Exploration of Picture Grammars, Grammar Learning and Inductive Logic Programming for Image Understanding

P. G. Ducksbury[‡], C. Kennedy, Z. Lock

QinetiQ, Advanced Processing Centre, St Andrews Road, Malvern, Worcs, WR14 3PS, UK.

ABSTRACT

Grammars have been used for the formal specification of programming languages [1], and there are a number of commercial products which now use grammars. However, these have tended to be focused mainly on flow control type applications. In this paper, we consider the potential use of picture grammars and inductive logic programming in generic image understanding applications, such as object recognition. A number of issues are considered, such as what type of grammar needs to be used, how to construct the grammar with its associated attributes, difficulties encountered with parsing grammars followed by issues of automatically learning grammars using a genetic algorithm. The concept of inductive logic programming is then introduced as a method that can overcome some of the earlier difficulties.

Keywords : Grammars, Genetic Algorithms, Inductive Logic Programming, Object Recognition, Feature Detection

1. INTRODUCTION

Grammars have been used for the formal specification of programming languages [1], the *Backus-Naur Form* (BNF) being the oldest formalism. A number of commercial products exist which use grammars, for example; Prograph [11], Labview [12], IRIS Explorer [13], BALSA [14], ARK [15] to name a few. These products are mainly (but not exclusively) concerned with flow control type applications. In this paper, we provide an overview of grammars and describe a possible use of them in image understanding such as site and object detection. The paper, firstly describes experiments in both manual and automatic construction of grammars, the automatic construction being applied using a genetic algorithm. Difficulties may be present in the construction of an accurate set of rules for a grammar, for the recognition of a particular object. For this reason the paper then describes how inductive logic programming can be employed to overcome the difficulties, by approaching the recognition problem from a data driven approach.

There are a number of different types of grammars referred to in the literature; for example, context-free grammars, context-sensitive graph grammars, layered graph grammars, attribute grammars, attributed multisets and attributed multiset grammars. In a context-free grammar, every left hand side consists of a single non-terminal node, whilst in a context-sensitive graph grammar both left and right side of a production are graphs. In layered graph grammars, the left hand side must be lexico-graphically smaller than its right hand side[1]. Attribute grammars are context-free grammars with some formal mechanism (attributes, evaluation rules, conditions) which enable non context-free aspects to be specified. The value associated with an occurrence of an attribute in a tree is determined by the mechanisms associated with the grammar's production rules. Attributed multisets are an unordered collection of attributed symbols, which provide the basis for representing visual programs. Attributed multiset grammars remove the notion of ordering present in a string grammar, such attributes are an integral part of the parsing. Details of work in visual languages and grammars can be found in [2-7, 10, 16-17] and in particular the work of Golin [8-9], which we refer to for our discussion on parsing, whilst inductive logic programming is described in [21-23].

2. ATTRIBUTE GRAMMARS

Attribute grammars are 'context free grammars' with formal devices (*attributes, evaluation rules, or conditions*), which enable non-context free aspects to be specified. In the literature, 'constraint grammars' appear, which can be considered as 'attribute grammars', in addition to 'relational grammars' which are members of the 'constraint' set of grammars.

‡ Correspondence : P.G.Ducksbury, Email : p.ducksbury@signal.qinetiq.com

[1] claimed to avoid cyclic derivations.

Each symbol may have an attribute associated with it and each attribute may have a domain of values as well as a logical condition that expresses some constraint. The value associated with an occurrence of an attribute in a tree is determined by evaluation rules, which are associated with the grammar's production rules. The ability to associate an attribute, or a set of attributes, makes the use of grammars far more desirable. Attributed multisets are an unordered collection of attributed symbols, which provide the basis for representing visual programs, whilst attributed multiset grammars remove the notion of ordering present in a string grammar. The attributes are an integral part of the parsing.

3. GRAMMARS FOR IMAGE UNDERSTANDING

Many of the picture grammar papers in the literature are based on building graphs from an edge structure within an image. This work looks at higher level aspects and makes the assumption that some level of image processing has been applied so that components, and/or, objects have been detected possibly with some associated confidence level.

3.1 Vehicle Identification

An example used to demonstrate the approach, is that of recognising vehicle types given some symbolic input. This example is relevant from a surveillance consideration, for example, vehicle type, colour, vehicle recognition number and distinguishing marks, etc can all be used to recognise vehicles with false registration plates. Consider some simple symbolic representations of vehicles as shown in figure 1, the relationship between the basic components being shown in figure 2. Here the vehicle headlights need to be within some relatively tight set of constraints compared with the grill. If this was not the case we could be in a position in which we are parsing the (possible) fog lights and not finding a suitable grill/badge in between. In addition to this, the orientation of some headlights relative to the grill acts as a discriminating factor, as well as, measurement inaccuracies from the feature detection stage for the individual components and the fact that vehicles may be parked on a sideways gradient. The grammar productions are generally written with only one or two symbols on the right hand side[2]. We can now derive a set of rules (written in Golin's [8] notation) of the following form :

1	car →	car-audi
2	car →	car-vw1
3	car →	car-vw2
4	car →	car-ford1
5	car →	car-ford2
6	car →	car-ford3
7	car-audi →	left_of(rectangle, remainder-audi)
8	remainder-audi →	left_of(radiator-audi, rectangle)
9	car-vw1 →	left_of(rectangle, remainder-vw1)
10	remainder-vw1 →	left_of(radiator-vw, rectangle)
11	car-vw2 →	left_of(circle, remainder-vw2)
12	remainder-vw2 →	left_of(radiator-vw,circle)
13	car-ford1 →	left_of(circle, remainder-ford1)
14	remainder-ford1 →	left_of(radiator-ford, circle)
15	car-ford2 →	left_of(rectangle, remainder-ford2)
16	remainder-ford2 →	left_of(radiator-ford, rectangle)
17	car-ford3 →	left_of(ellipse, remainder-ford3)
18	remainder-ford3 →	left_of(radiator-ford, ellipse)
19	radiator-audi →	contains(grill-audi, badge-audi)
19	radiator-audi →	grill-audi
20	radiator-vw →	contains(grill-vw, badge-vw)
20	radiator-vw →	grill-vw
21	radiator-ford →	contains(grill-ford, badge-ford)
21	radiator-ford →	grill-ford

[2] Production rules with more than two symbols on the right hand side are rewritten with additional sub rules.

22 contains → containsoperator
Val(contains) ← Val(xs_1,xf_1,ys_1,yf_1,xc,yc)
Condition : Val(contains)
(xc > xs_1 ^ xc < xf_1) ^ (yc > ys_1 ^ yc < yf_1)

23 left of → leftofoperator
Val(leftof) ← Val(c_1,c_2,c_3,xs_1,xf_1, ys_1,yf_1,xs_2,xf_2, ys_2,yf_2,θ)
Condition : Val(contains)
(xs_1 < xs_2) ^ (xf_1 < xf_2) ^ (xf_1 < xs_2) ^ (| xs_2 – xf_1 | < c_1) ^
(| ys1 – ys2 | < c2) ^ (| yf1 – yf2 | < c3) ^ (θ < ± c_4)

24 badge-audi → BADGE-A
25 badge-vw → BADGE-VW
26 badge-ford → BADGE-FORD
27 grill-audi → trapezium
28 grill-vw → trapezium
29 grill-ford → ellipse
30 rectangle → RECTANGLE
31 circle → CIRCLE
32 ellipse → ELLIPSE
33 trapezium → TRAPEZIUM

The above grammar can be represented simply in a 2D table that can be parsed by the software along with a similar table representing a potential object. This grammar definition is not all encompassing, but does make the parsing process simpler.

Using the parsing algorithm (figure 4) described in Golin's paper [8], a parse tree can be produced for a particular input consisting of a set of circles (C), trapeziums (T), badges (B) and rectangles (R). An example is shown in figure 5 in which it can be seen that invalid hypotheses develop as the initial parsing proceeds, these are then refined as the parsing continues.

4. EVIDENCE FUSION

Probabilities of detection of individual components have been included into this grammar scheme by means of a quantised distribution representing the belief in an object (currently used as a hypothesis with 5 states of belief ranging from low through to high). The probabilities are used as input in the layout file and represent an approximate distribution centred on the level of belief. These probabilities are fused using a simple belief network type of approach. Each grammar rule, which joins two nodes (hypotheses X and Y) together into a new hypothesis Z, applies the following belief equation

$$BEL(Z_i) = \alpha \sum_{j,k} P(Z_i | X_j, Y_k) \pi(x_j) \pi(y_k)$$

α is a normalising factor and π() the causal evidence from the two rules that are being combined whilst the *P*() is the set of prior probabilities, the indexes *i, j* and *k* range over the 5 states of belief, a full derivation of the belief equations are to be found in [24]. The prior probabilities are currently constructed according to the following rules[3]

$$P(Z_i | X_j, Y_k) = \begin{cases} 0.9 & if \quad i=j=k \\ 0.7 & if \quad (i=j) \wedge (|i-k| \le 1) \\ 0.3 & if \quad (i=k) \wedge (|i-j| \le 1) \\ 0.6 & if \quad (i=j) \wedge (|i-k| > 1) \\ 0.1 & if \quad (i=k) \wedge (|i-j| > 1) \\ 0.1 & otherwise \end{cases}$$

such that $\sum_{j,k} P(Z_i | X_j, Y_k) \le 1 \quad \forall_i$

[3] This will eventually be replaced with a more formal representation of the prior knowledge, given suitable training data.

These rules construct a banded three dimensional matrix which has the effect of clustering similar input evidence together. At present these probabilities are filtered through the tree as the parser dynamically constructs it. However, there is no reason why the computed belief value could not be included with the attributes of the grammar rules in some way, thus the construction of the tree would depend upon the evidence filtering through it. In addition to this, each rule that combines other pairs of rules should ideally have its own independent copy of a more suitable conditional probability matrix (lack of training data has currently prevented this).

5. LEARNING GRAMMARS USING GENETIC ALGORITHMS

One of the difficulties that has been apparent in applying image processing algorithms not just for automatic target recognition but also for associated tasks in image processing and understanding is that of the optimal choice of parameters and algorithms. In previous work [20] we used genetic algorithms (GA) to optimise the selection of parameters and algorithms for feature detection, here we consider them for the task of learning a grammar for object recognition.

5.1 Search Strategies

The problem we are addressing is essentially a constrained optimisation one. Whenever a constraint is violated the solution would be infeasible and should therefore have no fitness value. In some highly constrained domains obtaining feasible points may prove to be a difficult task. Penalty type methods try to obtain information from infeasible solutions by the degrading of the fitness in relation to the relative degradation in the constraint violation. This approach is somewhat difficult in our domain as the objective function is actually 'the parsing of a grammar' and it may not be possible to parse an infeasible grammar[4]. Different GA strategies for searching are described by Goldberg [18].

5.2 Grammar Encoding

Encoding of the grammar is achieved using a simple intuitive and direct mapping of the rules into strings, figure 3. Each rule of the grammar is therefore encoded into a bit string with distinct substrings representing the left hand side, right hand side, spatial operators, terminal symbols etc. Given this encoding an initial population of grammars can be built by randomly generating two-dimensional matrices of bit strings. Constraints are applied which cause the removal of infeasible grammars, e.g.

- Spatial operators and terminal symbols to be limited to some predefined set (i.e. left-of, right-of, in-front, behind, contains)
- No spatial operator chosen implies there can be only one right hand side for a rule
- Spatial operator chosen implies there must be two right hand sides for a rule
- Rule numbers are to be constrained to some valid set
- Rules are not to be recursive
- Rules must be allowed to reference the terminal set

These constraints result in a highly constrained but feasible population. Given this initial population we can now perform the operations of crossover and mutation as described below.

5.3 Crossover and Mutation

In performing crossover we are essentially playing a game of mix and match with members of the population that are highly correlated with previous success. When it comes to the operations of crossover and mutation a variety of different approaches exist, for example see Goldberg [18]. The current approach consists of a combination of an elitist strategy with a tournament selection, all members being chosen using a roulette wheel based selection strategy as described below.

[4] It may be interesting to incorporate some form of sensitivity analysis to see if information can be gained from infeasible grammars, e.g. is it possible to parse parts of grammar to obtain some useful information.

The elitist approach copies the 'best' solutions from one generation to the next thereby guaranteeing that the GA never loses its best solutions. The tournament selection chooses one member at random from the population as a parent and then chooses two others, these then essentially compete for the right to breed with the first chosen parent based on which of the two has the highest confidence measure. Roulette wheel selection[5] is used whenever it is required to choose a member of the population at random. Individual slots within the wheel are weighted in proportion to the members confidence measure (*or fitness value*). The crossover of the selected parents is done by a simple exchange of substrings between the two, the location within the parents being randomly chosen. If the new members are either infeasible or duplicates, then they are rejected on the grounds that they would waste valuable population space and reduce the diversity. This crossover process continues to iterate until a new population has been generated. The mutation operation again consists of choosing members at random using the roulette wheel selection and then mutating just a single bit within the string chosen at random.

The next required stage prior to the application of the GA is to produce the cost (fitness function), C, to assess to how good is a particular grammar in the population. This is defined as

$$C = \frac{N_2}{\left(T_n \middle/ (T_n - r)!\, r! \right)} + \frac{T_r}{T_n}$$

where T_r is the maximum number of terminal nodes reachable from a single path within the parse tree, N_2 is the number of nodes in the parse tree that have two children, T_n the number of terminal symbols in the object configuration and r is 2. The first part of the cost function measures the ratio of the number of rules that combine other rules versus the maximum combination of 2 from the set, whilst the second part measures how many terminal nodes are reachable during the parse.

6. INDUCTIVE LOGIC PROGRAMMING

Machine learning (ML) is a well-established branch of artificial intelligence (AI). It concerns the ability of a machine to automatically improve its performance at a particular task in response to experience. ML is a wide field covering many types of techniques and the applications into which these techniques can be employed. On the whole, techniques can be classified as symbolic and sub-symbolic. The distinction between these two types of technique concerns the data representations that are manipulated during the learning process.

Symbolic techniques manipulate representations of whole concepts with respect to the domain in question. For example, the concept of a main battle tank may be represented by the symbol 'Tank' by the learning algorithm. When the learning process outputs a learnt hypothesis, a solution to the set problem, it can be understood in terms of the constituent components because these will correspond to original whole concepts. In other words, these techniques are transparent as an observer can interrogate the learned hypothesis to discover its meaning in terms of the inputs.

Sub-symbolic techniques, however, manipulate representations that do not correspond to whole concepts. In many cases, the representations of whole concepts are broken down into smaller 'sub-symbols' before being input into the learning algorithm. Normally, these 'sub-symbols' are single bits where whole bit strings correspond to concepts. Because the learning process manipulates the inputs at the 'sub-symbol' level it is extremely difficult to perfectly understand the hypothesis, or in fact its derivation, in terms of the original concepts. Such systems are said to be opaque and so are not forthcoming to interrogation. However, such sub-symbolic techniques are widely deployed because of the robustness of such representations.

The different representations can be employed to reflect the various data granularities and features that occur within a complex environment. For example, initial image analysis at the pixel level is suited to sub-symbolic techniques as there are no higher-level concepts to represent. Each pixel can be represented as a bit string. Symbolic techniques could

[5] It may be a possiblity that a strategy that is based upon the combination of elitism, tournament and roulette wheel selection is providing to much selective pressure and could be leading to premature convergence.

then be employed once higher-level concepts have been evaluated, for example, for learning picture grammars previously discussed. The main advantage of symbolic techniques is the audit trails they can provide.

GA's are a machine learning technique that falls into both the symbolic and sub-symbolic classes, depending on the particular decomposition of concept for the problem to be considered.

Inductive logic programming (ILP) [21-23] is the area of ML described as the intersection between inductive learning and logic programming. Its knowledge representation strategy is based on a subset of first order logic known as Horn clause logic. The logic programming language Prolog is typically used to represent both the training examples and the resultant induced set of rules.

The inputs to an ILP system are the example set and background knowledge. First order logic provides an expressive knowledge representation strategy that allows the use of expert *background knowledge* of the problem domain to augment the examples during learning. In most ILP systems, the example set consists of both positive and negative examples of the concept whose description is to be induced. The background knowledge is prior information that may be used to guide the search. The output of an ILP system is the learnt hypothesis. Inducing a logic program involves searching through the hypothesis space of all possible logic programs and ILP uses the generality ordering of hypotheses to guide the search. If a hypothesis is shown to cover a particular negative example, it is too general and must be specialised. If it is shown not to cover a particular positive example, it is too specific and must be generalised. Using this ordering, the search space is a feasibly small subset of the infinite hypothesis space available.

This setting for learning first-order rules also has the advantage that it can describe interactions or relationships between components. For example, the *member* predicate in Prolog takes a single item and a list of items as input and returns true if the item is a member of the given list and false otherwise. A training set could consist of the examples given in table 1. The correct description of the membership concept is the following:

```
member(X,[X]).
member(X,[Y|Z]) :- member(X,Z).
```

This means that, an item is a member of a list if it is identical to the first element in the list and an item is a member of a list if it is not identical to the first element but is a member of the rest of the list.

6.1 Training

The experience provided to a learner typically takes the form of a list of observations from the real world known as a training set. For concept learning, a training example is a specific instance of a concept. For example, an instance of the concept of parenthood may be the fact *Jane is a parent of John*. If Jane is John's parent, then the above example is positive. If she is not, then it is a negative example of the concept. Training examples often take the form of *attribute-value pairs*, for example <*first name = John*> *and* <*mother = Jane*>. Others are represented as facts in first order predicate logic, for example, *mother (John, Jane).* The representation of training examples depends on the problem domain and the ML technique to be employed.

6.2 Testing

The aim of an inductive ML algorithm is to find a hypothesis that fits all of the training examples, i.e. that classifies them correctly. Once a hypothesis has been learnt, it must be evaluated before its deployment can be considered. For this reason, another example set is required in order to test and quantify the performance of the hypothesis. This example set is known as a test set. Usually, the training and test sets are derived from the set of original observations. The performance of the hypothesis over the test set can be measured using various well-known metrics. Solutions derived through the use of different ML techniques can also be compared using the same test sets. Example data came from a 3D aircraft database (figure 9) and consisted of lines forming the outline of an aircraft, start point co-ordinates, end point co-ordinates and elevation. Ten different types of aircraft where used at 12 orientations (taken at 15º intervals). The background information being general 'line' relations, including proximity, junction, triple junction, same length, proper parallel and right angle.

Two concept learning approaches where employed, firstly using two types of aircraft and all orientations and secondly using all types of aircraft with single orientation. In addition to this the impact of different types of background knowledge was explored.

7. RESULTS

Taking the manually constructed grammar from Section 3, together with the original vehicle layout and applying the parsing algorithm of figure 4 produces the constructed tree shown in figure 5. It can be seen from this that a number of multiple hypotheses are generated, for instance the two that are indicated show the parse of a radiator for a VW with no badge and similarly for an Audi. The tree also contains three root nodes, two of which are considered to be invalid/infeasible hypotheses. For example, the root nodes '21' and '22' do not generate valid parses of the tree whilst root node '20' does. The example grammar shown can correctly parse a variety of simple symbolic representations of the fronts of cars including those with missing badges. A number of issues arise from this and need to be addressed :

- Multiple parses may be created some of which may be invalid. Once a tree is generated a valid parse must be obtained from it.
- Undeveloped nodes may exist, i.e. these are nodes that cannot be reached from the root node (infeasible parses)
- Overlapping trees (overlapping grammars), since a given node may have more than one path through it.

The problem of multiple parses can be dealt with by recursively searching the tree from each of its root nodes to ensure that we can reach all the terminal symbols corresponding to objects in the vehicle layout (i.e. we can produce a spanning tree covering all terminal symbols). The issue of undeveloped nodes we choose to ignore for the present, as they start off as locally correct but then become globally incorrect as the parse develops. They have become 'infeasible' and can not be reached from the root node/s. The issue of overlapping trees can be detected by having a counter on each node that records the number of other nodes pointing into it, this could then be reported.

Taking the automatically constructed grammars, figure 6 illustrates a parse from a grammar produced during the evolution process, in which, not all objects have been linked together. Figure 7 shows a parse produced by a grammar after convergence has been achieved, all objects now being linked together. A small test set of 20 vehicle layouts was used and included invalid configurations such as vehicles with a circular headlight on one side and rectangular on the other side. Figure 8 shows a graph illustrating a characteristic of this particular genetic algorithm, namely the convergence with varying population sizes. For populations which are too small we can see that convergence to a global optimum has failed (the process being locked into a local minimum – the lower lines on the graph) whereas with the larger populations we move to the locality of a good solution more quickly. Goldberg [18] pointed out that De Jong [19] had observed that improved results could be obtained with reduced elitist percentages on some functions and suggested that elitism can improve local search at the expense of a global perspective.

For inductive logic programming, figure 9 illustrates some examples of boundaries extracted from 3D views of the given aircraft. From these, the line segment information was used to learn the hypothesis. Figure 10 shows the marked line segments that where learnt for recognising an A10 and a Saab-105 respectively. The learnt hypotheses being,

A10 :

part of(A,B),
proper_parallel(C,B),
right_angle(C,D),
proper_parallel(E,D),
right_angle(B,D)

SAAB 105 :

right_angle(A,B),
triple_junction(C,A,B),
same_length(C,D),
same_length(E,F),
proper_parallel(G,E),
proper_parallel(H,I),
proper_parallel(H,J),
proper_parallel(H,K).

In the case of the SAAB the last three primitives were learnt as part of the hypothesis but weren't needed for actual recognition.

8. CONCLUSIONS

These examples have shown that it is possible to write down an attributed grammar. These grammars can then assist in the recognition of objects. Some of the difficulties with picture layout grammars and/or multiset grammars are the requirements that are enforced for explicitly representing all spatial relationships, the difficulties in writing out a grammar manually[6] and the complexity of the grammar for realistic detection. It has also been demonstrated that in principle a genetic algorithm is a suitable search mechanism for automatically learning the structure of a picture grammar. A number of generic issues remain however. Multiple and missing objects. Multiple objects, due to the situation where feature detectors produce more objects, say, for example, additional rectangles around the actual vehicle, should not pose any difficulties. Either the attributes within the grammar should remove these or they are accepted into the tree, then, if the correct vehicle is there anyway then this is effectively a subset of the tree[7]. For missing objects, additional rules can be included to deal with this situation explicitly. In the case of the learnt grammars it has been possible to generate grammars that describe individual vehicle types/models but not grammars that describe all vehicles or all models for some vehicle type. The difficulty has been the convergence with the population becoming locked into local minima, as well as multiple and redundant rules being evolved within the grammar which may be using up valuable search space. The later could be removed during the population generation, crossover and mutation by application of a uniqueness test to the parts of the bit string that correspond to the spatial operators, and the right hand side. The ILP can reduce the difficulty in specifying all the specific spatial relationships in a grammar by learning at a symbolic level. We would however, envisage a multi-functional and hierarchical approach in which both ILP and picture grammars worked together to assist in recognition of objects.

9. ACKNOWLEDGEMENTS

This research was sponsored by the United Kingdom Ministry of Defence, Corporate Research Programme, TG10. © Copyright QinetiQ Ltd 2003. We also acknowledge Emma Peeling for the contribution towards ILP implementation as well as testing and evaluation.

10. REFERENCES

1. R. Bardohl, G. Taentzer, M. Minas, A. Schürr, ‘Application Of Graph Transformation To Visual Languages’ (1998), In H. Ehrig, et al., Handbook of Graph Grammars and Computing by Graph Transformation, volume II: Applications, Languages and Tools, pages 105-180. World Scientific, 1999.
2. Wittenburg, K., and L. Weitzman, ‘Relational Grammars: Theory and Practice in a Visual Language Interface for Process Modeling’. In Proceedings of Workshop on Theory of Visual Languages, May 30, 1996, Gubbio, Italy.
3. Haarslev, V., ‘A fully formalized theory for describing visual notations’, Proceedings of the AVI'96 post-conference Workshop on Theory of Visual Languages, Gubbio,Italy, 1996.
4. J. Rekers and Andy Schurr, ‘Defining and Parsing Visual Languages with Layered Graph Grammars’, Journal of Visual Languages and Computing, vol 8 no 1, p 27-55, 1997.
5. Costagliola G., et al., ‘Automatic parser generation for pictorial language’, IEEE Proc 1993, Symposium on visual languages, 1993.
6. Flasinski M., ‘Parsing of edNLC-graph grammars for scene analysis’, Pattern Recognition, vol 21, no 6, 623-629, 1988.
7. Shu Nan C., ‘Visual programming languages – a perspective and dimensional analysis’, Visual Languages, Plenuim Press, 1986, p11-34.
8. Golin E.J., ‘Parsing visual languages with picture layout grammars’, Journal of Visual Languages and Computing, 2(4) : 371-394, 1991.
9. Golin E.J., Reiss S.P., ‘The specification of visual language syntax’, in Proc IEEE workshop on visual languages, Rome, Italy, Oct 1989, pp 105-110.
10. Chang S.K., ‘Visual languages : A tutorial and survey’, IEEE Software, vol 4, pp 29-39, Jan 1987.

[6] Even the examples shown here have proved to be error prone
[7] A more advanced tree search mechanism is required.

11. Prograph, http://pictorius.com/prograph.html
12. Labview, http://amp.ni.com/niwc/labview
13. IRIS, http://www.nag.co.uk/Welcome_IEC.html
14. Brown M.H., 'Exploring algorithms using Balsa II', Computer, vol 21, no 5, May 1988, pp 14-36.
15. Smith R.B, (Xerox Parc.), 'Experiences with the Alternative Reality Kit', IEEE CG&A, 7(9):42-50,Sept 87.
16. Chok S., Marriot K., 'Parsing Visual Languages', In 18th Australasian Computer Science Conference, Glenog, South Australia, 1995.
17. Proc IEEE Symposium on visual languages, Los Amitos, CA, IEEE Computer Society Press, 1995 – 1997.
18. Goldberg D. E., 'Genetic Algorithms in Search, Optimisation and Machine Learning', *Addison Wesley*, 1989.
19. De Jong K.A., 'An analysis of the behaviour of a class of genetic adaptive systems', Doctoral dissertation, University of Michigan, 1975.
20. Ducksbury P.G., Varga M.J., Kent P.J., Foulkes S., Booth D.M., 'Genetic algorithms for automatic algorithm and parameter selection in ATR applications', SPIE Aerosense-98, vol 3371, 11-17 April, Orlando 1998.
21. Lavrač N., Džeroski S., *"Inductive Logic Programming: Techniques and Applications"*. Artificial Intelligence Series, Ellis Horwood (Simon & Schuster), 1994
22. Muggleton S., *"Inductive Logic Programming"*. Academic Press, London, 1992.
23. Muggleton S., DeRaedt L., "Inductive Logic Programming: Theory and Methods". *Journal of Logic Programming* 19/20: 629 - 679, 1994.
24. Ducksbury P.G., Varga M.J., 'Feature detection and fusion for intelligent compression', SPIE Aerosense 2001, vol 4380, April 16-20, 2001.

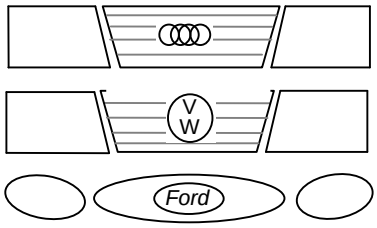

Figure 1; Example vehicle layouts

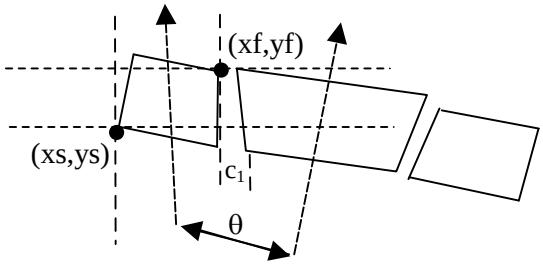

Figure 2; Vehicle alignments

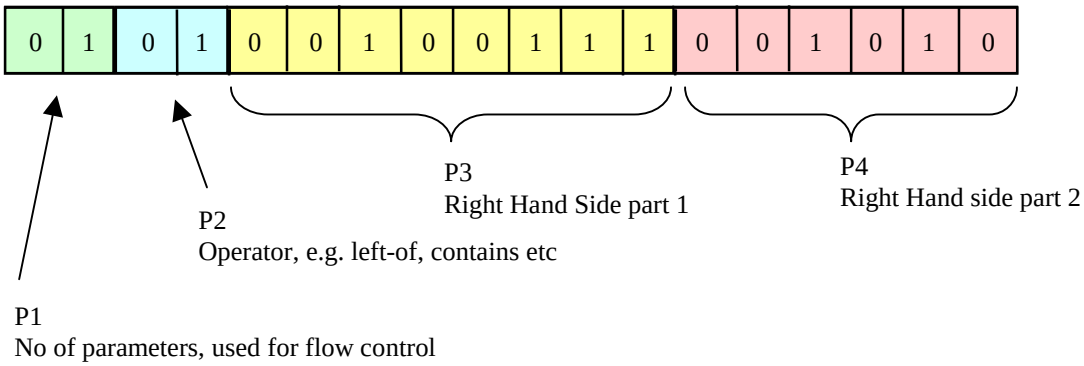

Figure 3; Genetic Algorithm encoding for grammar generation

```
Input          : An attributed multiset grammar and object layout (M)
Output         : A parse tree
Initialisation : set tree, todo and done to empty

for each b ∈ M do
        add a terminal node for b to todo and tree
while todo is not empty do
        let next = an element of todo, with X = symbol(X)
        for each production such that X occurs in the RHS do
                if (production = A → {X}) and all constraints are satisfied then
                        add a node for production to todo and tree
                else /* production has two operands */
                        for each occurrence of X in RHS of production do
                                let Y be the other symbol in production
                                for each old in done such that Y = symbol(old) do
                                        if all constraints are satisfied then
                                                add a node for production to todo and tree
        remove next from todo and add it to done
repeat
```

Figure 4; Golin's Parsing algorithm

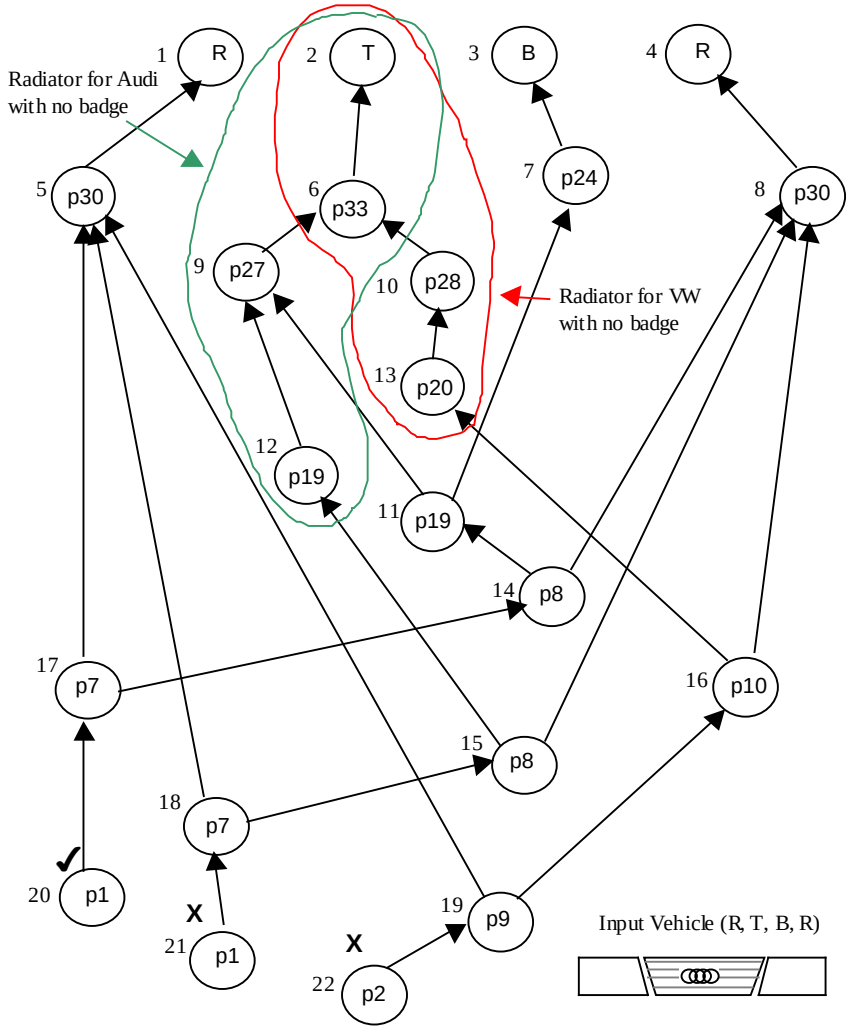

Figure 5; Parse tree

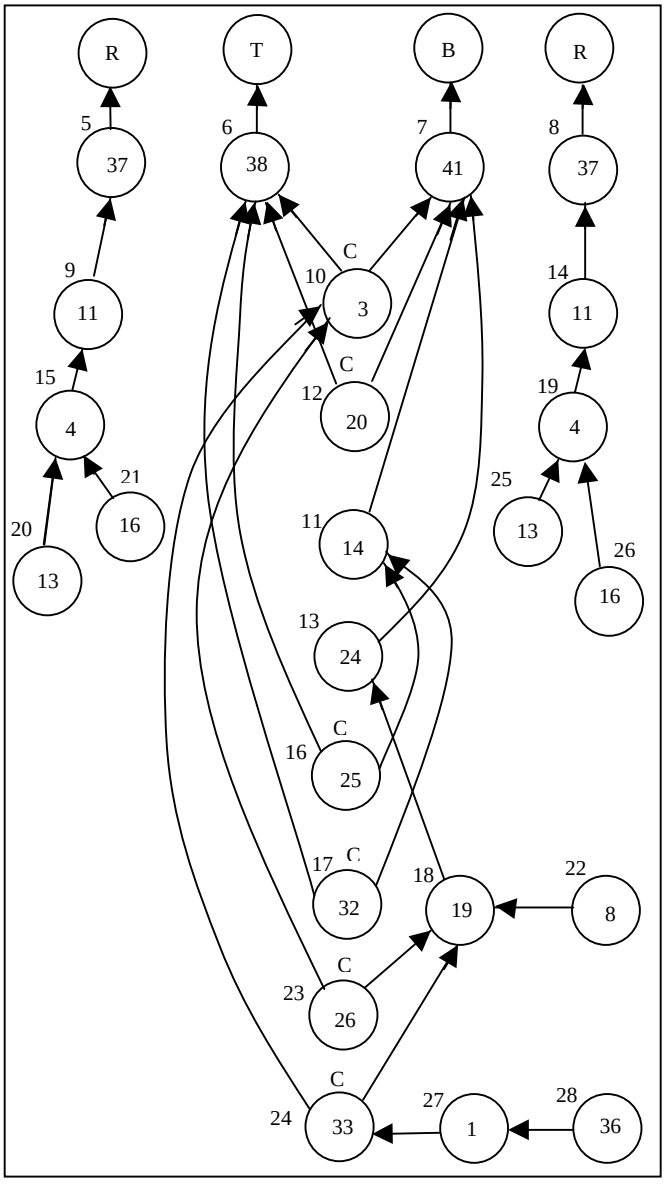

Figure 6, Parse tree from evolved grammar - during evolution

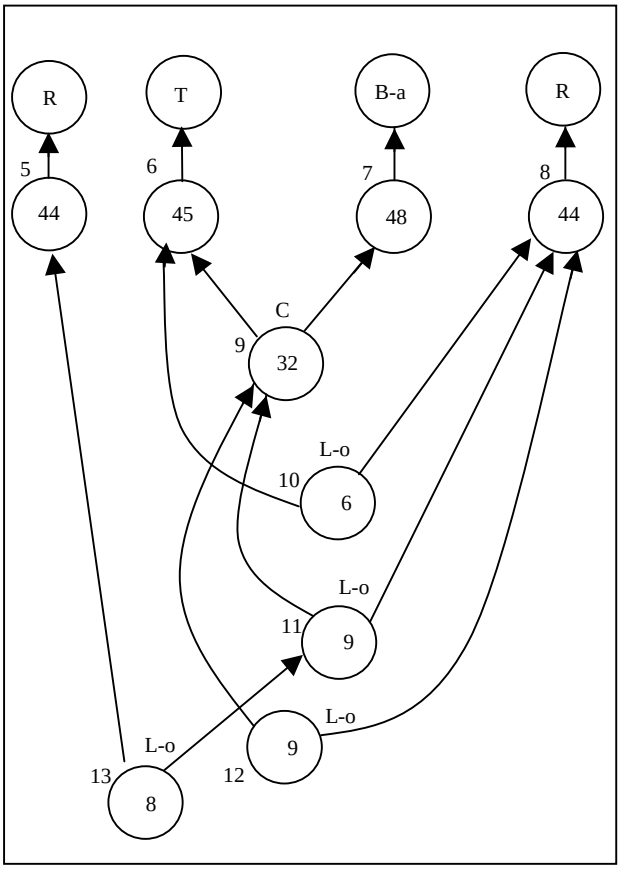

Figure 7, Parse tree from evolved grammar – after convergence

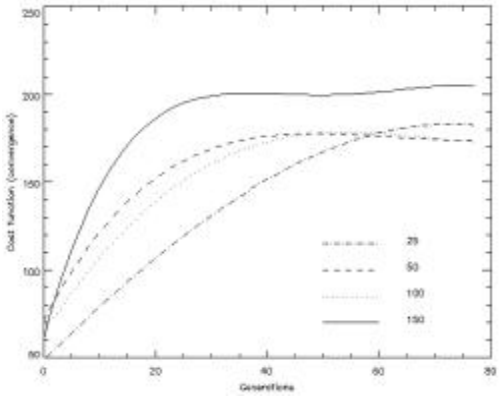

Figure 8, Grammar Convergence

Positive Examples		Negative Examples	
Item	List	Item	List
a	[a,b,c]	3	[2,4,6]
1	[1,3,5,7,9]	w	[x,y,z]
3	[1,1,2,3,4,8,11]	1	[2,3,4,5,6]
c	[a,1,b,2,c,3]	d	[a,1,b,2,c,3]
f	[f]	g	[h]

Table 1, ILP

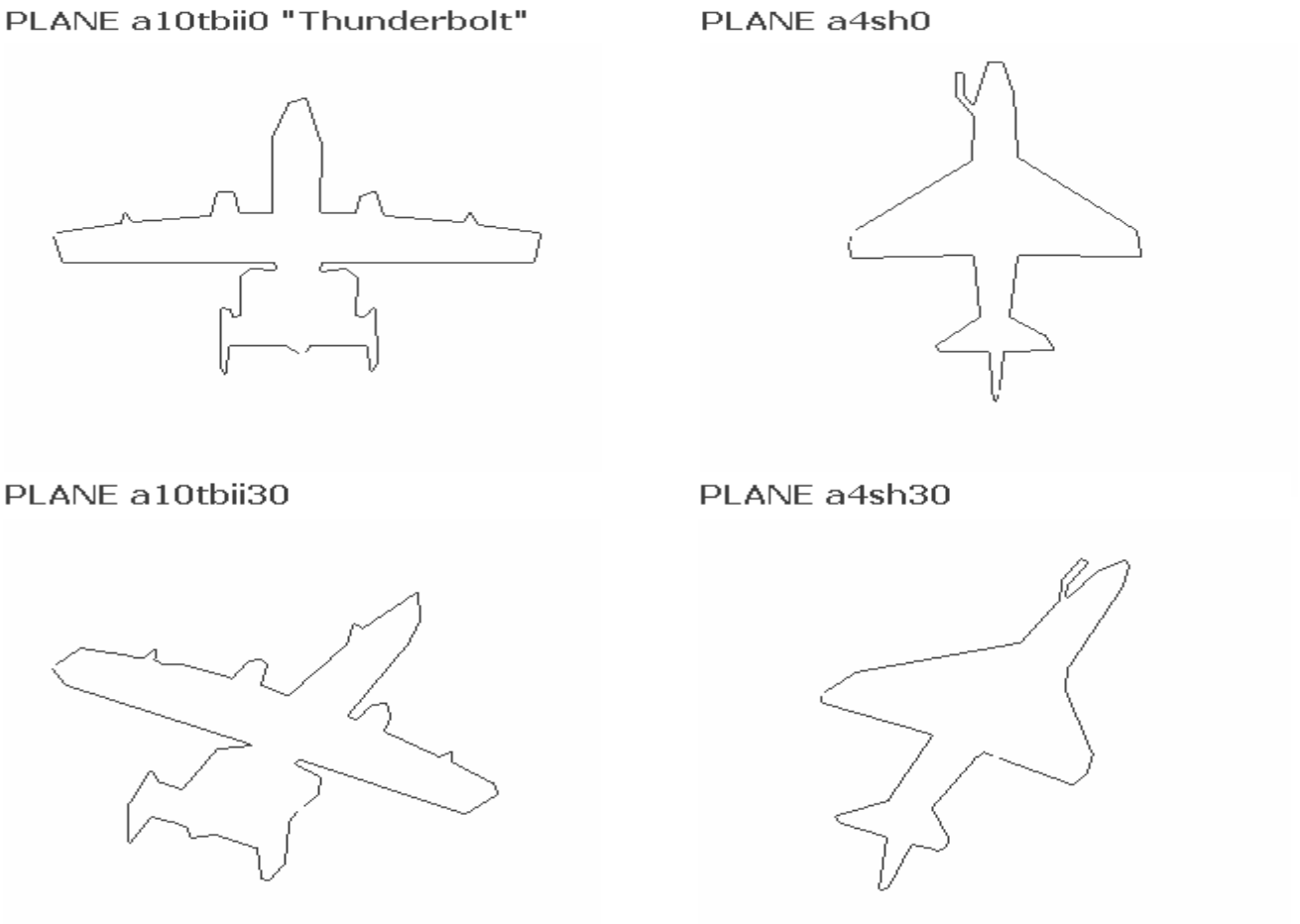

Figure 9, Example aircraft outlines

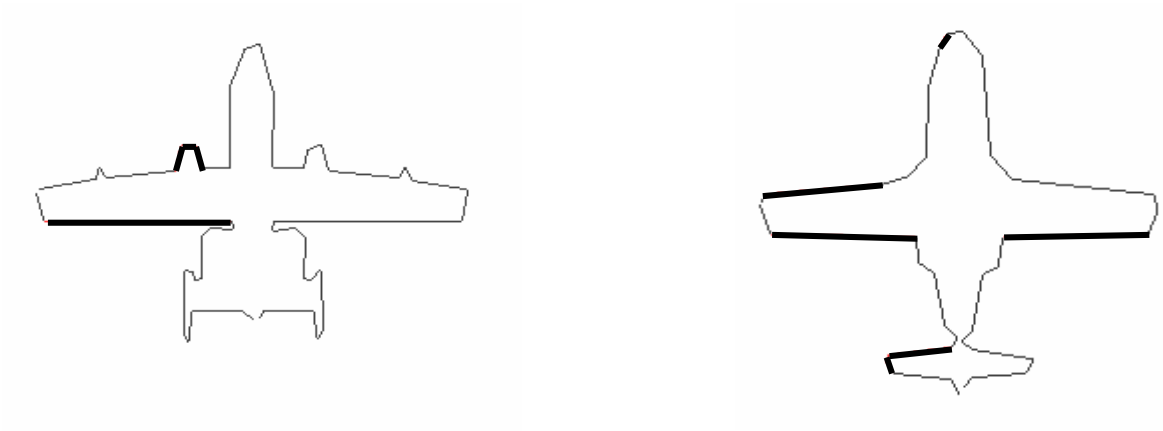

Figure 10, Recognised segments, A10 Thunderbolt and SAAB 105