Genetic Algorithms for Structural Editing

Richard Myers and Edwin R Hancock

Department of Computer Science
University of York
York YO1 5DD, UK

Abstract. This paper describes the use of discrete graphical editing operations to dynamically fit hierarchical structural models to input data. We use the tree adjoining grammar developed by Joshi [1] as a prototypical structural model, and realise the editing process using a genetic algorithm. The novelty of our approach lies firstly in the use of the edit distance between the ordered frontier nodes of a tree and a set of dictionaries of legal labels derived from the input as a cost function. Secondly, we apply genetic algorithms to tree adjoining grammars with the introduction of a new editing operation. We demonstrate the utility of the method on a simple natural language processing problem.

1 Introduction

Graphical models have recently attracted considerable interest in the connectionist literature where they have been used to embed causal relations into network structures. Broadly speaking, the available models can be divided into those that draw on undirected graphs and those that draw on directed graphs. According to this taxonomy, Markov models belong to the former category while causal networks fall into the latter category. In fact directed graph structures are of particular importance since they can be used to represent subsumption or part-whole hierarchies. Such representational structures are of pivotal importance in language understanding and vision where they are used to control the vertical flow of perceptual inference. Despite this interest in the modelling of probabilistic interactions in network structures, the issue of how to control the structure itself has attracted less attention. This is an important omission since in practice the task of extracting hierarchical relations from real-world data is invariably one of extreme fragility. In the machine vision domain, it was Sanfeliu and Fu who first illustrated the importance of graph-edit operations in matching noise corrupted relational structures [2].

It is the editing of directed graph structures to which we turn our attention in this paper. Specifically we focus on the issue of how to extract the most consistent tree-structure from imperfectly formed input. This can be viewed as an optimisation problem. Our goal is to manipulate tree representations of subsumption hierarchies to find the best interpretation of the input. The advantage of such an approach is that although the interpretation process is data-driven, top-down constraint propagation still occurs. The control flow is top-down but the flow of inference is bottom-up. To do this we require a hierarchical representation of the domain of interest and a means of measuring the accuracy with

which a model fits the data. It is clearly inappropriate to enumerate every possible interpretation in the model database. Rather, a natural means of extending some initial small database is required.

The obvious hierarchical representation is a graph - for a subsumption hierarchy, a tree. Models will need to undergo vertical edit operations to adapt their structure to the data. Unfortunately, most graphical editing operations described in the literature are rather naïve, being single-node (or -edge) insertion, deletion or relabelling [2–6] (to be fair, these are generally intended as measures of distance in matching algorithms rather than as true editing operations). The geometric crossover operator described by Cross et al. in [7] is somewhat closer to the mark: it involves bisecting two Delaunay graphs with a random line in order to edit a match relation which exists between them. However, this operation does not necessarily generalise to other types of graph, and is in any case global. What is required is a local structural edit operation which modifies intermediate subgraphs, ranging from single nodes to the entire graph.

The *tree adjoining grammar* formalism developed by Joshi in [1] provides an interesting starting point. Tree adjoining grammars were originally designed for natural language processing: rather than use a set of rewrite rules, these grammars provide sets of minimal parse trees which may be extended by adjunction (section 2). Thus parsing with a tree adjoining grammar can be seen as fitting a hierarchical model to the input and iteratively modifying that model until it adequately describes the data. The manipulation of hierarchical syntactic models is of interest in both vision [8, 9] and language processing; however, our interest here is primarily in the structural editing process. Nevertheless, language processing is the natural application of tree adjoining grammars and provides a convenient non-trivial hierarchical structure recovery problem.

The framework which we present is novel in two respects. First, we use a modified version of the string edit distance algorithm of Wagner and Fischer [10] to measure how well a tree fits the data. This provides the bottom-up, data-driven component. Second, we use a modified genetic algorithm to produce novel but consistent trees using the operators furnished by a tree adjoining grammar. This provides the top-down, constraint propagation component.

2 Parsing with Tree Adjoining Grammars

A tree adjoining grammar, G, is a pair, $(\mathbf{I}, \mathbf{A})$, of sets of ordered labelled trees, where $\mathbf{I}$ is the set of *initial trees*, which correspond to minimal sentences in the string language of G, and $\mathbf{A}$ is the set of *auxiliary trees*, which extend the trees in $\mathbf{I}$ via adjunction to give new initial trees. By definition, all the external nodes of an initial tree are preterminal symbols (lexical categories) and all of its internal nodes are nonterminals. Auxiliary trees are similar to initial trees, except that exactly one of the external nodes has the same label as the root: this is the *foot node*. Tree adjoining grammars are moderately context-sensitive, a property they owe to the manner in which recursion and dependencies are expressed [1].

Adjunction (figure 1) is a composition operation between initial and auxiliary trees: given an auxiliary tree, A, and an initial tree, I, which contains a node with the same label, X, as the root of A, the adjunction of I and A is achieved by removing the subtree in I rooted at X, inserting A in its place, then attaching the children of X to the foot node of A. Thus, adjunction inserts an extra layer of structure into an initial tree. It is usual to place constraints on adjunction at some of the internal nodes of trees. These constraints are discussed in detail in [11], where they are used to implement features via unification. The only constraint of interest to us at present is the null-adjunction constraint which forbids adjunction at any node bearing it.

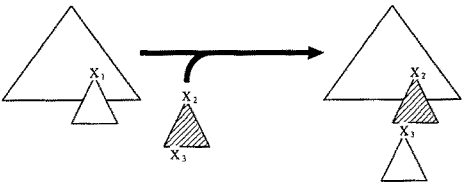
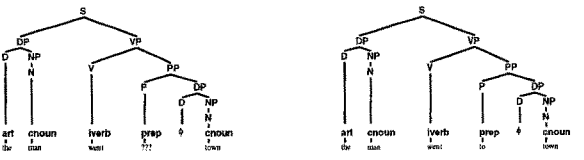


Fig. 1. Tree Adjunction.

2.1 Parsing

Our primary interest is in the manipulation of parse trees; consequently we consider a much simpler problem than the state of the art for language processing. Specifically, we concentrate on the structure of the parse tree and ignore augmentations such as features. The problem can be simply stated as that of finding the "most accurate" parse of an input sentence. We would like to be able to gracefully handle the case where a word in the input is unknown, misspelt or missing as illustrated in figure 2. We would also like ambiguous cases (see figure 3) to be handled neutrally in the absence of *a priori* evidence, i.e. the ordering of rules in the rulebase should not bias the parser in favour of any particular interpretation.



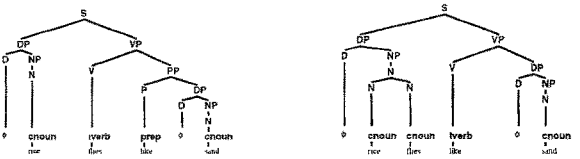
(a) Incomplete Sentence

(b) Complete Sentence

Fig. 2. Handling of Incomplete Sentences. The parser should recognise that a preposition is missing from the sentence *"the man went town", possibly enabling it to correct the error.

Our grammar is based on the lexicalised tree adjoining grammar given in [11]. We do not consider features, and have simplified the grammar considerably

to suit our experimental purposes. Nevertheless, we retain the null-adjunction constraint, and have also added a few trees to make the grammar more consistent with X-bar theory.



(a) "Rice" is the subject

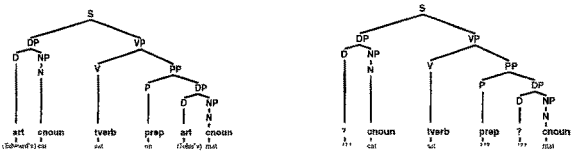
(b) "Flies" is the subject

Fig. 3. Handling of Ambiguous Sentences. The parser should not discriminate arbitrarily between the available interpretations. This example is a modification of the one in [12].

2.2 Accuracy of Parses

Earlier in this section, we described how parse trees for a particular sentence may be generated; it remains to define some measure of how accurate a parse is, and whether or not it contains missing constituents. In [11], a lexicalised grammar is used: each word in the input selects a set of trees which are then combined to form a parse; the final parses are assessed manually. In our scheme, each position in the sentence is assigned a dictionary of lexical entries based on the lexicon for the word at that position. For example, for the sentence "rice flies like sand", the dictionary at position 3 would be {tverb, prep}, hence the ambiguity (figure 3). It is now possible to evaluate the accuracy of a parse tree by considering how well its frontier matches the dictionaries assigned to the word-positions in the input. This is done using a modified version of the classic edit-distance algorithm given by Wagner and Fischer in [10]: where they used equality as a match condition, we use set-membership. This algorithm is known to find the minimum number of insertions and deletions necessary to make the two strings identical and thus provides a natural measure of how well a particular parse tree describes a given sentence (since the hierarchical structure of a sentence depends only on the grammatical rules used to parse it). We use a wildcard dictionary entry for unknown words: this allows sentences with missing or misspelt words to be parsed successfully whilst preserving the structure of the known parts of the sentence.

Using the string edit-distance in this way imparts a measure of "intelligent" behavior to the algorithm. Suppose the input is "Edward's cat sat on John's mat" and that neither "Edward's" nor "John's" is in the lexicon. The best parse is still that shown in figure 4 (a). Similarly, the best parse of *"cat sat mat" is that given in figure 4 (b).



(a) Unknown Words

(b) Missing Words

Fig. 4. Extreme Examples. In (a), words in parentheses are unknown, and may match anything (they must match something); the edit-distance is 2 without wildcards, 0 with. In (b) the essential structure of the utterance is recovered even though it is badly formed; the edit distance is 3. In both cases, the edit-distance algorithm allows the parser to make maximal use of the available information, yielding "correct" parses of the input.

3 Genetic Algorithms

The genetic algorithm (and variants thereof) is a well-known population-based optimisation method [13, 14] - reviews may be found in [15] and [16]. Briefly, a population of candidate solutions to a problem (the *individuals*), usually encoded as binary strings, is iteratively subjected to crossover in which parts of two individuals are mixed to yield two offspring, mutation in which one individual is subject to random change, and selection in which individuals are stochastically chosen to form the next generation. A measure of the quality of the solution represented by the individual is its *fitness*, which is translated into a probability of its survival into the next generation. Elsewhere we have demonstrated that the algorithm lends itself to problems which may be decomposed or partially decomposed and which have many local and global optima, for example line labelling [17, 18]. The algorithm composes a good solution by mixing sub-solutions with the crossover operator. Mutation operates at a low level as a source of background variation which allows new information to enter the population. The stochastic nature of selection allows the population to escape local optima.

Natural language processing can be viewed as an instance of the consistent labelling problem first formulated by Haralick and Shapiro in the 1970s [19]. The goal of the parser is to label the words in the input sentence with their lexical categories. However, unlike one-dimensional labelling problems, it is also necessary to construct a hierarchical representation of the sentence, its parse tree. We have demonstrated that genetic algorithms are suitable for labelling problems [18], especially when it is necessary to obtain several closely related solutions simultaneously [20]. Our interest here is the algorithm's solution-editing framework (crossover and mutation) rather than its optimisation properties which are not certain - indeed for medium (40 to 60 lines) line labelling problems, exhaustive search comfortably outperforms the algorithm. The rest of this sec-

tion formulates our version of natural language processing with a tree adjoining grammar for the genetic algorithm.

3.1 Solution Encoding

The population in our genetic algorithm consists of a set of initial trees drawn (initially at random) from the set of initial trees in the grammar. A natural way to encode trees is as strings. These are technically Lisp-like S-expressions, but we refer to them as S-trees to emphasise their structure. The grammar for constructing S-trees is given below.

```
  STREE --> '(' label ['(' SUBTREES ')'] ')'
SUBTREES --> STREE | STREE SUBTREES
```

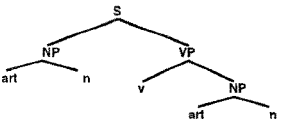


Fig. 5. S-Tree Example. This is the graphical representation of the S-tree `(S((NP((art)(n)))(VP((v)(NP((art)(n)))))))`.

Label is the label of a node in the tree which must be a string of characters. As an example, figure 5 gives a graphical representation of the S-tree `(S((NP((art)(n)))(VP((v)(NP((art)(n)))))))`. A null-label may be applied to any symbol in the tree, for example `(D:-)` is an empty D(eterminer). Other assignments are made by the edit-distance algorithm: when an assignment is inconsistent or involves an unknown word, it is denoted by an asterisk, '*'. Thus, the trees in figure 4 would be denoted as shown in table 1.

In principle, any tree can be represented in this manner, although the representation becomes clumsy when the node-labels are complex or the nodes contain a large amount of information. Simple features such as the null-adjunction constraint can be added by prefixing the node label with special characters according to some simple regular grammar. The advantage of this representation is its transparency, the ease of extracting the frontier while computing the edit-distance from the input, and the fact that the operations described in sections 2 and 3.3 can be implemented by matching and copying substrings. This representation is also compact.

3.2 Fitness Measure

The edit-distance cost function described in section 2.2 is a discrete measure which takes values from the set $\{0, 1, ..., N\}$, where N is the larger of the number of words in the input and the number of nodes in the frontier of the tree. To convert this into a fitness measure suitable for use with a genetic algorithm we exponentiate: $F_i = \exp(-\beta.ED_i)$, where ED_i is the edit-distance of the i^{th} S-tree, F_i is its fitness and β is an arbitrary scaling constant which defaults to 1.

To convert this into a survival probability for the selection step, we divide by the total fitness of the n members of the population: $P_i = F_i / \sum_{j=1}^{n} F_j$.

Table 1. String Representation of Parse Trees. Inconsistent assignments are marked with asterisks ('*').

(a) UNKNOWN WORDS

```
(S(
 (DP(
  (D((*art:Edward's)))
  (NP((N((cnoun:cat)))))))
 (VP(
  (V((tverb:sat)))
  (PP(
   (P((prep:on)))
    (DP(
     (D((*art:John's)))
     (NP((N((cnoun:mat))))))))))))
```

(b) MISSING WORDS

```
(S(
 (DP(
  (*D:-)
  (NP((N((cnoun:cat)))))))
 (VP(
  (V((tverb:sat)))
  (PP(
   (P((*prep:-)))
   (DP(
    (*D:-)
    (NP((N((cnoun:mat))))))))))))
```

3.3 Genetic Algorithm Operators

Crossover is implemented via *subtree crossover*, which we define as a special case of Koza's operator [21], but a more general case of the *substitution* operator used in [11]. Given two parent S-trees, the algorithm selects a node from each subject to the conditions that (1) the nodes have identical labels and are not subject to null-adjunction, (2) the subtrees rooted at the nodes are different, (3) the remainders of the parent trees following excision of the subtrees are different, (4) at least one of the nodes is not the root node, and (5) at least one of the subtrees is not empty. In practice, several pairs of nodes may satisfy these criteria: in this case the crossover sites are selected at random. A drawback of subtree crossover is that not all pairs of trees can be crossed, and only a subset of the nodes may be part of a crossover site in a particular crossing, both of which make the term "crossover rate" a little harder to define. However, since our choice of interpretation is restricted by the input data, in practice only a relatively small number of S-trees will survive the first few iterations of the algorithm, and these are likely to be of similar types. This crossover will thus be less disruptive and explore the search space less well than more traditional operators.

Mutation does not have such a natural implementation. It does not make sense to simply relabel nodes in a parse tree: internal nodes must never be relabelled since they describe permitted phrase structures in the grammar, and frontier nodes cannot generally be relabelled because only certain classes of words can form particular constituents. The adjunction operator seems a reasonable choice since this makes a point-modification to a single initial tree. This is implemented by forming a set of adjoinable auxiliary trees, selecting one and then finding a suitable adjunction site in the initial tree (i.e. a node with the same label as the root of the auxiliary tree, and not subject to the null-adjunction constraint).

A major disadvantage of this implementation is that mutation can no longer necessarily be considered a background operator: adjunction is a fundamental

means by which novel parse trees are constructed. One way around this limitation is to adjoin every tree with every adjoinable auxiliary tree at every possible site in a preprocessing step. This step, which we call "expansion" may be repeated as many times as desired; however it is computationally expensive and is therefore only suitable when the sets of trees are small. Adjunction may have far-reaching effects on the structure of a tree which is in sharp contrast to the local nature of the standard mutation operator.

4 Experiments

In a preliminary study, our algorithm was tested on artificially constructed sentences; for added realism, we also used 16 sentences drawn from a letter from a funding body. We increased the complexity of some sentences by adding adverbs, adjectives and additional words and clauses. Sample sentences are given below.

```
1) All awards are available.
2) All studentship awards are available.
3) Our awards are clearly helpful.
4) Please remember to include your award reference number.
5) Please remember to include the award reference number in the top right hand
   corner of the address label.
```

The grammar consisted of 49 initial and 45 auxiliary trees (370 and 1138 with expansion). We tested the algorithm on several different parameter settings with and without a single pass of expansion. 25 trials were conducted for each sentence with a variety of population sizes and iteration limits; crossover and mutation rates were fixed at 0.9. Lower values of crossover and mutation rates were also tried: these yielded uniformly poor performance and are not reported here. The results are given in table 2.

4.1 Discussion

Elsewhere we have reported that for line labelling, mutation rate is the most accurate predictor of success rate, followed by population size and crossover rate with iteration limit playing little part [18]. Our initial results for language parsing agree to some extent with this: all the best runs had population sizes of 4000, and the limit on iterations does not seem to be particularly relevant. However, low values of the mutation and crossover rates tended to give poor performance. This is unsurprising since both operators are of fundamental importance in the parsing process.

The need for high population size is remarkable, since the best population size we tested was about 100 times the cardinality of the unexpanded initial tree set. This is probably due to the well-known problem of premature convergence - we have shown previously that the diversity of the population decreases sharply in the first few iterations [20]. Thus for complex sentences there may be relatively few avenues open for search after the initial phase of the algorithm.

It is clear that grammar expansion does not help. This can perhaps be explained in terms of the structure of the input sentence. A "deep" sentence is

Table 2. Experimental Results. 25 runs were performed for each sentence to give a total of 400 runs. The crossover and mutation rates were fixed at 0.9. Some combinations were tried more than once.

POPULATION SIZE	ITERATION LIMIT	ACCURATE PARSES	
		expansion	no expansion
100	1000	0%	13%
200	500	0%	13%
500	200	0%	13%
1000	100	6%	25%
2000	50	6%	25%
		-	17%
2000	100	0%	19%
4000	25	6%	13%
4000	50	6%	31%
		-	22%
4000	100	6%	25%
		-	27%

one which requires many adjunctions to generate a correct parse: it has a lot of structure and will typically contain many modifiers. It appears that the deep structure of sentences is relatively inaccessible to the algorithm, since many specific adjunctions are required to generate a correct parse. Thus, a single pass of the expansion step is unlikely to substantially simplify the task of parsing complex sentences: only those sentences with one or two levels of complexity are made significantly easier. However, the expansion process does create a large number of "spurious" initial trees: it is likely that without expansion all 49 initial trees would be represented in a population of 1000, with few auxiliary trees to choose from for adjunction. Blindly increasing the numbers of initial and auxiliary trees appears to effectively increase the probability of making an incorrect choice.

5 Conclusion

The main contribution of this paper has been to investigate the use of discrete graphical editing operations within an optimisation framework. We have adapted the genetic algorithm for use with a tree adjoining grammar, and demonstrated its utility with a simple natural language processing example.

It is clear that there are several directions in which this work can be developed. It is worth investing some time fine-tuning the genetic algorithm's control parameters; these are notoriously difficult to set *a priori* [22–24] . To use our framework for serious language processing would require considerable work on augmenting the grammar and lexicon. We intend to develop the work by exploring more complex hierarchical problems furnished by vision, for example multilevel scene analysis.

References

1. A. K. Joshi. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In L. Karttunen D. R. Dowty and A. M. Zwicky, editors, *Natural Language Parsing.* Cambridge University Press, 1985.

2. A. Sanfeliu and K. S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE SMC*, 13:353–362, 1983.
3. M. A. Eshera and K. S. Fu. A graph distance measure for image analysis. *IEEE SMC*, 14:398–407, 1984.
4. D. Shasha and K. Zhang. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18:1245–1262, 1989.
5. K. Zhang. A constrained edit distance between unordered labeled trees. *Algorithmica*, 15:205–222, 1996.
6. D. B. Skillicorn. A parallel tree difference algorithm. *Information Processing Letters*, 60:231–235, 1996.
7. Richard. C. Wilson Andrew. D. J. Cross and Edwin. R. Hancock. Inexact graph matching using genetic search. *Pattern Recognition*, 30:953–970, 1997.
8. R. C. Wilson and E. R. Hancock. Hierarchical discrete relaxation. *Lecture Notes in Computer Science*, 1121:120–129, 1996.
9. S. Geman E. Bienenstock and D. Potter. Compositionality, MDL priors, and object recognition. In *NIPS 96*, pages 838–844, 1996.
10. R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21:168–173, 1974.
11. T. Becker et al. A lexicalised tree adjoining grammar for english. Technical report, University of Pennsylvania, 1995. IRCS Report 95-03.
12. J. Allen. *Natural Language Understanding.* Benjamin/Cummings, 1994.
13. J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1975.
14. G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5:96–101, 1994.
15. D. B. Fogel. An introduction to simulated evolutionary optimisation. *IEEE Transactions on Neural Networks*, 5:3–14, 1994.
16. M. Srinivas and L. M. Patnaik. Genetic algorithms: A survey. *IEEE Computer*, 27:17–26, 1994.
17. D. A. Huffman. Impossible objects as nonsense sentences. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 6, pages 295–323. Edinburgh University Press, 1971.
18. R. Myers and E. R. Hancock. Genetic algorithm parameters for line labelling. *Pattern Recognition Letters*, 18:1363–1371, 1997.
19. R. M. Haralick and L. G. Shapiro. The consistent labelling problem: Parts 1 and 2. *IEEE PAMI*, 1 (I) and 2 (II):173–184 (I) and 193–203 (II), 1979 (I) and 1981 (II).
20. R. Myers and E. R. Hancock. Genetic algorithms for ambiguous labelling problems. *Lecture Notes in Computer Science (EMMCVPR 97)*, 1223:345–360, 1997.
21. J. R. Koza. *Genetic Programming.* MIT Press, 1992.
22. J. J. Grefenstette. Optimisation of control parameters for genetic algorithms. *IEEE SMC*, 16:122–128, 1986.
23. L. J. Eshelman J. D. Schaffer, R. A. Caruna and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimisation. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60, 1989.
24. K. A. DeJong and W. M. Spears. An analysis of the interacting rôles of population size and crossover in genetic algorithms. In *Proceedings of the First Workshop on Parallel Problem Solving from Nature.* Springer-Verlag, 1990.