac{n_{(e,c,\bullet)}}{n_e} = \begin{cases} n_i \Lambda(m_i) & \text{if } A(F(e)) \in N_P, \\ 0 & \text{if } A(F(e)) \notin N_P. \end{cases}$$

Term 2.2. Similarly, for an edge-end $i = (e, S(e))$, with $e \in E \setminus E_P$, we can evaluate the partial sum over d in Term 2.2, holding e fixed, by

$$\sum_{d \in K(e,S(e))} n_{(e,\bullet,d)} \ln \frac{n_{(e,\bullet,d)}}{n_e} = \begin{cases} n_i \Lambda(m_i) & \text{if } A(S(e)) \in N_P, \\ 0 & \text{if } A(S(e)) \notin N_P. \end{cases}$$

Term 2.3. The term R_e is always non-negative on account of the following standard property of the ln function.

PROPOSITION 8. If $\forall \alpha \in \{1, \ldots N\}$ $p_\alpha, q_\alpha > 0$ and $\sum_{\alpha=1}^{N} p_\alpha = \sum_{\alpha=1}^{N} q_\alpha$ then

$$\sum_{\alpha=1}^{N} p_\alpha \ln \frac{p_\alpha}{q_\alpha} \geq 0,$$

with equality iff $\forall \alpha$ $p_\alpha = q_\alpha$.

The term R_e measures the correlation between the two colours c, d in $n_{(e,c,d)}$: it vanishes when and only when $n_{(e,c,d)} = n_{(e,c,\bullet)} n_{(e,\bullet,d)} / n_e$ for all c, d. The term R_e can be non-zero only for an edge e that is split into four edges (e, c, d), where $c, d \in \{-1, +1\}$; this implies that the edge has both its incident nodes, $A(F(e))$ and $A(S(e))$, in the portion but is itself outside the portion. Such an edge is called a *re-entrant* edge, and R_e is called the *re-entrant correction*.

Term 1. For an edge $e \in E_P$ we have the same m_i value for both its ends (that is, $m_{(e,F(e))} = m_{(e,S(e))}$). Thus we can evaluate the partial sum over c in Term 1, holding e fixed, by

$$\sum_{c \in \{-1,+1\}} n_{(e,c,c)} \ln \frac{n_{(e,c,c)}}{n_e} = n_i \Lambda(m_i)$$

where i is either end of e.

We can combine Terms 2.1, 2.2 and 1 into a single expression,

$$\sum_{j \in J_P} n_j \Lambda(m_j)$$

where

$$J_P = \{ (e, h) \in EE \mid h \in A^{-1}(N_P) \setminus H_P \}.$$

We need to check that this expression counts the correct $n_i \Lambda(m_i)$ values. For an edge e in E_P, $n_i \Lambda(m_i)$ is counted for *one* of the ends of e, as required, since one end of e will be attached to a governing hook and the other will not be. For any other edge, the ends will be counted iff they are attached to nodes in N_P, as required – with the exception of indicator edge-ends i; these are omitted from the sum, but this makes no difference since, for such an i, $K(i)$ is $\{-1\}$ or $\{+1\}$ and hence $m_i = \pm 1$ and $\Lambda(m_i) = 0$.

Gathering together these simplifications, the total change in the objective function *Obj* due to the split is

$$\Delta Obj = \sum_{j \in J_P} n_j \Lambda(m_j) - \sum_{n \in N_P} (k_n - 1) n_n \Lambda(m_n) + \sum_{e \in E \setminus E_P} R_e.$$

The m_j and m_n values are based on information locally available at j and n, so the network is able to calculate the first two terms in this expression. It is not able to calculate the re-entrant corrections R_e, so it takes them as zero. Thus the network estimates ΔObj by computing

$$\begin{aligned} U(P) &= \sum_{j\in J_P} n_j\Lambda(m_j) - \sum_{n\in N_P}(k_n-1)n_n\Lambda(m_n) \\ &= \sum_{n\in N_P}\ \sum_{h\in A^{-1}(\{n\})\setminus H_P}\ \sum_{j\in\pi^{-1}(\{h\})} n_j(\Lambda(m_j)-\Lambda(m_n)). \end{aligned} \tag{12}$$

In general, $U(P) \le \Delta Obj$, with equality iff all the re-entrant corrections are 0. Thus $U(P)$ measures the increase in the objective function Obj guaranteed by local information, neglecting any further increase due to correlations between the colours at opposite ends of the re-entrant edges.

PROPOSITION 9. $U(P) \ge 0$, with equality iff $\forall j\in J_P\ m_j = m_{A(\pi(j))}$.

Proof. For any $n \in N_P$ and any $h \in A^{-1}(\{n\}) \setminus H_P$ we have, using (11),

$$\sum_{j\in\pi^{-1}(\{h\})} n_j\tfrac{1\pm m_j}{2} = n_n\tfrac{1\pm m_n}{2} = \sum_{j\in\pi^{-1}(\{h\})} n_j\tfrac{1\pm m_n}{2}.$$

Therefore,

$$\begin{aligned} &\sum_{j\in\pi^{-1}(\{h\})} n_j(\Lambda(m_j)-\Lambda(m_n)) \\ &\quad= \sum_{j\in\pi^{-1}(\{h\})} n_j\big(\tfrac{1+m_j}{2}\ln\tfrac{1+m_j}{2} - \tfrac{1+m_n}{2}\ln\tfrac{1+m_n}{2} + \tfrac{1-m_j}{2}\ln\tfrac{1-m_j}{2} - \tfrac{1-m_n}{2}\ln\tfrac{1-m_n}{2}\big) \\ &\quad= \sum_{j\in\pi^{-1}(\{h\})} n_j\big(\tfrac{1+m_j}{2}\ln\tfrac{1+m_j}{1+m_n} + \tfrac{1-m_j}{2}\ln\tfrac{1-m_j}{1-m_n}\big) \end{aligned}$$

which is non-negative by proposition 8, and is zero iff $\forall j\in\pi^{-1}(\{h\})\ m_j = m_n$. Hence the proposition follows from (12). ▮

The change in the cost function, (6), can be calculated exactly from the m_n values:

$$\Delta Cost = Cost(\mathcal{N}') - Cost(\mathcal{N}) = -\sum_{\substack{n\in N_P\\ c\in\{-1,1\}}}(k_n-1)n_{(n,c)}\ln\frac{n_{(n,c)}}{n_n} = V(P)$$

where

$$V(P) = -\sum_{n\in N_P}(k_n-1)n_n\Lambda(m_n) \ge 0 \tag{13}$$

as in Term 3 above.

We shall take $U(P)/V(P)$ as our estimate of fissility. Proposition 9 shows that this satisfies the requirements stipulated in section 5: it is always non-negative, and is zero when and only $\forall j \in J_P\ m_j = m_{A(\pi(j))}$, which is analogous to equations (4) in section 5.

6.4 *Estimating fissility for several splits in parallel*

So far we have just estimated the fissility due to a single split of a portion P. In reality the network needs to consider many possible portions in parallel and choose the one with the highest value for $U(P)/V(P)$. Now, the value of $U(P)/V(P)$ depends only on the mean colours at the nodes of N_P and adjacent edge-ends; therefore it is possible to compute fissilities for two portions in parallel provided they have no nodes in common. In fact, to make the most of the parallelism the network will partition itself into portions $P_1, \dots P_k$, with each node in one portion, and evaluate $U(P_1)/V(P_1), \dots U(P_k)/V(P_k)$ simultaneously.

These portions are represented in the network by storing real numbers t_e, g_h, a_i at each $e \in E$, $h \in H$ and $i \in \bigcup_{r=1}^{k} I_{P_r}$, such that

$$t_e = \begin{cases} 1 & \text{if } e \in \bigcup_{r=1}^{k} E_{P_r} \\ 0 & \text{otherwise} \end{cases} \qquad g_h = \begin{cases} 1 & \text{if } h \in \bigcup_{r=1}^{k} H_{P_r} \\ 0 & \text{otherwise} \end{cases} \qquad a_i \begin{cases} > 0 & \text{if } i \in \bigcup_{r=1}^{k} I_{P_r}^{+1} \\ < 0 & \text{if } i \in \bigcup_{r=1}^{k} I_{P_r}^{-1} \end{cases}$$

Thus, $t_e = 1$ iff e is a transparent edge of one of the portions, $g_h = 1$ iff h is a governing hook, and $sgn(a_i)$ gives the sign of each indicator edge-end i (where sgn is the usual signum function: $sgn(x) = 1$ if $x > 0$, $sgn(x) = -1$ if $x < 0$, $sgn(0) = 0$). These numbers between them determine all the portions $P_1, \dots P_k$. Let

$$I = \bigcup_{r=1}^{k} I_{P_r} = \{\, (e,h) \in EE \mid t_e = 0 \wedge g_h = 1 \,\}, \qquad J = \bigcup_{r=1}^{k} J_{P_r} = \{\, (e,h) \in EE \mid g_h = 0 \,\}.$$

6.5 *Propagation of colour*

Let $\mathcal{P} = (N^*, H^*, E^*, L^*, A^*, F^*, S^*, M^*)$ be a pattern, let EE^* be its set of edge-ends and $\pi^*\colon EE^* \to H^*$ be the projection function with $\forall (e^*, h^*) \in EE^*\ \pi^*(e^*, h^*) = h^*$. Let $p\colon \mathcal{P} \to \mathcal{N}$ be the unique parse of $\mathcal{P}$ using the grammar $\mathcal{N}$, and let $p'\colon \mathcal{P} \to \mathcal{N}'$ be the parse of $\mathcal{P}$ using the refined grammar $\mathcal{N}'$. Recall from section 5 that $\mathcal{N}$ simulates $\mathcal{N}'$ by assigning colours to the nodes of $\mathcal{P}$. If a node $n^* \in N^*$ maps to $p(n^*) \in N$ and is assigned a colour c then this means that, if $\mathcal{P}$ were parsed according to $\mathcal{N}'$ rather than $\mathcal{N}$, then n^* would map to $p'(n^*) = (p(n^*), c)$ instead of $p(n^*)$. Similarly, a hook $h^* \in H^*$, presently mapping to $p(h^*)$, would map to $p'(h^*) = (p(h^*), c)$, where c is the colour of the node $A^*(h^*)$; and an edge $e^* \in E^*$ would map to $p'(e^*) = (p(e^*), c, d)$, where c and d are the colours of the nodes $A^*(F^*(e^*))$ and $A^*(S^*(e^*))$ respectively.

Hence we have

$$\begin{aligned}
&\forall (n,c) \in N' \quad n_{(n,c)} = \mathrm{Exp}_{\mathcal{P}}\ |p'^{-1}(\{(n,c)\})| = \mathrm{Exp}_{\mathcal{P}}\ \big|\{\, n^* \in N^* \mid p(n^*) = n \wedge c_{n^*} = c \,\}\big| \\
&\forall (e,c,d) \in E' \quad n_{(e,c,d)} = \mathrm{Exp}_{\mathcal{P}}\ \big|\{\, e^* \in E^* \mid p(e^*) = e \wedge c_{A^*(F^*(e^*))} = c \wedge c_{A^*(S^*(e^*))} = d \,\}\big| \\
&\forall e \in E\ \forall c \in K(e, F(e)) \quad n_{(e,c,\bullet)} = \mathrm{Exp}_{\mathcal{P}}\ \big|\{\, e^* \in E^* \mid p(e^*) = e \wedge c_{A^*(F^*(e^*))} = c \,\}\big| \\
&\forall e \in E\ \forall d \in K(e, S(e)) \quad n_{(e,\bullet,d)} = \mathrm{Exp}_{\mathcal{P}}\ \big|\{\, e^* \in E^* \mid p(e^*) = e \wedge c_{A^*(S^*(e^*))} = d \,\}\big|
\end{aligned}$$

where c_{n^*} is the colour assigned to node $n^* \in N^*$. Hence, by (10),

$$\forall e \in E \quad m_{(e,F(e))} = \frac{1}{n_e} \sum_{c \in K(e,F(e))} c\, n_{(e,c,\bullet)} = \frac{1}{n_e} \operatorname{Exp}_{\mathcal{P}} \Big(\sum_{e^* \in p^{-1}(\{e\})} c_{A^*(F^*(e^*))} \Big)$$

$$\forall e \in E \quad m_{(e,S(e))} = \frac{1}{n_e} \sum_{d \in K(e,S(e))} d\, n_{(e,\bullet,d)} = \frac{1}{n_e} \operatorname{Exp}_{\mathcal{P}} \Big(\sum_{e^* \in p^{-1}(\{e\})} c_{A^*(S^*(e^*))} \Big)$$

which can be rewritten as

$$\forall j \in EE \quad m_j = \frac{1}{n_j} \operatorname{Exp}_{\mathcal{P}} \Big(\sum_{j^* \in p^{-1}(\{j\})} c_{A^*(\pi^*(j^*))} \Big). \tag{14}$$

Similarly, by (9),

$$\forall n \in N \quad m_n = \frac{1}{n_n} \operatorname{Exp}_{\mathcal{P}} \Big(\sum_{n^* \in p^{-1}(\{n\})} c_{n^*} \Big). \tag{15}$$

This confirms the interpretation of m_j and m_n as mean colours, taken over all pattern nodes mapping to j or n and over all patterns.

The colours c_{n^*} of the nodes $n^* \in N^*$ are calculated by a process of propagation through the pattern, starting at the indicator edge-ends and proceeding through the transparent edges and governing hooks. This process can be defined formally as follows. Define the set

$$\begin{aligned} \mathcal{E} = \{ (n_1^*, h_1^*, e^*, h_2^*, n_2^*) \mid e^* \in E^* \wedge \{F^*(e^*), S^*(e^*)\} = \{h_1^*, h_2^*\} \\ \wedge A^*(h_1^*) = n_1^* \wedge A^*(h_2^*) = n_2^* \wedge g_{p(h_2^*)} = 1 \}. \end{aligned}$$

For any nodes n_1^*, n_2^* in $\mathcal{P}$, let $\Gamma_{n_1^*,n_2^*} = 1$ if there is a path through $\mathcal{P}$ by which colour can propagate from n_1^* to n_2^*, and $\Gamma_{n_1^*,n_2^*} = 0$ otherwise. Formally, $\Gamma_{n_1^*,n_2^*}$ is defined recursively by

$$\Gamma_{n_1^*,n_2^*} = \begin{cases} 1 & \text{if } n_1^* = n_2^*, \\ t_{p(e^*)} \Gamma_{n_1^*,n^*} & \text{if } n_1^* \neq n_2^* \text{ and } (n^*, h^*, e^*, h_2^*, n_2^*) \in \mathcal{E}, \end{cases} \tag{16}$$

or, equivalently,

$$\Gamma_{n_1^*,n_2^*} = \begin{cases} 1 & \text{if } n_1^* = n_2^*, \\ \sum_{(n_1^*, h_1^*, e^*, h^*, n^*) \in \mathcal{E}} t_{p(e^*)} \Gamma_{n^*,n_2^*} & \text{if } n_1^* \neq n_2^*, \end{cases} \tag{17}$$

where the summation is taken over all quintuples $(n_1^*, h_1^*, e^*, h^*, n^*)$ in $\mathcal{E}$ whose first component equals the given node n_1^*.

Then colour can be defined by

$$c_{n^*} = \sum_{i \in I} \sum_{i^* \in p^{-1}(\{i\})} \Gamma_{A^*(\pi^*(i^*)),\, n^*}\, sgn(a_i). \tag{18}$$

Colour is calculated, from (16), by iterating the following operation for every $(n_1^*, h_1^*, e^*, h_2^*, n_2^*) \in \mathcal{E}$:

$$c_{n_2^*} := \begin{cases} sgn(a_{(p(e^*),p(h_2^*))}) & \text{if } t_{p(e^*)} = 0, \\ c_{n_1^*} & \text{if } t_{p(e^*)} = 1. \end{cases} \tag{19}$$

(This is done in step 2.11 of the Choose-Split algorithm: see the end of this section.) Next, for any $i \in I$ and $j \in J$, define

$$n_{ij} = \mathrm{Exp}_{\mathcal{P}} \Big(\sum_{i^* \in p^{-1}(\{i\})} \sum_{j^* \in p^{-1}(\{j\})} \Gamma_{A^*(\pi^*(i^*)), A^*(\pi^*(j^*))} \Big). \tag{20}$$

Then, from (14), (18) and (20),

$$\forall j \in J \quad m_j = \frac{1}{n_j} \, \mathrm{Exp}_{\mathcal{P}} \Big(\sum_{j^* \in p^{-1}(\{j\})} c_{A^*(\pi^*(j^*))} \Big) = \frac{1}{n_j} \sum_{i \in I} n_{ij} \, sgn(a_i). \tag{21}$$

6.6 Choosing the a_i values

Suppose that values of t_e and g_h have been chosen (which amounts to choosing N_P, E_P and H_P for each portion); then the set of indicator edge-ends I_P is determined for each portion, but the a_i values (specifying which indicator edge-ends are positive and which are negative) have not yet been chosen.

We are seeking values for a_i for each edge-end $i \in I$ to maximise $U(P_r)/V(P_r)$ for each portion P_r. It is simpler, and in practice sufficient, to maximise $U(P_r)$ for each portion. This is equivalent to maximising the total U,

$$U_{\text{total}} = \sum_{r=1}^{k} U(P_r) = \sum_{j \in J} n_j \Lambda(m_j) - \sum_{n \in N} (k_n - 1) n_n \Lambda(m_n), \tag{22}$$

using (12). Consider an alternative formulation of this problem: consider the problem of maximising

$$U' = \sum_{j \in J} n_j \Big(\frac{1+m_j}{2} \ln \frac{1+\overline{m}_j}{1+\overline{m}_{A(\pi(j))}} + \frac{1-m_j}{2} \ln \frac{1-\overline{m}_j}{1-\overline{m}_{A(\pi(j))}} \Big) \tag{23}$$

using new parameters $\overline{m}_j, \overline{m}_n$, which are free to vary in the interval $[-1, 1]$, subject to the constraint $n_n \overline{m}_n = \sum_{j \in \pi^{-1}(\{h\})} n_j \overline{m}_j$ for each hook h with $g_h = 0$ and $A(h) = n$. This optimisation problem can be solved by applying the following two steps alternately:

(i) keeping the a_i parameters fixed, find the values for $\overline{m}_j$ and $\overline{m}_n$ that maximise U';
(ii) keeping the $\overline{m}_j, \overline{m}_n$ parameters fixed, find the values for a_i that maximise U'.

Step (i) is easily performed. By (11), we have

$$\sum_{j \in \pi^{-1}(\{h\})} n_j \frac{1 \pm \overline{m}_j}{1 \pm \overline{m}_n} = n_n = \sum_{j \in \pi^{-1}(\{h\})} n_j \frac{1 \pm m_j}{1 \pm m_n}$$

for any hook h with $g_h = 0$ and $A(h) = n$, so we can apply proposition 8, which shows that

$$\sum_{j \in \pi^{-1}(\{h\})} n_j \frac{1 \pm m_j}{2} \ln \frac{1 \pm \overline{m}_j}{1 \pm \overline{m}_n}$$

is maximised by taking $(1\pm\overline{m}_j)/(1\pm\overline{m}_n) = (1\pm m_j)/(1\pm m_n)$, i.e. $\overline{m}_j = m_j$ and $\overline{m}_n = m_n$. Therefore U' is maximised by taking $\overline{m}_j = m_j$ for all $j \in J$, and $\overline{m}_n = m_n$ for all $n \in N$. The maximum so obtained is

$$U' = \sum_{j\in J} n_j \Big(\frac{1+m_j}{2} \ln \frac{1+m_j}{1+m_{A(\pi(j))}} + \frac{1-m_j}{2} \ln \frac{1-m_j}{1-m_{A(\pi(j))}} \Big) = U_{\text{total}}.$$

This shows that the problem of maximising U' (with parameters $\overline{m}_j, \overline{m}_n$ free to vary) is equivalent to the problem of maximising U_{total} (with $\overline{m}_j$ and $\overline{m}_n$ eliminated), so this formulation of the problem is equivalent to the previous one.

Step (ii) is performed as follows. Consider the effect of changing a single value of a_i from negative to positive. The difference this makes to m_j, $\Delta_i m_j$ (defined as the value of m_j with a_i positive minus the value of m_j with a_i negative), is $2n_{ij}/n_j$, by (21). Hence, from (23), the difference this makes to U' is

$$\begin{aligned}
\Delta_i U' &= \sum_{j\in J} n_j \frac{\Delta_i m_j}{2} \big(\ln \frac{1+\overline{m}_j}{1+\overline{m}_{A(\pi(j))}} - \ln \frac{1-\overline{m}_j}{1-\overline{m}_{A(\pi(j))}}\big) \\
&= \sum_{j\in J} n_j (\Delta_i m_j)(\tanh^{-1}(\overline{m}_j) - \tanh^{-1}(\overline{m}_{A(\pi(j))})) \\
&= \sum_{j\in J} 2n_{ij}(\tanh^{-1}(\overline{m}_j) - \tanh^{-1}(\overline{m}_{A(\pi(j))})) \\
&= \text{Exp}_{\mathcal{P}} \ \big\{ 2 \sum_{j\in J} \sum_{i^*\in p^{-1}(\{i\})} \sum_{j^*\in p^{-1}(\{j\})} \Gamma_{A^*(\pi^*(i^*)),A^*(\pi^*(j^*))}(\tanh^{-1}(\overline{m}_j) - \tanh^{-1}(\overline{m}_{A(\pi(j))})) \big\} \\
&= \text{Exp}_{\mathcal{P}} \ \big\{ 2 \sum_{i^*\in p^{-1}(\{i\})} \delta_{A^*(\pi^*(i^*))} \big\}
\end{aligned}$$

where, for any node $n^* \in N^*$,

$$\delta_{n^*} = \sum_{j\in J} \sum_{j^*\in p^{-1}(\{j\})} \Gamma_{n^*,A^*(\pi^*(j^*))}(\tanh^{-1}(\overline{m}_j) - \tanh^{-1}(\overline{m}_{A(\pi(j))})). \tag{24}$$

The δ_{n^*} values can be computed, from (17), using the following recurrence relation

$$\begin{aligned}
\forall n_1^* \in N^* \quad \delta_{n_1^*} = &\sum_{h_1^*\in A^{*-1}(\{n_1^*\})} (1-g_{h_1^*}) \sum_{j_1^*\in \pi^{*-1}(\{h_1^*\})} (\tanh^{-1}(\overline{m}_{p(j_1^*)}) - \tanh^{-1}(\overline{m}_{p(n_1^*)})) \\
&+ \sum_{(n_1^*,h_1^*,e^*,h_2^*,n_2^*)\in\mathcal{E}} t_{p(e^*)}\delta_{n_2^*}
\end{aligned} \tag{25}$$

where the second summation is taken over every quintuple in $\mathcal{E}$ whose first component equals the given node n_1^*. This recurrence relation shows that the δ values can be calculated by a spreading-activation process flowing backwards (i.e. in the opposite direction to colour) from nodes in $A^*(\pi^*(p^{-1}(J)))$ to nodes in $A^*(\pi^*(p^{-1}(I)))$. (This is done in step 2.8 of the algorithm below.)

Now, if we set $a_i = \frac{1}{2}\Delta_i U'$ (where the purpose of the factor of $\frac{1}{2}$ is merely to get rid of the factor of 2 in the formula for $\Delta_i U'$) then the a_i will be positive or negative as required to maximise U'. This procedure can be carried out simultaneously for each $i \in I$, the result being a set of values for a_i that maximises U', as required.

In summary, this procedure requires the network to store values of a_i for every $i \in I$, $\overline{m}_j$ for every $j \in J$, and $\overline{m}_n$ for every node n. These values are updated for every pattern so as to maintain the conditions

$$\begin{aligned} a_i &= \mathrm{Exp}_{\mathcal{P}} \Big\{ \sum_{i^* \in p^{-1}(\{i\})} \delta_{A^*(\pi^*(i^*))} \Big\} \\ \overline{m}_j &= m_j = \frac{1}{n_j} \mathrm{Exp}_{\mathcal{P}} \Big\{ \sum_{j^* \in p^{-1}(\{j\})} c_{A^*(\pi^*(j^*))} \Big\} \\ \overline{m}_n &= m_n = \frac{1}{n_n} \mathrm{Exp}_{\mathcal{P}} \Big\{ \sum_{n^* \in p^{-1}(\{n\})} c_{n^*} \Big\}. \end{aligned} \tag{26}$$

These updates may be carried out in parallel (rather than alternately, as I said above), since they only involve incremental changes for each pattern. From now on I shall simply write '$\overline{m}_j$' as 'm_j' and '$\overline{m}_n$' as 'm_n'. The precise update rules are shown at steps 2.9 and 2.12 in the algorithm below.

One problem that occurs in practice with this process is that, after the first few patterns, all the a_i values in a portion can become positive, or all negative, in which case all the m_j values will go to $+1$, or all to -1, and the network is then stuck in this state. A good way of preventing this is to enforce the normalisation condition

$$\sum_{i \in I_P} a_i = 0 \tag{27}$$

for each portion P. This condition ought to hold approximately anyway, since from (26) we have

$$\begin{aligned} \sum_{i \in I_P} a_i &= \mathrm{Exp}_{\mathcal{P}} \Big\{ \sum_{i^* \in p^{-1}(I_P)} \delta_{A^*(\pi^*(i^*))} \Big\} \\ &= \mathrm{Exp}_{\mathcal{P}} \Big\{ \sum_{j^* \in p^{-1}(J_P)} (\tanh^{-1}(m_{p(j^*)}) - \tanh^{-1}(m_{p(A^*(\pi^*(j^*)))})) \Big\} \\ &= \sum_{j \in J_P} n_j (\tanh^{-1}(m_j) - \tanh^{-1}(m_{A(\pi(j))})) \end{aligned}$$

which vanishes, by (11), if we approximate $\tanh^{-1} x$ by x.

The normalisation condition (27) is enforced as follows. For each node $n \in N$ calculate

$$\begin{aligned} U_n &:= \sum_{i \in I \cap \pi^{-1}(A^{-1}(\{n\}))} a_i = \sum_{h \in A^{-1}(\{n\})} g_h \sum_{i \in \pi^{-1}(\{h\})} (1 - t_e) a_i \\ V_n &:= \sum_{i \in I \cap \pi^{-1}(A^{-1}(\{n\}))} n_i = \sum_{h \in A^{-1}(\{n\})} g_h \sum_{i \in \pi^{-1}(\{h\})} (1 - t_e) n_i \end{aligned} \tag{28}$$

(thus giving $\sum_{n \in N_P} U_n = \sum_{i \in I_P} a_i$ and $\sum_{n \in N_P} V_n = \sum_{i \in I_P} n_i$). Next, spread the U_n and V_n values evenly throughout the nodes of the portion. Then all nodes n in a single portion P will have common values $U_n = (\sum_{i \in I_P} a_i)/|N_P|$ and $V_n = (\sum_{i \in I_P} n_i)/|N_P|$, so we subtract $n_i U_n / V_n$ from each a_i, thus making (27) true. See step 2.10 of the Choose-Split algorithm for the precise details.

6.7 *Choosing the t_e and g_h values*

I have described how a_i values are found, assuming t_e and g_h values have already been determined. The problem of choosing the t_e and g_h values is much harder. There are hardly any systematic relationships between the fissilities for different assignments of t_e and g_h values; there is no way to find out whether a particular assignment produces a high fissility except by trying it. Hence the algorithm proceeds as follows.

First, a random assignment of t_e and g_h values is chosen (see steps 2.1 and 2.2 of the Choose-Split algorithm). This must be done in a way consistent with the definition of a portion: each node must have one governing hook, and each transparent edge must have a governing hook at one end. Step 2.3 of the algorithm enforces these conditions.

Next, the algorithm chooses a_i values, as in section 6.6 above, and calculates the resulting mean colours and fissilities of the portions. A good estimate of fissility can be obtained quickly from as few as 10 patterns.

Next, all portions with low fissility are recycled: that is, their t_e and g_h values are reset in a random but consistent way, producing a new partitioning of the network into portions. Then the process is repeated. Most choices of t_e and g_h produce very low fissilities and are quickly changed, so the algorithm can examine a large number of possibilities quickly. If portion has a high fissility then it is allowed to persist for a certain number, *MaxAge*, of patterns, to produce an accurate measure of fissility, before being recycled.

We are primarily interested in small portions, involving no more than three or four nodes and edges, so this process of randomly varying t_e and g_h is a feasible search strategy (see the examples in section 9).

During this process, the following records are kept.

At each node $n \in N$, age_n records the age of the current portion containing n, i.e. the number of patterns that have been dealt with since this portion was formed. These values are set to 0 in step 1 of the Choose-Split algorithm and incremented at step 2.7 every time a pattern is processed. Also, every time we decide to recycle a portion, the age_n values are reset to 0 throughout the portion (steps 2.16–2.18); the actual recycling is then carried out in steps 2.1–2.3.

At each node n, $fissility_n$ records the estimated fissility of the current portion, P, containing n. It is calculated, from (12) and (13), by a similar process to the normalisation of the a_i parameters in section 6.6: first we calculate

$$\begin{aligned} U_n &:= \sum_{h \in A^{-1}(\{n\})} (1 - g_h) \sum_{j \in \pi^{-1}(\{h\})} n_j(\Lambda(m_j) - \Lambda(m_n)) \\ V_n &:= -(k_n - 1) n_n \Lambda(m_n) \end{aligned} \tag{29}$$

at each n, giving $\sum_{n \in N_P} U_n = U(P)$ and $\sum_{n \in N_P} V_n = V(P)$; then we spread the U_n and V_n values evenly throughout the portion; then, at each n, $U_n/V_n = U(P)/V(P)$, the estimated fissility of the portion (see step 2.14).

Also, at each node n, $best_n$ records the highest fissility seen so far in any portion containing n. Every time a portion survives to the maximum age, *MaxAge*, its fissility is compared with $best_n$, and $best_n$ is updated if necessary (step 2.16.1).

At each edge $e \in E$, $bestt_e$ records the best value of t_e found so far, i.e. the one used in the most fissile portion containing the nodes at either end of e. Likewise, at each hook $h \in H$, $bestg_h$ records the best value of g_h found so far, i.e. the one used in the most fissile portion containing the node of h. At each edge-end i, $besta_i$ records the best a_i so far. The $bestt_e$, $bestg_h$ and $besta_i$ values are updated at the same time as the $best_n$ values (step 2.16.1).

When this process has been continued over a sufficiently large number of patterns, the best set of portions can be read off from the $bestt_e$, $bestg_h$ and $besta_i$ values. From these, the best portion is selected as the one where the $best_n$ values are highest. (See step 3.)

There is one final way in which the efficiency of this search process can be improved. Many portions considered are unnecessarily large: that is, they contain edges that could be removed without any reduction in fissility. To state this as a precise criterion: a value of $t_e = 1$ should always be changed to 0 if this can be done without reducing $U_{\text{total}} = \sum_{r=1}^{k} U(P_r)$. We can estimate the effect that changing t_e has on U_{total} by calculating $\partial U_{\text{total}}/\partial t_e$. For this, we have to think of t_e temporarily as a parameter that can vary continuously between 0 and 1, and rewrite the rule for colour propagation, (19), as

$$c_{n_2^*} := (1 - t_{p(e^*)})\, sgn(a_{(p(e^*),p(h_2^*))}) + t_{p(e^*)}\, c_{n_1^*} \tag{30}$$

for every $(n_1^*, h_1^*, e^*, h_2^*, n_2^*) \in \mathcal{E}$. Now, let e be an edge in E with $t_e = 1$, and let $j = (e, h_1)$ and $i = (e, h_2)$ be its edge-ends, with $g_{h_1} = 0$ and $g_{h_2} = 1$. Then, from (22), (14) and (15),

$$\begin{aligned}
\frac{\partial U_{\text{total}}}{\partial t_e} &= \sum_{j\in J} n_j \tanh^{-1}(m_j) \frac{\partial m_j}{\partial t_e} \; - \sum_{n\in N} (k_n - 1) n_n \tanh^{-1}(m_n) \frac{\partial m_n}{\partial t_e} \\
&= \sum_{j\in J} \tanh^{-1}(m_j)\, \mathrm{Exp}_{\mathcal{P}} \left\{ \sum_{j^*\in p^{-1}(\{j\})} \frac{\partial c_{A^*(\pi^*(j^*))}}{\partial t_e} \right\} \\
&\qquad - \sum_{n\in N} (k_n - 1) \tanh^{-1}(m_n)\, \mathrm{Exp}_{\mathcal{P}} \left\{ \sum_{n^*\in p^{-1}(\{n\})} \frac{\partial c_{n^*}}{\partial t_e} \right\} \\
&= \mathrm{Exp}_{\mathcal{P}} \left\{ \sum_{j\in J} \tanh^{-1}(m_j) \sum_{j^*\in p^{-1}(\{j\})} \frac{\partial c_{A^*(\pi^*(j^*))}}{\partial t_e} \right. \\
&\qquad \left. - \sum_{n\in N} (k_n - 1) \tanh^{-1}(m_n) \sum_{n^*\in p^{-1}(\{n\})} \frac{\partial c_{n^*}}{\partial t_e} \right\} \\
&= \mathrm{Exp}_{\mathcal{P}} \left\{ \sum_{j\in J} \left(\tanh^{-1}(m_j) - \tanh^{-1}(m_{A(\pi(j))})\right) \sum_{j^*\in p^{-1}(\{j\})} \frac{\partial c_{A^*(\pi^*(j^*))}}{\partial t_e} \right\}
\end{aligned}$$

Now, we can evaluate $\partial c_{n^*}/\partial t_e$, for any node $n^* \in N^*$, by the following argument. Changing t_e will change the colour that flows through each edge $e^* \in p^{-1}(\{e\})$. In fact, for each $(n_1^*, h_1^*, e^*, h_2^*, n_2^*) \in \mathcal{E}$ such that $p(e^*) = e$, the colour arriving at n_2^* is changed, with a multiplying factor of $c_{n_1^*} - sgn(a_i)$, by (30). Each of these changes will have an effect on c_{n^*} iff there is a path from n_2^* to n^*. Hence

$$\frac{\partial c_{n^*}}{\partial t_e} = \sum_{\substack{(n_1^*, h_1^*, e^*, h_2^*, n_2^*) \in \mathcal{E} \\ \text{such that } p(e^*)=e}} (c_{n_1^*} - sgn(a_i))\, \Gamma_{n_2^*, n^*}.$$

This allows us to complete our calculation:

$$\begin{aligned} \frac{\partial U_{\text{total}}}{\partial t_e} &= \mathrm{Exp}_{\mathcal{P}} \Bigg\{ \sum_{j \in J} \left(\tanh^{-1}(m_j) - \tanh^{-1}(m_{A(\pi(j))})\right) \\ &\qquad \times \sum_{j^* \in p^{-1}(\{j\})} \sum_{\substack{(n_1^*, h_1^*, e^*, h_2^*, n_2^*) \in \mathcal{E} \\ \text{such that } p(e^*)=e}} (c_{n_1^*} - sgn(a_i))\, \Gamma_{n_2^*, A^*(\pi^*(j^*))} \Bigg\} \\ &= \mathrm{Exp}_{\mathcal{P}} \Bigg\{ \sum_{\substack{(n_1^*, h_1^*, e^*, h_2^*, n_2^*) \in \mathcal{E} \\ \text{such that } p(e^*)=e}} (c_{n_1^*} - sgn(a_i))\, \delta_{n_2^*} \Bigg\} \end{aligned} \tag{31}$$

using (24). In this expression, i is the edge-end that would become an indicator if we changed t_e from 1 to 0. We assume its sign, $sgn(a_i)$, would be the same as that of the mean colour presently flowing through the edges $e^* \in p^{-1}(\{e\})$, which is m_j. Hence we take $sgn(a_i)$ to be equal to $sgn(m_j)$.

The value of this partial derivative, $\partial U_{\text{total}}/\partial t_e$, is calculated and stored as θ_e (see step 2.13 of the Choose-Split algorithm below). If an edge e with $t_e = 1$ has a low value of θ_e then t_e is set to 0 in step 2.19, thus simplifying the portion.

6.8 Checking for ambiguity

The splitting process normally produces unambiguous grammar networks. If the grammar $\mathcal{N}$ is unambiguous and we choose a portion in which definite colours ± 1 can be assigned to all pattern nodes, then when we split the portion the grammar will remain unambiguous. However, it is possible that for some portions and some patterns, some nodes n^* receive no colour because there is no edge-end $i^* \in p^{-1}(I)$ with $\Gamma_{A^*(\pi^*(i^*)), n^*} = 1$ (see equation (18)). In this case if we were to split the portion the grammar network would become ambiguous. Portions of this sort are rare and can be detected by the presence of a zero colour. Step 2.15 carries out this check: whenever a pattern node has a colour of 0 the corresponding portion of $\mathcal{N}$ is given an *age* of 0, which causes it to be recycled next time we reach step 2.1.

6.9 Summary of the Choose-Split algorithm

The argument above describes how to determine the best way to split the network. This procedure is carried out by the Choose-Split algorithm, as follows.

In the grammar network, $\mathcal{N}$, the following real numbers are stored: $n_n, m_n, age_n, best_n, U_n, V_n, fissility_n$ at each node n; $g_h, bestg_h$ at each hook h; $n_e, t_e, bestt_e, \theta_e$ at each edge e; and $m_i, a_i, besta_i$ at each edge-end i. For each pattern the real numbers c_{n^*}, δ_{n^*} are stored at each node n^* of the pattern. There are five global parameters, for which I use the following values: $\varepsilon = 0.001, \Theta = 0.1, NumPatterns = 1000, MaxAge = 150, CheckInterval = 10$. The algorithm is as follows.

```
(1) Set the initial values of the parameters:
    $\forall e \in E \; bestt_e := 0$
    $\forall i \in EE \; besta_i := 0$
    $\forall h \in H \; bestg_h := 0$
    $\forall n \in N \; best_n := 0, \; age_n := 0$.
(2) Repeat the following sequence of steps NumPatterns times.
  (2.1) For each $e \in E$, if $age_{A(F(e))} = 0$ and $age_{A(S(e))} = 0$ then
            set $t_e := 0$ or 1, randomly, with equal probability; and set $\theta_e := 0$.
  (2.2) For each node $n$, if $age_n = 0$ then
            choose one hook $h$ in $A^{-1}(\{n\})$ randomly (with equal probability) and set
            $g_h := 1$; set $g_h := 0$ for the other hooks $h \in A^{-1}(\{n\})$;
            $\forall h \in A^{-1}(\{n\}) \; \forall i \in \pi^{-1}(\{h\}) \; a_i := 0, \; m_i := random(-0.1, 0.1)$;
            $\forall h \in A^{-1}(\{n\}) \; \forall i \in \pi^{-1}(\{h\}) \; m_i := m_i - \frac{1}{n_n} \sum_{i \in \pi^{-1}(\{h\})} n_i m_i$;
            $m_n := 0$;
            where $random(a,b)$ is a random number chosen from a uniform proba-
            bility distribution over the interval $[a,b]$.
  (2.3) Reconcile the $t_e$ values with the $g_h$ values by repeating the following two
        operations, in a random order, as many times as possible.
    (2.3.1) Select randomly an edge $e$ with $t_e = 1$ and $g_{F(e)} = 0 = g_{S(e)}$, choose
            $h := F(e)$ or $S(e)$ randomly, set $g_h := 1$, and set $g_{h'} := 0$ for all hooks $h'$
            other than $h$ in $A^{-1}(\{A(h)\})$.
    (2.3.2) Select randomly an edge $e$ with $t_e = 1$ and $g_{F(e)} = 1 = g_{S(e)}$, choose
            $h := F(e)$ or $S(e)$ randomly, set $g_h := 0$, and set $g_{h'} := 1$ for *one* randomly
            chosen hook $h'$ other than $h$ in $A^{-1}(\{A(h)\})$.
  (2.4) Receive a pattern $\mathcal{P} = (N^*, H^*, E^*, L^*, A^*, F^*, S^*, M^*)$ from the environment.
  (2.5) Parse the pattern, using the parsing algorithm in section 4, giving the homo-
        morphism $p : \mathcal{P} \to \mathcal{N}$.
  (2.6) Update the $n$ values:
```

$$\begin{aligned} \forall e \in E \quad n_e &:= n_e + \varepsilon(|p^{-1}(\{e\})| - n_e) \\ \forall n \in N \quad n_n &:= n_n + \varepsilon(|p^{-1}(\{n\})| - n_n) \end{aligned} \qquad \text{cf (7)}$$

```
  (2.7) Update the age values: $\forall n \in N \; age_n := age_n + 1$.
  (2.8) Propagate the $\delta$ values through the pattern, as follows.
```

(2.8.1) For each $n^* \in N^*$, $\delta_{n^*} := 0$.

(2.8.2) Repeat the following operation at each $n_1^* \in N^*$ until convergence

$$\delta_{n_1^*} := \sum_{h_1^* \in A^{*-1}(\{n_1^*\})} (1 - g_{h_1^*}) \sum_{j_1^* \in \pi^{*-1}(\{h_1^*\})} (\tanh^{-1}(m_{p(j_1^*)}) - \tanh^{-1}(m_{p(n_1^*)})) \quad + \sum_{(n_1^*, h_1^*, e^*, h_2^*, n_2^*) \in \mathcal{E}} t_{p(e^*)} \delta_{n_2^*}$$

where the second summation is over all quintuples in $\mathcal{E}$ whose first element equals the given node n_1^* (cf (25)).

(2.9) For every $i \in I$,

$$a_i := a_i + \frac{1}{age_{A(\pi(i))}} \Big(\sum_{i^* \in p^{-1}(\{i\})} \delta_{A^*(\pi^*(i^*))} - a_i \Big) \qquad \text{cf (26)}$$

(2.10) Normalise the a_i values (i.e. enforce equation (27)), as follows.

(2.10.1) For every node $n \in N$,

$$U_n := \sum_{h \in A^{-1}(\{n\})} g_h \sum_{i \in \pi^{-1}(\{h\})} (1 - t_e) a_i$$
$$V_n := \sum_{h \in A^{-1}(\{n\})} g_h \sum_{i \in \pi^{-1}(\{h\})} (1 - t_e) n_i \qquad \text{cf (28)}$$

(2.10.2) Repeat the following at every edge $e \in E$ with $t_e = 1$ until convergence:

$$U_{n_1}, U_{n_2} := \frac{1}{2}(U_{n_1} + U_{n_2}) \qquad V_{n_1}, V_{n_2} := \frac{1}{2}(V_{n_1} + V_{n_2})$$

where $n_1 = A(F(e))$ and $n_2 = A(S(e))$.

(2.10.3) For every $i \in I$, $a_i := a_i - n_i U_{A(\pi(i))} / V_{A(\pi(i))}$.

(2.11) Propagate colour through the pattern, as follows.

(2.11.1) For each $n^* \in N^*$, $c_{n^*} := 0$.

(2.11.2) Repeat the following operation until convergence:

$$\forall (n_1^*, h_1^*, e^*, h_2^*, n_2^*) \in \mathcal{E} \quad c_{n_2^*} := \begin{cases} sgn(a_{(p(e^*), p(h_2^*))}) & \text{if } t_{p(e^*)} = 0, \\ c_{n_1^*} & \text{if } t_{p(e^*)} = 1. \end{cases} \qquad \text{cf (19)}$$

(2.12) Update the mean colours:

$$\forall j \in J \quad m_j := \frac{n_j(age_{A(\pi(j))} - 1) m_j + \sum_{j^* \in p^{-1}(\{j\})} c_{A^*(\pi^*(j^*))}}{n_j(age_{A(\pi(j))} - 1) + |p^{-1}(\{j\})|}$$
$$\forall n \in N \quad m_n := \frac{n_n(age_n - 1) m_n + \sum_{n^* \in p^{-1}(\{n\})} c_{n^*}}{n_n(age_n - 1) + |p^{-1}(\{n\})|} \qquad \text{cf (26)}$$

where each m_j and m_n is limited to the range $[-1, 1]$.

(2.13) For every $e \in E$ such that $t_e = 1$,

$$\theta_e := \theta_e + \frac{1}{age_{A(F(e))}} \sum_{\substack{(n_1^*, h_1^*, e^*, h_2^*, n_2^*) \in \mathcal{E} \\ \text{such that } p(e^*) = e}} \big((c_{n_1^*} - sgn(m_j)) \delta_{n_2^*} - \theta_e \big) \qquad \text{cf (31)}$$

where $j = (e, h_1)$ is the end of e with $g_{h_1} = 0$.

(2.14) Calculate fissilities, as follows.

(2.14.1) For every node $n \in N$,

$$\begin{aligned} U_n &:= \sum_{h \in A^{-1}(\{n\})} (1 - g_h) \sum_{j \in \pi^{-1}(\{h\})} n_j(\Lambda(m_j) - \Lambda(m_n)) \\ V_n &:= -(|A^{-1}(\{n\})| - 1) n_n \Lambda(m_n). \end{aligned} \qquad \text{cf (29)}$$

(2.14.2) Repeat the following at every edge $e \in E$ such that $t_e = 1$, until convergence:

$$U_{n_1}, U_{n_2} := \frac{1}{2}(U_{n_1} + U_{n_2}) \qquad V_{n_1}, V_{n_2} := \frac{1}{2}(V_{n_1} + V_{n_2})$$

where $n_1 = A(F(e))$ and $n_2 = A(S(e))$.

(2.14.3) For every $n \in N$, $\mathit{fissility}_n := U_n / V_n$.

(2.15) Check for ambiguity: for every $n \in N$, if, for some $n^* \in p^{-1}(\{n\})$, $c_{n^*} = 0$, then set $\mathit{age}_n := 0$.

(2.16) For each node $n \in N$, if $\mathit{age}_n = \mathit{MaxAge}$ then do the following.

(2.16.1) If $\mathit{best}_n < \mathit{fissility}_n$ then

```
for every $h \in A^{-1}(\{n\})$, do:
    $\mathit{bestg}_h := g_h$;
    for each $i = (e, h) \in \pi^{-1}(\{h\})$,
        if $i \in I$ then $\mathit{besta}_i := a_i$;
        if $\mathit{best}_{n'} < \mathit{fissility}_n$ then $\mathit{bestt}_e := t_e$,
        (where $\{n, n'\} = \{A(F(e)), A(S(e))\}$);
and set $\mathit{best}_n := \mathit{fissility}_n$
```

(2.16.2) $\mathit{age}_n := 0$

(2.17) For each node $n \in N$, if age_n is a multiple of *CheckInterval* and if $\mathit{fissility}_n < \max(0.05, \mathit{best}_n \times \min(0.95, 2 \times \mathit{age}_n / \mathit{MaxAge}))$ then $\mathit{age}_n := 0$.

(2.18) Spread *age* values throughout each portion, by executing the following operation repeatedly at each edge $e \in E$ such that $t_e = 1$, until convergence:

$$\mathit{age}_{n_1}, \mathit{age}_{n_2} := \min(\mathit{age}_{n_1}, \mathit{age}_{n_2})$$

where $n_1 = A(F(e))$ and $n_2 = A(S(e))$.

(2.19) For each edge $e \in E$,

if $t_e = 1$, $\mathit{age}_{A(F(e))}$ is a multiple of *CheckInterval*, and $\theta_e < \Theta$ then set $t_e := 0$ and $a_i := m_j$, where $j = (e, h_1)$ and $i = (e, h_2)$ are the edge-ends of e, with $g_{h_2} = 1$.

(3) Identify the best portion as follows. Consider the graph whose nodes are N and whose edges are those of E with $\mathit{bestt}_e = 1$, and select the connected component

of this graph with the greatest value of $best_n$. Let N_P and E_P be the nodes and edges of this connected component. Let

$$H_P := \{ h \in A^{-1}(N_P) \mid bestg_h = 1 \}$$
$$I_P^{+1} := \{ i = (e,h) \in EE \mid e \notin E_P \wedge h \in H_P \wedge besta_i > 0 \}$$
$$I_P^{-1} := \{ i = (e,h) \in EE \mid e \notin E_P \wedge h \in H_P \wedge besta_i \leq 0 \}.$$

Then the best portion is $(N_P, H_P, E_P, I_P^{+1}, I_P^{-1})$.

7. Merging

Merging is the inverse operation to splitting; a merger is desirable when it simplifies the grammar $\mathcal{N}$, reducing $Cost(\mathcal{N})$ without reducing $Obj(\mathcal{N})$ appreciably. In section 5 I showed that errors in splitting can be corrected by further splitting (which produces a refinement of the desired grammar) followed by merging (which simplifies the grammar to the desired one). In this paper I shall only consider the simplest kind of merger, in which two nodes are merged into one.

The first task is to define this merging operation formally. Let $\mathcal{N} = (N, H, E, L, A, F, S, M)$ be the grammar, and assume that it is unambiguous and simple (these terms were defined in section 3). Let n_1 and n_2 be the nodes in N that are to be merged, and let $b{:}A^{-1}(\{n_1\}) \to A^{-1}(\{n_2\})$ be a bijection specifying how the hooks of n_1 are to be merged with the hooks of n_2. The merger produces a network $\mathcal{N}'$ with a homomorphism $m{:}\mathcal{N} \to \mathcal{N}'$ such that

$$\forall n, n' \in N \ (m(n) = m(n') \Leftrightarrow (n = n' \vee \{n, n'\} = \{n_1, n_2\})),$$
$$\forall h_1 \in A^{-1}(\{n_1\}) \ \forall h_2 \in A^{-1}(\{n_2\}) \ (m(h_1) = m(h_2) \Leftrightarrow b(h_1) = h_2).$$

We can construct $\mathcal{N}'$ and m satisfying these conditions as follows.

$$N' = N \setminus \{n_1\}$$
$$\forall n \in N \ m(n) = \begin{cases} n_2 & \text{if } n = n_1 \\ n & \text{otherwise} \end{cases}$$
$$H' = H \setminus A^{-1}(\{n_1\})$$
$$\forall h \in H' \ A'(h) = A(h)$$
$$\forall h \in H \ m(h) = \begin{cases} b(h) & \text{if } A(h) = n_1 \\ h & \text{otherwise} \end{cases}$$
$$E' = \{ (m(F(e)), m(S(e)), M(e)) \mid e \in E \}$$
$$\forall (h^*, h^\dagger, l) \in E' \quad F'(h^*, h^\dagger, l) = h^* \quad S'(h^*, h^\dagger, l) = h^\dagger \quad M'(h^*, h^\dagger, l) = l$$
$$\forall e \in E \ m(e) = (m(F(e)), m(S(e)), M(e))$$
$$L' = L \qquad \forall l \in L \ m(l) = l.$$

Finally let $\mathcal{N}' = (N', H', E', L', A', F', S', M')$. The network $\mathcal{N}'$ is simple, provided that, for all h in $A^{-1}(\{n_1\})$, there is no edge between h and $b(h)$.

The second task is to identify the changes to the objective and cost functions produced by a merger. Merging uses the same criterion function as splitting, namely

$$\frac{Obj(\mathcal{N}) - Obj(\mathcal{N}')}{Cost(\mathcal{N}) - Cost(\mathcal{N}')}. \qquad \text{cf (5)}$$

The denominator of this ratio is always non-negative (see below). A good merger will be one where the ratio is close to zero or negative. This ratio is called the *immiscibility* (reluctance to merge) of the two nodes, n_1 and n_2, given the bijection $b{:}A^{-1}(\{n_1\}) \to A^{-1}(\{n_2\})$ between their hooks. As with the calculation of fissility in the splitting algorithm, we should like a way of calculating immiscibility without actually carrying out the merger; this would allow us to consider all possible mergers and pick the best one. Hence we need a formula for immiscibility expressed in terms of $\mathcal{N}$; this can be obtained as follows. From (6),

$$\begin{aligned} Cost(\mathcal{N}) - Cost(\mathcal{N}') &= -(k-1)n_{n_1}\ln n_{n_1} - (k-1)n_{n_2}\ln n_{n_2} \\ &\quad + (k-1)(n_{n_1}+n_{n_2})\ln(n_{n_1}+n_{n_2}) \\ &= (k-1)C \end{aligned} \tag{32}$$

where $k = |A^{-1}(\{n_1\})| = |A^{-1}(\{n_2\})|$ and

$$C = (n_{n_1}+n_{n_2})\ln(n_{n_1}+n_{n_2}) - n_{n_1}\ln n_{n_1} - n_{n_2}\ln n_{n_2} \geq 0.$$

Also,

$$Obj(\mathcal{N}) - Obj(\mathcal{N}') = \sum_{e\in E} n_e \ln n_e - \sum_{e'\in E'} n_{e'} \ln n_{e'} + (k-1)C.$$

Now, any edge $e \in E$ can be identified uniquely by specifying its hooks, $F(e)$ and $S(e)$, and the edge $m(e)$ it corresponds to in E'; so, for any hooks $h^*, h^\dagger \in H$ and edge $e' \in E'$, let us introduce the temporary notation $n_{[e'h^*h^\dagger]}$, defined by $n_{[e'h^*h^\dagger]} = n_e$, where e is the unique edge such that $m(e) = e'$, $F(e) = h^*$ and $S(e) = h^\dagger$, or $n_{[e'h^*h^\dagger]} = 0$ if there is no such edge e. This notation allows us to write

$$\begin{aligned} Obj(\mathcal{N}) - Obj(\mathcal{N}') &= \sum_{\substack{e'\in E'\\ h^*,h^\dagger\in H}} n_{[e'h^*h^\dagger]} \ln n_{[e'h^*h^\dagger]} - \sum_{e'\in E'} n_{e'} \ln n_{e'} + (k-1)C \\ &= \sum_{\substack{e'\in E'\\ h^*,h^\dagger\in H}} n_{[e'h^*h^\dagger]} \ln \frac{n_{[e'h^*h^\dagger]}}{n_{e'}} + (k-1)C \end{aligned} \tag{33}$$

since $\sum_{h^*,h^\dagger\in H} n_{[e'h^*h^\dagger]} = n_{e'}$. If we also introduce the notation $n_{[e'\bullet h^\dagger]} = \sum_{h^*\in H} n_{[e'h^*h^\dagger]}$ and $n_{[e'h^*\bullet]} = \sum_{h^\dagger\in H} n_{[e'h^*h^\dagger]}$, we can express (33) as

$$\begin{aligned} Obj(\mathcal{N}) - Obj(\mathcal{N}') &= \sum_{\substack{e'\in E'\\ h^*,h^\dagger\in H}} n_{[e'h^*h^\dagger]} \ln \frac{n_{[e'h^*h^\dagger]}}{n_{[e'\bullet h^\dagger]}} + \sum_{\substack{e'\in E'\\ h^*,h^\dagger\in H}} n_{[e'h^*h^\dagger]} \ln \frac{n_{[e'h^*h^\dagger]}}{n_{[e'h^*\bullet]}} \\ &\quad - \sum_{\substack{e'\in E'\\ h^*,h^\dagger\in H}} n_{[e'h^*h^\dagger]} \ln \frac{n_{[e'h^*h^\dagger]}n_{e'}}{n_{[e'\bullet h^\dagger]}n_{[e'h^*\bullet]}} + (k-1)C \\ &\leq \sum_{\substack{e'\in E'\\ h^*,h^\dagger\in H}} n_{[e'h^*h^\dagger]} \ln \frac{n_{[e'h^*h^\dagger]}}{n_{[e'\bullet h^\dagger]}} + \sum_{\substack{e'\in E'\\ h^*,h^\dagger\in H}} n_{[e'h^*h^\dagger]} \ln \frac{n_{[e'h^*h^\dagger]}}{n_{[e'h^*\bullet]}} + (k-1)C \end{aligned} \tag{34}$$

using proposition 8 from section 6.3. These two sums over $e', h^*, h^\dagger$ can be combined into a simpler expression if we express them in terms of edge-ends. Define two equivalence relations, $\sim$ and $\equiv$, on edge-ends of $\mathcal{N}$ by

$$\begin{aligned}(e_1,h_1) \sim (e_2,h_2) \;\Leftrightarrow\; & M(e_1) = M(e_2) \;\wedge\; \big([F(e_1) = h_1 \wedge F(e_2) = h_2 \wedge S(e_1) = S(e_2)] \;\vee \\ & \qquad [S(e_1) = h_1 \wedge S(e_2) = h_2 \wedge F(e_1) = F(e_2)]\big) \\ (e_1,h_1) \equiv (e_2,h_2) \;\Leftrightarrow\; & m(h_1) = m(h_2) \;\wedge\; (e_1,h_1) \sim (e_2,h_2).\end{aligned}$$

(Note that $(e_1,h_1) \sim (e_2,h_2)$ is a sufficient condition for (e_1,h_1) and (e_2,h_2) to be merged by any merger that merges h_1 and h_2; $(e_1,h_1) \equiv (e_2,h_2)$ is sufficient for (e_1,h_1) and (e_2,h_2) to be merged by m.) Then, for any edge-end $i = (e,h^*)$, where $F(e) = h^*$, $S(e) = h^\dagger$ and $m(e) = e'$, the term $n_{[e'h^*h^\dagger]}$ is simply n_i and $n_{[e'\bullet h^\dagger]}$ is the sum of n_j over all edge-ends j such that $j \equiv i$. Similarly if $i = (e,h^\dagger)$, where $F(e) = h^*$, $S(e) = h^\dagger$ and $m(e) = e'$, then $n_{[e'h^*h^\dagger]}$ is n_i and $n_{[e'h^*\bullet]}$ is the sum of n_j over all j such that $j \equiv i$. Hence (34) simplifies to

$$Obj(\mathcal{N}) - Obj(\mathcal{N}') \le \sum_{i \in EE} n_i \ln \frac{n_i}{\sum_{j \equiv i} n_j} + (k-1)C.$$

The summation here is over EE, the set of all edge-ends of $\mathcal{N}$. However, the summand is non-zero only when i is incident to n_1 or n_2, so we may restrict the summation to these i, giving

$$\begin{aligned}Obj(\mathcal{N}) - Obj(\mathcal{N}') &\le \sum_{h_1 \in A^{-1}(\{n_1\})} \sum_{(i_1,i_2) \in R_{h_1,b(h_1)}} \Big(n_{i_1} \ln \frac{n_{i_1}}{n_{i_1}+n_{i_2}} + n_{i_2} \ln \frac{n_{i_2}}{n_{i_1}+n_{i_2}}\Big) + (k-1)C \\ &= (k-1)C - \sum_{h_1 \in A^{-1}(\{n_1\})} \sum_{(i_1,i_2) \in R_{h_1,b(h_1)}} I_{i_1,i_2} \qquad (35)\end{aligned}$$

where

$$\begin{aligned}\forall h_1,h_2 \in H \quad R_{h_1,h_2} &= \{\, (i_1,i_2) \mid \pi(i_1) = h_1 \wedge \pi(i_2) = h_2 \wedge i_1 \sim i_2 \,\} \\ \forall i_1,i_2 \in EE \quad I_{i_1,i_2} &= -n_{i_1} \ln \frac{n_{i_1}}{n_{i_1}+n_{i_2}} - n_{i_2} \ln \frac{n_{i_2}}{n_{i_1}+n_{i_2}}.\end{aligned}$$

From this we can obtain an estimate (in fact, an upper bound) on the immiscibility of n_1 and n_2. By (35) and (32),

$$\text{immiscibility} = \frac{Obj(\mathcal{N}) - Obj(\mathcal{N}')}{Cost(\mathcal{N}) - Cost(\mathcal{N}')} \le I_{n_1,n_2,b}$$

where

$$\begin{aligned}I_{n_1,n_2,b} &= \frac{(k-1)C - \sum_{h_1 \in A^{-1}(\{n_1\})} \sum_{(i_1,i_2) \in R_{h_1,b(h_1)}} I_{i_1,i_2}}{(k-1)C} \\ &= \frac{1}{k-1}\Big[\sum_{h_1 \in A^{-1}(\{n_1\})} \Big(1 - \frac{\sum_{(i_1,i_2) \in R_{h_1,b(h_1)}} I_{i_1,i_2}}{C}\Big) \;-\; 1\Big] \\ &= \frac{1}{k-1}\Big[\sum_{h_1 \in A^{-1}(\{n_1\})} I_{h_1,b(h_1)} \;-\; 1\Big]\end{aligned}$$

where the non-negative quantity I_{h_1,h_2} is defined as $1 - \frac{1}{C}\sum_{(i_1,i_2)\in R_{h_1,h_2}} I_{i_1,i_2}$.

We now have a firm theoretical basis for searching for the least-immiscible way of merging two nodes. We have a cautious estimate $I_{n_1,n_2,b}$ of the harm done in merging n_1 and n_2 using the bijection b: it is an upper bound on the true immiscibility, so if $I_{n_1,n_2,b}$ is low we can be sure that the true immiscibility is low. We have also analysed $I_{n_1,n_2,b}$ into contributions from each pair of merged hooks, I_{h_1,h_2}, where $h_2 = b(h_1)$, and even into contributions from pairs of merged edge-ends, I_{i_1,i_2}, so we can seek the lowest possible $I_{n_1,n_2,b}$ by trying to make the contributions I_{h_1,h_2} as small as possible.

The algorithm for doing this is called Choose-Merge, and is as follows.

(1) For each node $n_1 \in N$, do the following step.

(1.1) Identify the nodes $n_2 \in N$ that are potentially mergeable with n_1: these are the ones for which $|A^{-1}(\{n_1\})| = |A^{-1}(\{n_2\})|$ and there exist an edge-end i_1 incident to n_1 and an edge-end i_2 incident to n_2 such that $i_1 \sim i_2$. For each such node n_2, do the following steps.

(1.1.1) Construct a bijection $b{:}A^{-1}(\{n_1\}) \to A^{-1}(\{n_2\})$ by taking each hook $h_1 \in A^{-1}(\{n_1\})$ in turn and doing the following step.

(1.1.1.1) For each hook $h_2 \in A^{-1}(\{n_2\})$, if $h_2 \notin b(A^{-1}(\{n_1\}))$ and there is no edge between h_1 and h_2, then do the following steps.

(1.1.1.1.1) Compute

$$I_{h_1,h_2} = 1 - \frac{\sum_{(i_1,i_2)\in R_{h_1,h_2}} I_{i_1,i_2}}{C}$$

where

$$I_{i_1,i_2} = -n_{i_1} \ln \frac{n_{i_1}}{n_{i_1}+n_{i_2}} - n_{i_2} \ln \frac{n_{i_2}}{n_{i_1}+n_{i_2}}$$
$$C = (n_{n_1}+n_{n_2})\ln(n_{n_1}+n_{n_2}) - n_{n_1}\ln n_{n_1} - n_{n_2}\ln n_{n_2}$$

(1.1.1.1.2) If this value of I_{h_1,h_2} is the lowest one found so far (for the present h_1), record it at h_1 (call it I_{h_1}), and modify the function b so that it maps h_1 to h_2.

(1.1.2) Calculate

$$I_{n_1,n_2,b} = \frac{1}{|A^{-1}(\{n_1\})|-1}\Big[\sum_{h_1\in A^{-1}(\{n_1\})} I_{h_1} \; - \; 1\Big]$$

using the I_{h_1} values recorded at each hook h_1.

(1.1.3) If this value of $I_{n_1,n_2,b}$ is the lowest one found so far (for the present n_1) record it at n_1 (call it I_{n_1}), and also record the current n_2 and b.

(2) Select the node $n_1 \in N$ with the lowest recorded value of I_{n_1}. The output of the algorithm is n_1, the corresponding node n_2, the constructed bijection $b{:}A^{-1}(\{n_1\}) \to A^{-1}(\{n_2\})$, and the immiscibility $I_{n_1} = I_{n_1,n_2,b}$.

8. The entire learning algorithm

This section puts together the pieces from previous sections to give the complete learning algorithm. The whole algorithm is implemented in the connectionist programming language defined in Fletcher (2000), although I am presenting it here in a higher-level and less formal way, for ease of understanding.

At the outset we have:

- the image grid;
- a population of patterns, arriving one at a time on the image grid from the environment;
- an initial grammar.

All these are described in sections 3 and 5. There are two positive real parameters, for which I use values $\mu_0 = 0.05$, $\sigma_0 = 0.07$. The learning algorithm is as follows.

```
(1) Let $\mathcal{N}$ be the initial grammar. For every node $n$ and edge $e$ in $\mathcal{N}$, set $n_n := 0$ and
    $n_e := 0$.

(2) Repeat the following sequence of steps until no further splits or merges take
    place.
    (2.1) Apply the Choose-Split algorithm (section 6).
    (2.2) Apply the Choose-Merge algorithm (section 7).
    (2.3) Remove any edge $e$ with $n_e = 0$ and any node $n$ with $n_n = 0$.
    (2.4) If the minimum immiscibility found by Choose-Merge is below $\mu_0$ then
              merge the nodes $n_1$ and $n_2$, using the bijection $b$, as given by Choose-
              Merge
          else
              if the maximum fissility found by Choose-Split exceeds $\sigma_0$ then split the
              portion identified by Choose-Split.
```

9. Examples

In this section I shall illustrate the behaviour of the learning algorithm using five pattern populations. The aim of the algorithm is to find a grammar network $\mathcal{N}$ that generates a grid language $GL(\mathcal{N})$ equal to the pattern population. The image grid is as described in section 3 (see figure 4) and is 20 nodes in height and 40 nodes in width (except in the last example where a slightly larger grid is used); a pattern is drawn in the grid by activating a subset of the nodes, hooks and edges of the grid.

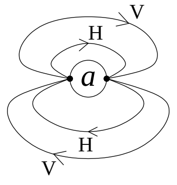
At the end of the section I shall provide a table of the size of the search space, as an indication of the computational difficulty of the task.

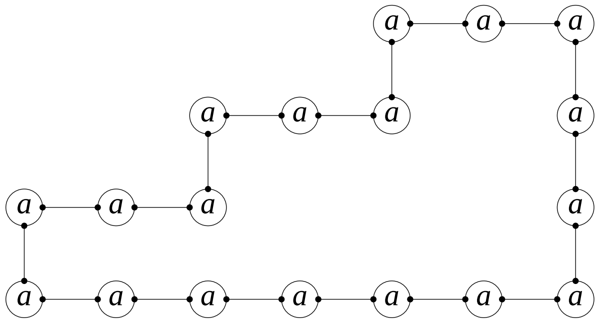
9.1 Staircases

The first example uses only nodes with two hooks. The patterns are staircases, consisting of n stairs, a horizontal base, and a vertical strut; n ranges between 1 and 19 (19 being the maximum size that can fit on the grid), with all the values of n equally probable. A typical pattern, with $n = 3$, is shown on the right-hand side of figure 11. The staircase can occur at any position on the grid. The initial grammar is shown at step 1 in figure 11: it consists of a single node with two hooks, and edges connecting the hooks in all possible ways. By a sequence of four splits, each involving only one node, the initial grammar is refined into a grammar $\mathcal{N}$ that represents the pattern population. No further splits occur. Figure 11 shows the grammar networks at each stage, following the split and removal of unused edges: i.e. it shows the network as it is after step 2.3 of the learning algorithm (section 8). Because the algorithm goes directly to a solution, no merges are necessary. The right-hand side of the figure shows how a typical pattern is parsed at each stage: each grammar node is marked with a unique letter and each pattern node is marked with the letter of the grammar node it maps to. Edge labels and arrows in the pattern are omitted from the figure, for the sake of clarity, but they are in accordance with figure 4: i.e. horizontal edges have arrows pointing rightwards and vertical edges have arrows pointing upwards. (The same display conventions will be used for all the examples.) It can be seen from figure 11 how the learning process works by making successively finer grammatical distinctions. The language generated by the final grammar, $L(\mathcal{N})$, is $c^*a^*(dbe)^*$ (using regular expression notation, traversing the pattern anti-clockwise), which is a superset of the pattern population; but the requirement that the pattern be drawable in the grid restricts the language to $GL(\mathcal{N}) = \{\, c^{2n+1}a^{n-1}(dbe)^n \mid 1 \leq n \leq 19\,\}$, which is exactly the pattern population. It would be impossible for $L(\mathcal{N})$ to equal the pattern population as this would require a context-sensitive grammar.

Notice that if, after learning had finished, the image grid were enlarged, then the grid language would change to $\{\, c^{2n+1}a^{n-1}(dbe)^n \mid 1 \leq n \leq N\,\}$ for some N; thus the algorithm has been trained exclusively on staircases of up to 19 steps, but it is capable of recognising staircases of any size. A similar comment applies to all the pattern populations that follow.

Step 1: the initial grammar.





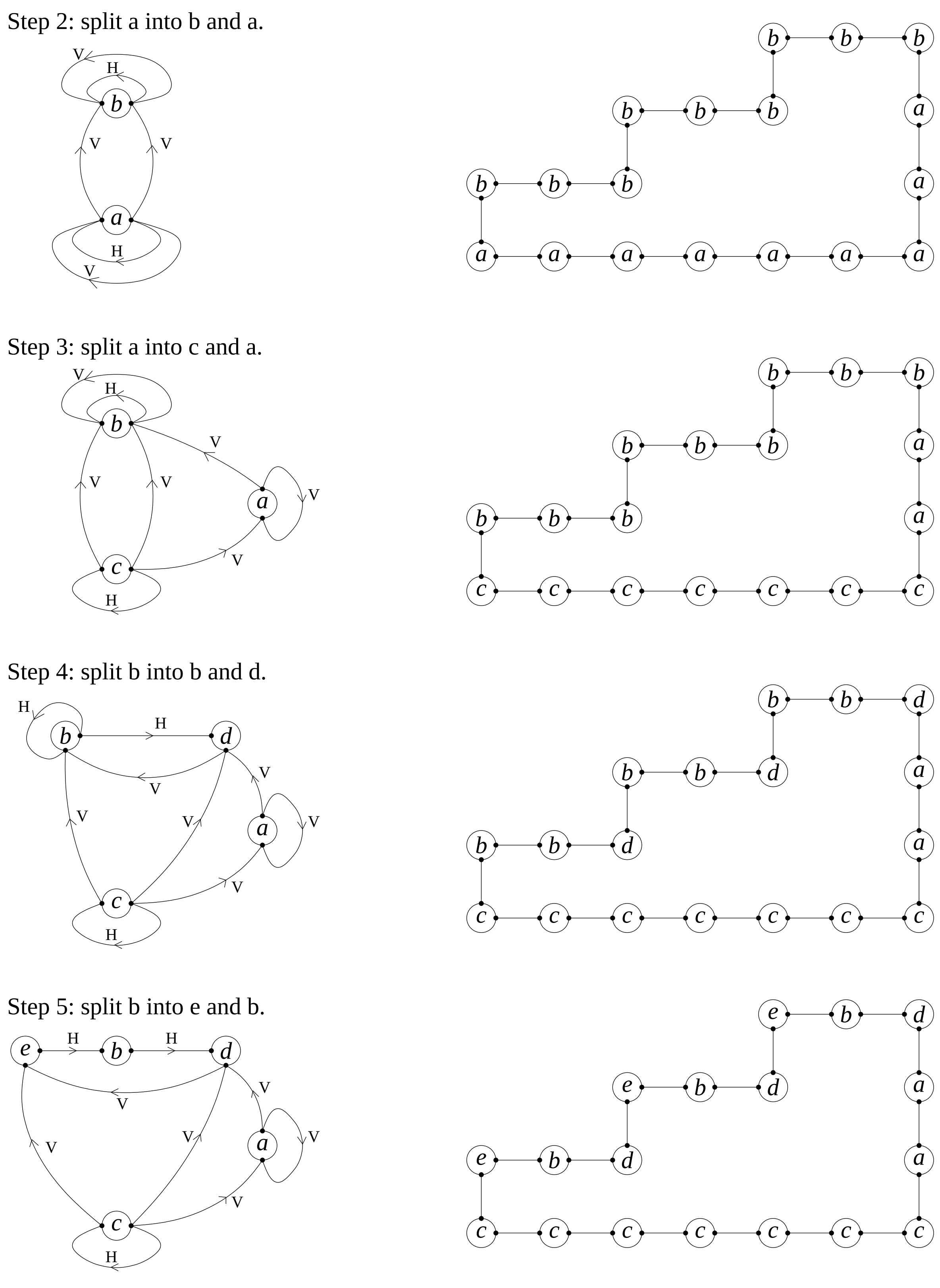
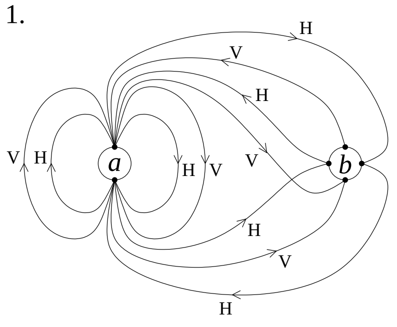

Step 2: split a into b and a.
Step 3: split a into c and a.
Step 4: split b into b and d.
Step 5: split b into e and b.

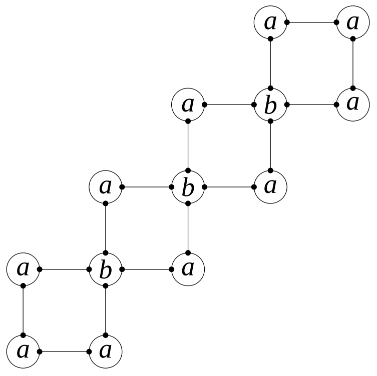
Figure 11. The ‘staircase’ pattern population. On the left are shown the grammar networks at each step of learning, and on the right is shown how a typical pattern is parsed.

9.2 Trellis patterns

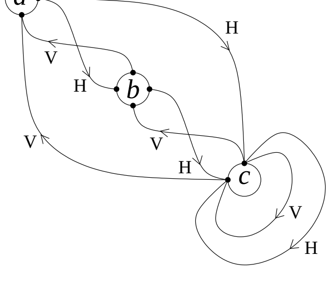
The second example involves patterns that are less string-like. A trellis pattern consists of a sequence of n squares connected together at their top-right and bottom-left corners, where n is distributed as $\min(1+Exp(9), 19)$. ($Exp(m)$ denotes a discrete exponential probability distribution with mean m.) Figure 12 shows a typical pattern, with $n = 4$. The pattern may occur at any position on the grid. The algorithm proceeds directly in three single-node splits to a solution (step 4 in the figure) in which $GL(\mathcal{N})$ equals the pattern population. However, in this case it does not stop there but splits a portion consisting of nodes a, b and c and the edges between them, to give the network shown in step 5. This last split does not alter $GL(\mathcal{N})$, but it does reduce $L(\mathcal{N})$ to bring it closer to the pattern population. Further splits of the same kind follow, reducing $L(\mathcal{N})$ further without changing $GL(\mathcal{N})$. These splits following step 4 involve learning grammatical constraints that are already enforced by the structure of the image grid, and hence these splits may be considered unnecessary. If the aim is merely to make $GL(\mathcal{N})$ equal the pattern population then it would be best to stop the learning at step 4; whereas if the aim is to make $L(\mathcal{N})$ approximate the pattern population as closely as possible then the learning should be continued forever. As in the staircase example, $L(\mathcal{N})$ will never equal the pattern population exactly, since this would require a context-sensitive grammar.

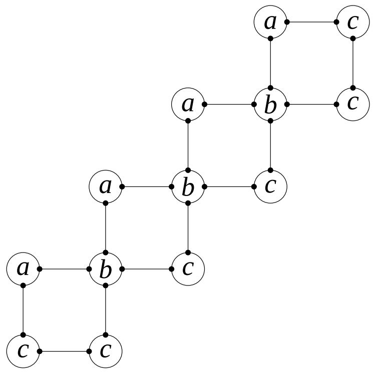
Step 1.





Step 2: split a into a and c.





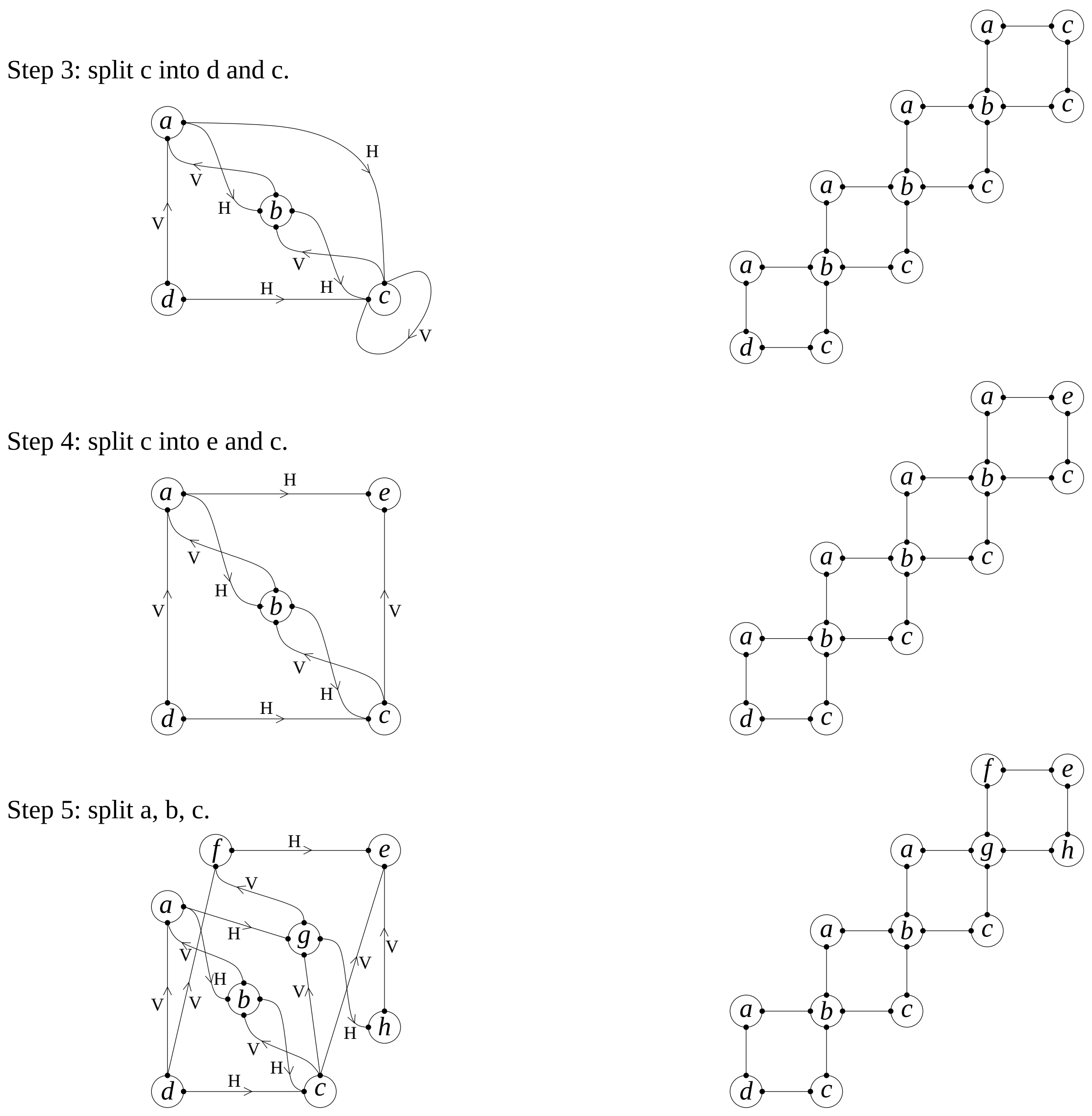


Figure 12. The ‘trellis’ pattern population. As in figure 11, the grammar network at each stage is shown on the left and a typical pattern is shown on the right.

9.3 Stalagmites

The third example involves non-Eulerian graphs and nested iteration. A pattern consists of a sequence of n vertical rectangles (‘stalagmites’) on a horizontal base, where n is distributed as $\min(1+Exp(9),20)$; the pattern may occur at any position on the grid. The stalagmites have variable height, distributed as $\min(1+Exp(4), headroom)$, where *headroom* is the space between the base and the top of the grid. Figure 13 shows an example pattern with four stalagmites. The algorithm proceeds directly in four single-node splits to the network shown in the figure, for which $GL(\mathcal{N})$ equals the pattern population. As in the previous case, further splits follow, which reduce $L(\mathcal{N})$ without affecting $GL(\mathcal{N})$.

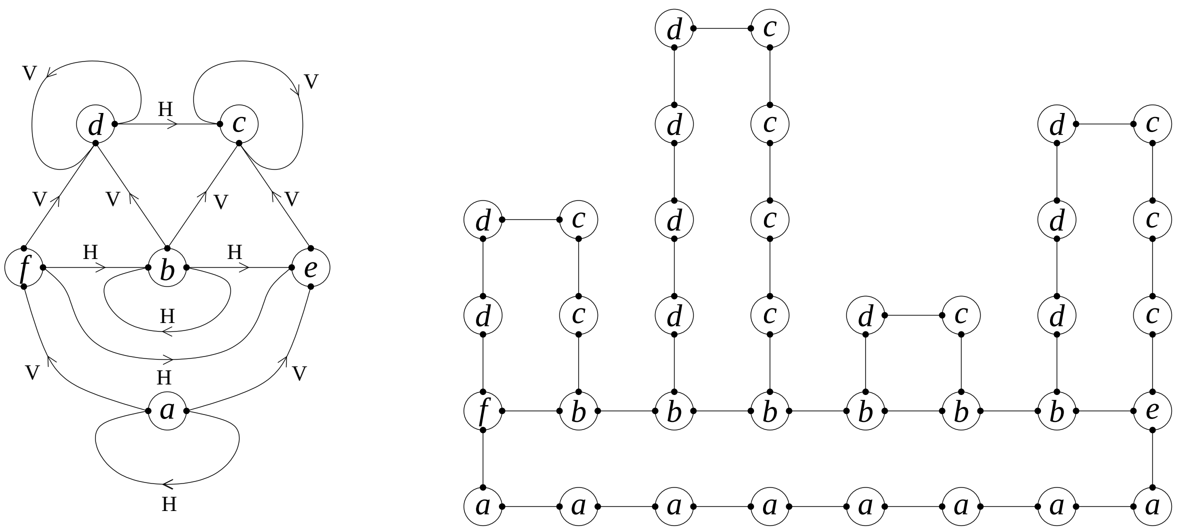


Figure 13. A ‘stalagmites’ pattern and the associated grammar network.

9.4 Tessellations

The next example involves an iteration in two dimensions and requires a much larger grammar. A pattern is a tessellation of squares and crosses, with a rectangular border. The number of squares in each row, m, is distributed as $\min(1+Exp(4), 6)$ and the number of squares in each column, n, is distributed as $\min(1+Exp(3), 6)$; figure 14 shows an example with $m = 3$ and $n = 2$. The tessellation may occur at any position in the grid.

This time the learning process is less straightforward. Figure 14 shows the grammar network after 6 single-node splits, 2 two-node splits, 5 three-node splits, and 6 mergers. A couple of the single-node splits involve a poor choice of a_i values, which are corrected by later splits and merges (in accordance with the theoretical argument in section 5). The resulting network is shown in the figure; $GL(\mathcal{N})$ equals the pattern population. If the learning process is continued then further splits occur, which do not alter $GL(\mathcal{N})$, just as in the previous two examples.

9.5 Carpets

The final example requires an even larger grammar than the previous one; the size of the image grid has also been increased to 25×40. A ‘carpet’ pattern is a rectangle with zigzags around the border. There are m zigzags horizontally and n vertically, where m is distributed as $\min(2+Exp(12), 17)$ and n is distributed as $\min(2+Exp(8), 10)$; figure 15 shows the case where $m = 5$ and $n = 3$. Note that there are also small squares at the corners. The pattern may occur at any position in the grid.

The top half of figure 15 shows the 76-node grammar learned after 34 single-node splits, 17 two-node splits, 4 three-node splits, 1 four-node split, and 11 mergers.

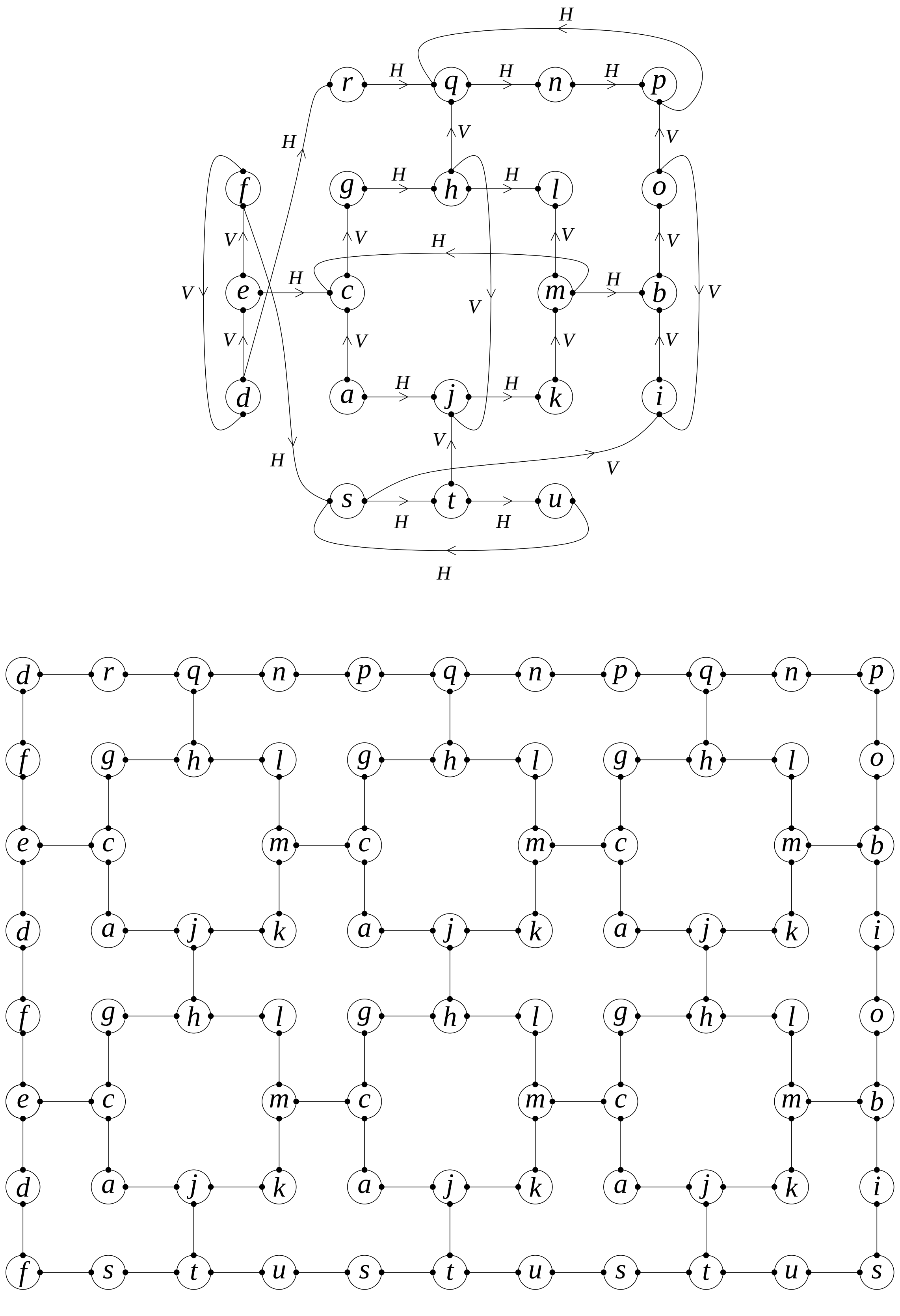


Figure 14. A ‘tessellation’ pattern and the associated grammar network.

For this grammar, $GL(\mathcal{N})$ equals the pattern population. However, note that there is some redundancy in the top and bottom rows: nodes d, n, o, z, y could be merged with f, p, q, B, A and also with g, r, s, D, C; also, nodes $5, 4, \alpha, \beta, \pi$ could be merged with $7, 6, \gamma, \delta, \sigma$. The merging procedure overlooks these mergers because it only considers merging nodes two at a time.

As in the previous examples, further splits occur if the learning process is continued, introducing further redundancies without altering $GL(\mathcal{N})$.

9.6 The size of the search space

The computational difficulty of the learning task can be gauged roughly from the size of the search space, i.e. the number of possible (non-isomorphic) simple grammar networks with a given number of nodes. This number depends on the numbers of hooks that the nodes are permitted to have: table 2 shows the case where the nodes are allowed to have two or three hooks, as in the stalagmites and tessellation examples, and the case where the nodes are allowed to have two or four hooks, as in the trellis example. The table assumes that there are two edge labels, as in all the examples. The exact values for $n = 1$ are obtained by direct enumeration; the upper and lower bounds for $n > 1$ are calculated by an elementary counting argument.

nodes with 2 or 3 hooks		nodes with 2 or 4 hooks	
n	no. of networks with n nodes	n	no. of networks with n nodes
1	730	1	703,770
2	1.6×10^{16}–1.2×10^{18}	2	4.5×10^{30}–5.2×10^{33}
3	1.7×10^{40}–2.2×10^{43}	3	3.6×10^{74}–3.0×10^{79}
4	9.5×10^{74}–3.0×10^{79}	4	3.9×10^{137}–3.1×10^{144}
5	2.9×10^{120}–2.7×10^{126}	5	6.3×10^{219}–6.1×10^{228}

Table 2. The number of non-isomorphic simple networks with up to five nodes.

10. Conclusions

The algorithm presented in this paper enables a connectionist network to configure itself to represent a regular graph grammar. It learns from positive examples only, and it is able to deal with non-Eulerian graphs without reducing them to a string-like form. This enables it to cope with a variety of iterative structures in two dimensions. As far as I am aware, this is the only learning algorithm known that can deal with graph languages of this type. Moreover, the parallel approach to parsing described in section 3.1 seems to have the potential to locate and correct errors better than

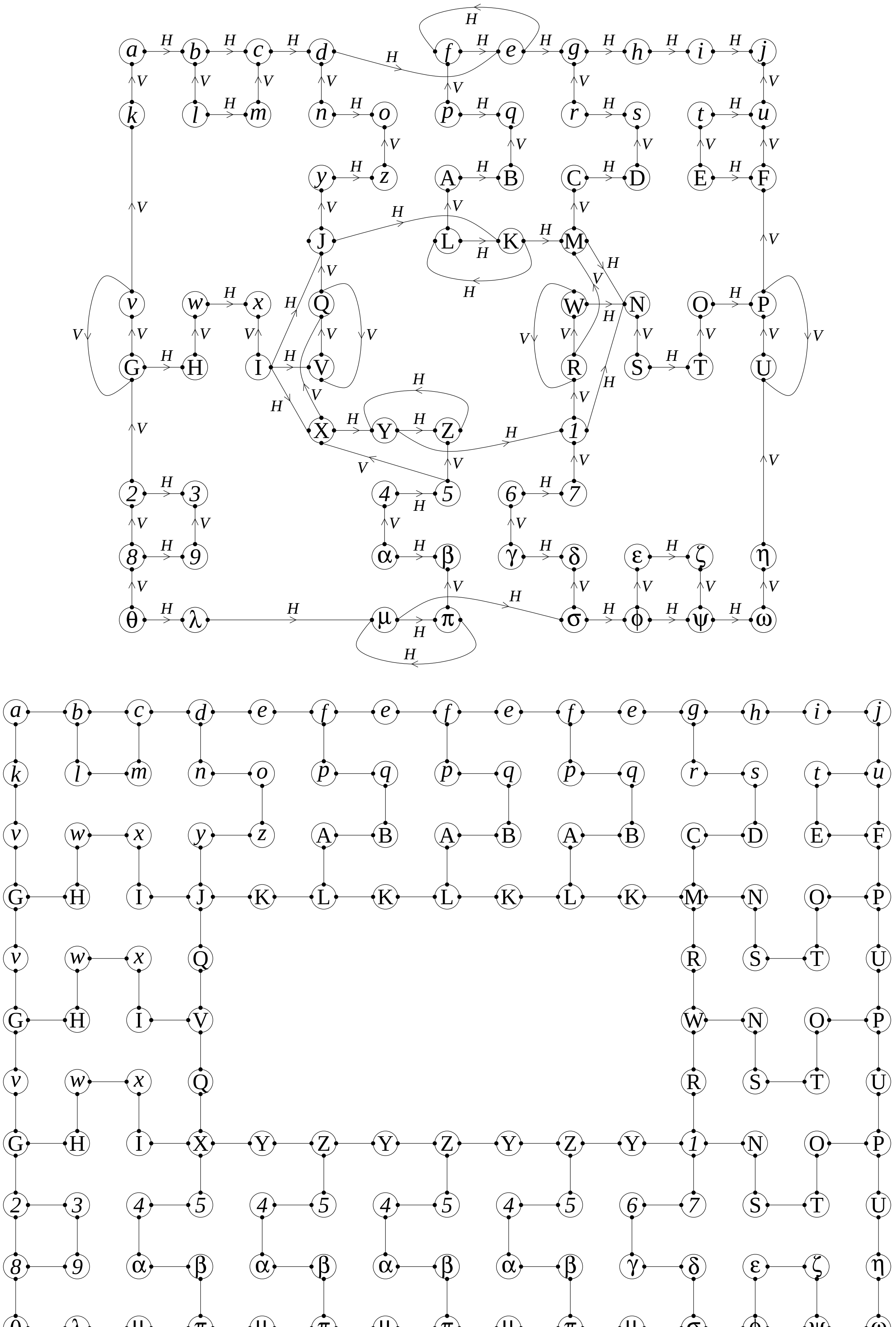

a b c d e f e f e f e g h i j
k l m n o p q p q p q r s t u
v w x y z A B A B A B C D E F
G H I J K L K L K L K M N O P
v w x Q R S T U
G H I V W N O P
v w x Q R S T U
G H I X Y Z Y Z Y Z Y 1 N O P
2 3 4 5 4 5 4 5 4 5 6 7 S T U
8 9 α β α β α β α β γ δ ε ζ η
θ λ μ π μ π μ π μ π μ σ φ ψ ω

Figure 15. A 'carpet' pattern and the associated grammar network.

sequential parsing methods, though the present version of the algorithm makes no attempt to do so; this is work for the future.

Some technical limitations should be noted. In the first place, the main learning step consists of splitting a portion of the network into two; there are occasions when a three-fold split is desirable, which could in principle be accomplished by two two-fold splits, but the algorithm will not necessarily realise this. Secondly, the merging procedure is limited at present to merging of pairs of nodes. This means there are some desirable simplifications of the grammar that it will not be able to make (see section 9.5). Thirdly, the algorithm lacks a proper halting condition. If the aim is to make the grid language $GL(\mathcal{N})$ generated by the grammar equal to the pattern set then the algorithm should halt as soon as this is achieved. It fails to do so because it is really driven by differences between the language $L(\mathcal{N})$ generated by the grammar and the pattern set (see the examples in section 9). This could be rectified by using 'waking' and 'dreaming' phases, as in Fletcher (1991); at present, the algorithm only has a waking phase.

Hence the algorithm in its present form cannot be regarded as a complete solution to the problem of inferring regular graph grammars. However, to dwell on these limitations would be to lose sight of the main purpose of the exercise, which was to open up new possibilities in connectionist symbol processing. Symbol processing has traditionally been implemented on sequential computers, and this has in some respects had a distorting effect on the associated theory. For example, formal language theory has been heavily influenced by automaton theory and hence parsing has been seen as a process of traversal or scanning (Ginsborg 1966: Ch. 2); I have argued in section 3.1 that traversal is inessential to parsing and obscures its real nature as the construction of a homomorphism between the pattern and the grammar. Moreover, the traditional artificial-intelligence approach to problem solving is to see it as a *search* through a space of possibilities, guided by heuristics and back-tracking. My algorithm works in a different, distinctively connectionist way: the global task is divided into local tasks for the individual nodes and edges, and provided the nodes and edges perform their local tasks correctly the network as a whole proceeds more-or-less directly to the solution. Admittedly, there is an element of (parallel) search in the way in which the t_e and g_h values are set in the Choose-Split procedure (see section 6); however, the method by which the a_i values are determined (step 2.9 of Choose-Split) uses continuous interpolation between many possible configurations instead of enumeration of configurations. The important point, though, is that the algorithm *does no back-tracking*: if it makes a mistake in splitting then it corrects it by further splitting and merging. Viewing the search space as a category of networks and homomorphisms (see sections 3.3 and 5) allows the algorithm to find its way to a solution with a minimum of searching, despite the very large size of the space (see section 9.6).

A third way in which connectionism provides a new perspective on symbol processing is by imposing computational requirements that other algorithms do not attempt to meet. To appreciate this point it is helpful to classify the non-connectionist grammatical inference algorithms cited in section 2 into *piecemeal* and *batch* methods. Piecemeal methods receive one pattern at a time and adjust the grammar in response to it. Consequently they may be thrown off course by a single noisy or 'stray' pattern possessing idiosyncratic grammatical features; they are also vulnerable to effects of pattern presentation order. Batch methods learn from the whole pattern set, and consequently require the pattern set, or a summarised form of it such as a prefix-tree acceptor, to be stored. Both approaches are undesirable from a connectionist perspective, which stresses robustness and limited storage capacity. My algorithm avoids both drawbacks: it requires only the current pattern to be stored and it adjusts the grammar only in response to observed statistical regularities in the pattern set, not to individual patterns.

The algorithm has aspects that are normally considered characteristic of 'symbolic computation' and aspects normally considered characteristic of 'subsymbolic computation'. The 'symbolic' aspects are as follows.

- The algorithm can handle nested iterative structure.
- It has a clear global semantics: that is, the grammatical knowledge embodied in the network, and the gain in knowledge achieved by a refinement step, can be stated explicitly in terms of the theory of networks and homomorphisms in section 3 and the objective and cost functions in section 6.
- It has a clear local semantics: every item of information stored in the network has a meaning.
- The local semantics is provably related to the global semantics, via the theory of fissility in section 6 and the theory of immiscibility in section 7.

The 'subsymbolic' aspects are as follows.

- Numerical parameters (the t_e, g_h and a_i values), distributed across a portion of the network, are used as fine-grained constituents of tentative grammatical hypotheses (see section 6).
- Locally-visible correlations are used to drive the learning process (see equation (4) and the surrounding discussion in section 5). Local correlations are also the basis of Hebb's rule, and hence are often seen as characteristic of subsymbolic learning.

Nevertheless, the algorithm was not designed as a deliberate hybrid of symbolic and subsymbolic processing but simply as the most appropriate one for the job. Perhaps I can best sum up the message of this paper as follows: artificial-intelligence problems involving the learning of structured concepts can benefit from a connectionist perspective; and connectionism can benefit from ignoring the symbolic/subsymbolic dichotomy.

References

Angluin, D., 1978, On the complexity of minimum inference of regular sets. *Information and Control*, **39**: 337–350.

Angluin, D., 1987, Learning regular sets from queries and counterexamples. *Information and Computation*, **75** (2): 87–106.

Angluin, D., 1988, Queries and concept learning. *Machine Learning*, **2**: 319–342.

Angluin, D., 1992, Computational learning theory: survey and selected bibliography. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing* (New York: ACM Press), pp. 351–369.

Angluin, D., and Kharitonov, M., 1991, When won't membership queries help? In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (New York: ACM Press), pp. 444–454.

Barnden, J.A., and Pollack, J.B., 1991, Problems for high-level connectionism. In *High-Level Connectionist Models*, edited by Barnden, J.A., and Pollack, J.B., Advances in Connectionist and Neural Computation Theory, vol. 1 (Norwood, New Jersey: Ablex), pp. 1–16.

Bartsch-Spörl, B., 1983, Grammatical inference of graph grammars for syntactic pattern recognition. *Lecture Notes in Computer Science*, **153**: 1–7.

Bauderon, M., 1996, A category-theoretical approach to vertex replacement: the generation of infinite graphs. *Lecture Notes in Computer Science*, **1073**: 27–37.

Bell, T.C., Cleary, J.G., and Witten, I.H., 1990, *Text Compression* (Englewood Cliffs, New Jersey: Prentice-Hall).

Bhattacharyya, P., and Nagaraja, G., 1993, Learning a class of regular languages in the probably approximately correct learnability framework of Valiant. In *Grammatical Inference: Theory, Applications and Alternatives*, First Colloquium on Grammatical Inference, Essex, U.K., 22nd–23rd April 1993, IEE Digest no. 1993/092 (London: IEE).

Brandenburg, F.J., and Chytel, M.P., 1991, Cycle chain code picture languages. *Lecture Notes in Computer Science*, **532**: 157–173.

Brandenburg, F.J., and Skodinis, K., 1996, Graph automata for linear graph languages. *Lecture Notes in Computer Science*, **1073**: 336–350.

Browne, A., 1998, Performing a symbolic inference step on distributed representations. *Neurocomputing*, **19** (1–3): 23–34.

Bunke, H., and Haller, B., 1992, Syntactic analysis of context-free plex languages for pattern recognition. In *Structured Document Image Analysis*, edited by H.S. Baird, H. Bunke, and K. Yamamoto (Berlin: Springer-Verlag), pp. 500–519.

Carrasco, R.C., Oncina, J., and Calera, J., 1998, Stochastic inference of regular tree languages. *Lecture Notes in Artificial Intelligence*, **1433**: 187–198.

Castaño, M.A., Vidal, E., and Casacuberta, F., 1995, Finite-state automata and connectionist machines: a survey. *Lecture Notes in Computer Science*, **930**: 433–440.

Charniak, E., and Santos, E., 1991, A context-free connectionist parser which is not connectionist, but then it is not really context-free either. In *High-Level Connectionist Models*, edited by Barnden, J.A., and Pollack, J.B., Advances in Connectionist and Neural Computation Theory, vol. 1 (Norwood, New Jersey: Ablex), pp. 123–134.

Corbí, A., Oncina, J., and García, P., 1993, Learning regular languages from a complete sample by error correcting techniques. In *Grammatical Inference: Theory, Applications and Alternatives*, First Colloquium on Grammatical Inference, Essex, U.K., 22nd–23rd April 1993, IEE Digest no. 1993/092 (London: IEE).

Cottrell, G.W., 1989, *A Connectionist Approach to Word Sense Disambiguation* (London: Pitman).

Dányi, G., 1993, Regular inference with maximal valid grammar method. In *Grammatical Inference: Theory, Applications and Alternatives*, First Colloquium on Grammatical Inference, Essex, U.K., 22nd–23rd April 1993, IEE Digest no. 1993/092 (London: IEE).

Das, R., Giles, C.L., and Sun, G.Z., 1993, Using prior knowledge in a NNPDA to learn context-free languages. In *Advances in Neural Information Processing Systems 5*, edited by S.J. Hanson, J.D. Cowan, and C.L. Giles (San Mateo, California: Morgan Kaufmann).

de la Higuera, C., 1998, Learning stochastic finite automata from experts. *Lecture Notes in Artificial Intelligence*, **1433**: 79–89.

Dinsmore, J. (ed.), 1992, *The Symbolic and Connectionist Paradigms: Closing the Gap* (Hillsdale, New Jersey: Lawrence Erlbaum Associates).

Drewes, F., and Kreowski, H.-J., 1991, A note on hyperedge replacement. *Lecture Notes in Computer Science*, **532**: 1–11.

Dupont, P., 1996, Incremental regular inference. *Lecture Notes in Artificial Intelligence*, **1147**: 222–237.

Dupont, P., Miclet, L., and Vidal, E., 1994, What is the search space of the regular inference? *Lecture Notes in Artificial Intelligence*, **862**: 25–37.

Ehrig, H., 1979, Introduction to the algebraic theory of graph grammars (a survey). *Lecture Notes in Computer Science*, **73**: 1–69.

Ehrig, H., Korff, M., and Löwe, M., 1991, Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. *Lecture Notes in Computer Science*, **532**: 24–37.

Elman, J.L., 1990, Finding structure in time. *Cognitive Science*, **14** (2): 179–212.

Elman, J.L., 1992, Grammatical structure and distributed representations. In *Connectionism: Theory and Practice*, edited by S. Davis (New York: Oxford University Press), pp. 138–178.

Engelfriet, J., and Rozenberg, G., 1991, Graph grammars based on node rewriting: an introduction to NLC graph grammars. *Lecture Notes in Computer Science*, **532**: 12–23.

Fahlman, S.E., 1991, *The Recurrent Cascade-Correlation Architecture.* CMU-CS-91-100, School of Computer Science, Carnegie Mellon University.

Fanty, M., 1985, *Context-Free Parsing in Connectionist Networks.* TR174, Computer Science Department, University of Rochester.

Flasiński, M., 1993, On the parsing of deterministic graph languages for syntactic pattern recognition. *Pattern Recognition*, **26** (1): 1–16.

Fletcher, P., 1991, A self-configuring network. *Connection Science*, **3** (1): 35–60.

Fletcher, P., 1992, Principles of node growth and node pruning. *Connection Science*, **4** (2): 125–141.

Fletcher, P., 1993, Neural networks for learning grammars. In *Grammatical Inference: Theory, Applications and Alternatives*, First Colloquium on Grammatical Inference, Essex, U.K., 22nd–23rd April 1993, IEE Digest no. 1993/092 (London: IEE).

Fletcher, P., 2000, The foundations of connectionist computation. *Connection Science*, **12** (2): 163–196.

Fodor, J.A., 1975, *The Language of Thought* (Cambridge, Massachusetts: Harvard University Press).

Fodor, J.A., and Pylyshyn, Z.W., 1988, Connectionism and cognitive architecture: a critical analysis. *Cognition*, **28** (1 & 2): 3–71.

Fu, K.-S., and Booth, T.L., 1975, Grammatical inference: introduction and survey, parts I & II. *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-5**: 95–111 & 409–423.

García, P., and Vidal, E., 1990, Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**: 920–925.

Garfield, J.L., 1997, Mentalese not spoken here: computation, cognition and causation. *Philosophical Psychology*, **10** (4): 413–435.

Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z., and Lee, Y.C., 1992, Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, **4**: 393–405.

Giles, C.L., and Omlin, C.W., 1992, Inserting rules into recurrent neural networks. In *Neural Networks for Signal Processing II*, edited by S.Y. Kung, F. Fallside, J.A. Sorenson, and C.A. Kamm, Proceedings of the 1992 IEEE-SP Workshop (New York: IEEE), pp. 13–22.

Ginsborg, S., 1966, *The Mathematical Theory of Context-Free Languages* (New York: McGraw-Hill).

Giordano, J.-Y., 1994, Inference of context-free grammars by enumeration: structural containment as an ordering bias. *Lecture Notes in Artificial Intelligence*, **862**: 212–221.

Gold, E.M., 1967, Language identification in the limit. *Information and Control*, **10**: 447–474.

Gold, E.M., 1978, Complexity of automaton identification from given data. *Information and Control*, **37**: 302–320.

Goldblatt, R., 1984, *Topoi: The Categorial Analysis of Logic* (Amsterdam: North-Holland).

Gregor, J., and Thomason, M.G., 1996, A disagreement count scheme for inference of constrained Markov networks. *Lecture Notes in Artificial Intelligence*, **1147**: 168–178.

Habel, A., 1992, *Hyperedge Replacement: Grammars and Languages* (Berlin: Springer-Verlag).

Hadley, R.F., 1999, Connectionism and novel combinations of skills: implications for cognitive architecture. *Minds and Machines*, **9** (2): 197–221.

Hadley, R.F., and Cardei, V.C., 1999, Language acquisition from sparse input without error feedback. *Neural Networks*, **12** (2): 217–235.

Hadley, R.F., and Hayward, M.B., 1997, Strong semantic systematicity from Hebbian connectionist learning. *Minds and Machines*, **7** (1): 1–37.

Hanson, S.J., and Kegl, J., 1987, PARSNIP: a connectionist network that learns natural language grammar from exposure to natural language sentences. In *Program of the Ninth Annual Conference of the Cognitive Science Society, Seattle, Washington* (Hillsdale, New Jersey: Lawrence Erlbaum Associates), pp. 106–119.

Haselager, W.F.G., and van Rappard, J.F.H., 1998, Connectionism, systematicity, and the frame problem. *Minds and Machines*, **8** (2): 161–179.

Ho, K.S.E., and Chan, L.W., 1997, Confluent preorder parsing of deterministic grammars. *Connection Science*, **9** (3): 269–293.

Ho, K.S.E., and Chan, L.W., 1999, How to design a connectionist holistic parser. *Neural Computation*, **11** (8): 1995–2016.

Horgan, T., and Tienson, J., 1996, *Connectionism and the Philosophy of Psychology* (Cambridge, Massachusetts: MIT Press).

Jeltsch, E., and Kreowski, H.-J., 1991, Grammatical inference based on hyperedge replacement. *Lecture Notes in Computer Science*, **532**: 461–474.

Jordan, M.I., 1988, *Serial Order: A Parallel Distributed Processing Approach.* Report No. 8604, Institute of Cognitive Science, University of California, San Diego.

Kwasny, S.C., and Faisal, K.A., 1990, Connectionism and determinism in a syntactic parser. *Connection Science*, **2** (1 & 2): 63–82.

Lang, K.J., 1992, Random DFA's can be approximately learned from sparse uniform examples. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory, Pittsburgh, Pennsylvania* (New York: Association for Computing Machinery), pp. 45–52.

Lari, K., and Young, S.J., 1990, The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech & Language*, **4** (1): 35–56.

Levy, L.S., and Joshi, A.K., 1978, Skeletal structural descriptions. *Information and Control*, **39**: 192–211.

Lichtblau, U., 1991, Recognizing rooted context-free flowgraph languages in polynomial time. *Lecture Notes in Computer Science*, **532**: 538–548.

Lucas, S.M., 1993, New directions in grammatical inference. In *Grammatical Inference: Theory, Applications and Alternatives*, First Colloquium on Grammatical Inference, Essex, U.K., 22nd–23rd April 1993, IEE Digest no. 1993/092 (London: IEE).

Lucas, S.M., and Damper, R.I., 1990, Syntactic neural networks. *Connection Science*, **2**: 195–221.

Main, M.G., and Rozenberg, G., 1987, Fundamentals of edge-label controlled graph grammars. *Lecture Notes in Computer Science*, **291**: 411–426.

Mäkinen, E., 1994, On the relationship between diagram synthesis and grammatical inference. *International Journal of Computer Mathematics*, **52**: 129–137.

Marcus, G.F., 1998, Rethinking eliminative connectionism. *Cognitive Psychology*, **37** (3): 243–282.

Mitchell, T.M., 1978, *Version Spaces: An Approach to Concept Learning.* PhD thesis, Stanford University.

Nagl, M., 1987, Set theoretic approaches to graph grammars. *Lecture Notes in Computer Science*, **291**: 41–54.

Naumann, S., and Schrepp, J., 1993, Grammatical inference in DACS. In *Grammatical Inference: Theory, Applications and Alternatives*, First Colloquium on Grammatical Inference, Essex, U.K., 22nd–23rd April 1993, IEE Digest no. 1993/092 (London: IEE).

Newell, A., and Simon, H., 1976, Computer science as empirical inquiry: symbols and search. *Communications of the Association for Computing Machinery*, **19** (3): 113–126.

Phillips, S., 1999, Systematic minds, unsystematic models: learning transfer in humans and networks. *Minds and Machines*, **9** (3): 383–398.

Radhakrishnan, V., and Nagaraja, G., 1987, Inference of regular grammars via skeletons. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-17** (6): 982–992.

Reilly, R., 1991, Connectionist technique for on-line parsing. *Network*, **3**: 37–45.

Ron, D., Singer, Y., and Tishby, N., 1994, The power of amnesia. In *Advances in Neural Information Processing Systems 6*, edited by J. Cowan, G. Tesauro, and J. Alspector (San Mateo, California: Morgan Kaufmann), p. 176.

Roques, M., 1994, Dynamic grammatical representation in guided propagation networks. *Lecture Notes in Artificial Intelligence*, **862**: 189–202.

Rozenberg, G. (ed.), 1997, *Handbook of Graph Grammars and Computing by Graph Transformation* (Singapore: World Scientific).

Ruiz, J., and García, P., 1996, Learning k-piecewise testable languages from positive data. *Lecture Notes in Artificial Intelligence*, **1147**: 203–210.

Sakakibara, Y., 1997, Recent advances of grammatical inference. *Theoretical Computer Science*, **185** (1): 15–45.

Selman, B.A., 1985, *Rule-based processing in a connectionist system for natural language understanding.* CSRI-168, Computer Systems Research Institute, University of Toronto.

Smolensky, P., 1988, On the proper treatment of connectionism. *Behavioral and Brain Sciences*, **11** (1): 1–23.

Sougné, J., 1998, Connectionism and the problem of multiple instantiation. *Trends in Cognitive Science*, **2** (5): 183–189.

Stolcke, A., and Omohundro, S., 1994, Inducing probabilistic grammars by Bayesian model merging. *Lecture Notes in Artificial Intelligence*, **862**: 106–118.

Tomita, M., 1982, Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society, Ann Arbor, Michigan* (Boulder, Colorado: The Cognitive Science Society), pp. 105–108.

Tuller A., 1967, *A Modern Introduction to Geometries* (Princeton, New Jersey: D. Van Nostrand Company).

Valiant, L.G., 1984, A theory of the learnable. *Communications of the Association for Computing Machinery*, **27** (11): 1134–1142.

Waltz, D.L., and Pollack, J.B., 1985, Massively parallel parsing: a strongly interactive model of natural-language interpretation. *Cognitive Science*, **9** (1): 51–74.

Williams, R.J., and Zipser, D., 1989, Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, **1** (1): 87–111.

Zeng, Z., Goodman, R.M., and Smyth, P., 1994, Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks*, **5** (2): 320–330.