TR-IT-0002

Analysis, generation and more by means of genetic algorithms

Yves Lepage

June 1993

Abstract

This report describes an experiment of applying genetic algorithms to example-based machine translation. A very simple introduction to genetic algorithms is given. A possible application of this technique to analysis and generation is proposed. It is based on the *board* data structure, which is the association of a text and a structure. This data structure allows *non-directionality*, an original property which is kept in this experiment. If part of a text and part of a structure are provided, possible completions can be filled in. Some assessment measures are given.

Keywords

Example-based translation, genetic algorithms, non-directionality.

Contents

List of Figures

Introduction

One of the differences between the example-based and the rule-based approaches in natural language processing (NLP) is the type of engine used. Whereas in rule-based systems it is generally a context-free parser, various sorts of engine can be used for the example-based approach. This report shows that an engine based on an optimisation technique, genetic algorithms, can perform analysis and generation.

Two things make this possible:

- the definition of a data structure, called *board* after the proposal in [Vauquois & Chappuy 85] and which is in essence bidirectional. It is the association of a string (for example, a sentence) and a tree (a linguistic structure). This data structure is thus ready to be interpreted for analysis (from string to tree) and generation (from tree to string).

- the definition of a distance (strictly speaking, a proximity score) on this data structure, which may contain variables. Analysis and generation can then be defined as optimisation problems. For example, an analysis result is the closest object to the board formed by the input sentence and an initially unspecified or variable tree.

This report first gives a very simplified view of genetic algorithms. It then shows which objects and functions were implemented to provide a simple genetic algorithm tool. The most important function, fitness and crossover, are detailed. Precisely how analysis, generation and a third interesting operation can be defined completes the view of the engine.

Experiments were conducted on data drawn from the ATR dialogue corpora. These data had to be transformed to fit the purpose of these experiments.

The quality of the results obtained during the experiments for analysis, generation and non-directionality are given in charts. Also, runtime measures have been carried out.

The advantages and the flaws of this work are considered in conclusion.

1 Genetic algorithms

Genetic algorithms are a collection of techniques for approaching the solution of optimisation problems. These techniques are only heuristics: one cannot be sure of reaching the solution, if it exists. But these techniques have been shown to have good convergence results for some problems [Goldberg 89].

The term *genetic algorithm* originates in a comparison with population evolution and the idea that populations might optimise some function on their individuals[1].

1.1 Principles

In this section we give very basic notions about genetic algorithms in their simplest form. For a better presentation, see [de Garis 91].

Population Usual programming techniques handle only one object at a time. By contrast, genetic algorithms deal with a collection of individuals, called a *population.*

For each individual in a population, one can compute a function, called the *fitness function.* Those individuals for which the fitness function is optimum, are the *best individuals.*

Reproduction From two individuals, one can derive two new individuals by cutting them into two pieces and gluing the pieces back in the way illustrated in Figure 1. This process is called *crossover*[2].

Generation On a population, the previous operation can be repeated until the number of children equals the number of parents. In this way, one derives one population from another, and this operation can be repeated a number of times. In the last generation, the best individuals hopefully are solutions to the optimisation problem at hand.

---

[1] Like other advertising terms in computer science, from "artificial brains" in the fifties, to "neural networks" today, the term "genetic algorithm", drawn from a daring analogy, has some ideological implications I do not share at all. Unfortunately, it is the term in usage.

[2] In the complete model, some random modification of part of the children may occur. This accounts for *mutation* to complete the genetic metaphor.

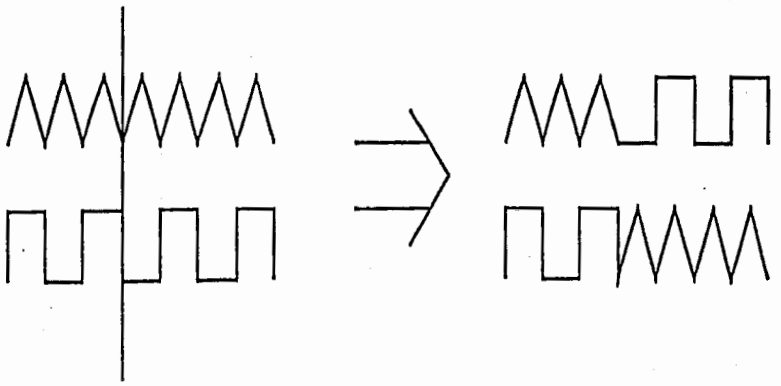


Figure 1: Principle of crossover

1.2 Realisation

In the previous description of genetic algorithm, some crucial issues are raised:

- How to select individuals to be parents?
- Where to cut when performing crossover?
- How to determine the number of generations?

One can imagine many possible answers to these questions. Thus the diversity of genetic algorithms. Below, we will describe how we answered these questions in the present experiment.

Selection In order to have good convergence results, it seems necessary that those individuals which have higher fitness value should intervene more frequently in the production of the next generation.

Selection for crossover is done randomly, but an individual has a higher probability of being selected as a parent if its fitness has a higher value. The probability of individual i, given its fitness f_i, is defined as:

$$p_i = \frac{f_i}{\sum_i f_i}$$

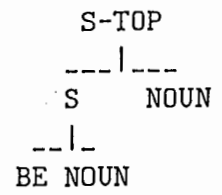
Crossover For simplicity, the place in the data structure where crossover occurs is determined at random. The way individuals are cut will be explained in the following section.

Halting This experiment was designed to see whether genetic algorithms were of some interest for our purpose. Hence, we tested precisely the number of generations necessary to get good results.

2 Implementation

2.1 The toolbox

We developed the program rapidly because a toolbox for the primitive objects had been implemented before (see [Lepage 92b]). It provides various objects, among which the *board* data structure is the most important one for this experiment. A board is the association of a tree and a text, as illustrated in Figure 2.



"is this the conference office ?"

Figure 2: A board

The toolbox has been implemented following an "object-oriented" approach. Hence, an object type is defined by a particular structure and a particular set of functions. For each object type, there usually are functions for creating, deleting, copying, reading and writing.

For the present experiment, two new object types have been created using the same methodology. The first one is *population*, the second one is *individual*.

2.2 The population object

2.2.1 Structure

A population is a list of individuals (see [Lepage 92b] for the list object type).

2.2.2 Functions

Input/output

input a population is read as a list of individuals separated by spaces;

output a population is written as a list of individuals separated by spaces.

Performance

fitness the fitness of a whole population is defined as the sum of the fitness of all individuals in the population;

best yields the first best individual found in the population.

Genetic algorithm proper

selection the selection function randomly selects an individual in the population to be a parent. An individual is selected with a probability defined according to the formula given in Section 1.2;

copulation [3] produces two new individuals from two parent individuals. This function calls the *crossover* function on individuals;

generation builds a new population from a previous one, by a series of calls to the previous function. The new population has the same size as the previous one;

evolution calls the previous function as many times as said in the arguments.

[3] Looking for a word ending with "tion", I could not resist it. I apologise.

2.3 The individual object

2.3.1 Structure

An individual consists of two parts.

- the first part pertains to the role of the individual in the population. It contains

  parents an individual points to the parents it comes from;

  crossover point the place where crossover took place when the individual was produced is remembered;

  fitness value the fitness of the individual.

- the second part is the internal structure of the individual. For our experiment, it is a *board*, *i.e.* the association of a tree and a string.

2.3.2 Functions

Input/output

input an individual is read according to the input function of the data structure it encapsulates;

output an individual is written according to the output function of the data structure it encapsulates;

copy this function copies an individual;

Genetic algorithm proper

The following functions are defined independently of the internal structure of an individual.

fitness computes the fitness value of a given individual. See below for a definition of this function in the present experiment.

inherit creates a new individual. Its parents are given as arguments. The individual created is half valid as it waits for crossover with another individual.

crossover for two individuals, randomly draws a crossover point, then cuts and cross-glues back the individuals. See below for a description of this function in the present experiment.

genealogy prints the tree of the ancestors of a given individual. It is for trace purposes.

2.3.3 The fitness function

In the present experiment, the fitness of an individual is defined as the distance to a goal. The goal is an individual given as input to the program.

As an individual is a board, its fitness is defined through the board distance, which is defined as the sum of the distances between the strings, on one hand, and between the trees, on the other hand, which are contained in the two boards (see Figure 3).

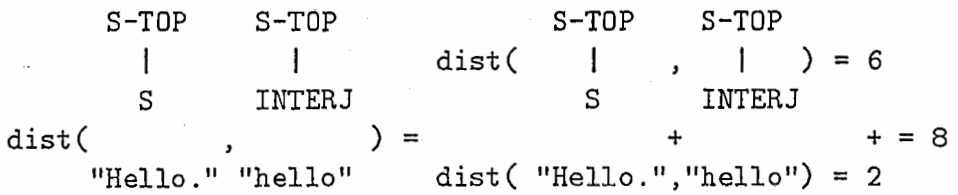


Figure 3: Distance between two boards

The string distance is the Wagner and Fischer distance generalised to string patterns (strings with variables). The tree distance is the Selkow distance generalised to tree patterns (trees with variables). In fact, these distances are implemented as a single subroutine working on a single generalised data structure called *woods*, or rather wood patterns. Is has been shown that the generalisations of these distances on wood patterns is not a mathematical distance, strictly speaking. We call it a *proximity score* (see [Lepage et al. 92] for more details). With this proximity score, the results shown in Figures 4 and 5 are obtained.

2.3.4 Crossover

First let us recall that in our implementation, all objects are factored by their end. For example, consider the trees in Figure 6. They are factored in the way shown in Figure 7. This factorisation avoids the creation of unnecessary copies.

When crossover is performed on two individuals, the cutting point is selected at random. The crossover point is an integer which refers to

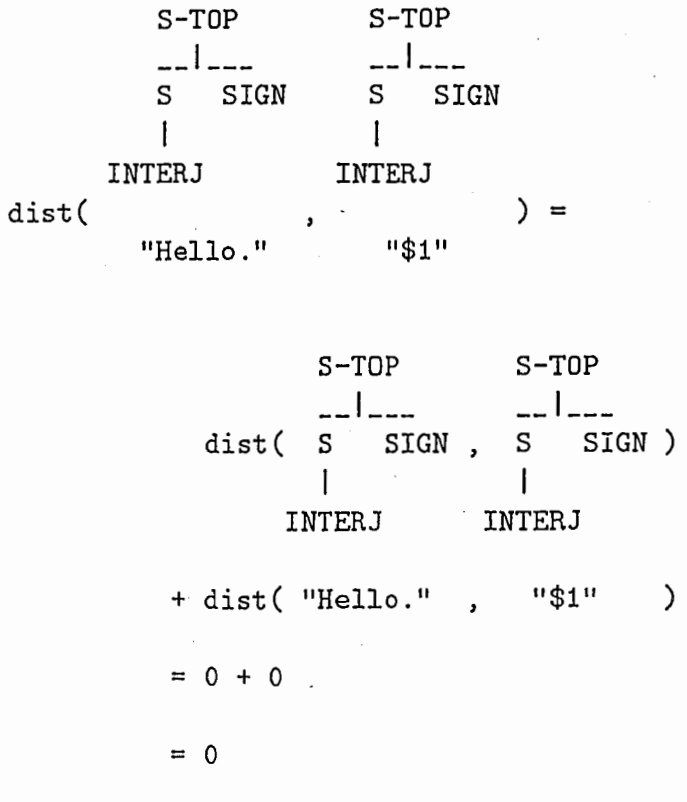


Figure 4: Proximity score with string part unknown

a position in the structure (list or tree) when traversing the structure in preorder. A copy of each parent is made until the crossover point is reached, and then the pointers to the rest of the structures are simply exchanged. This performs the crossover.

The effect of crossover on lists is simple to understand. On trees, its effect is to exchange the sister forests after the crossover point, as illustrated in Figure 8. This technique is quite different from the exchange of the subtrees described in [Koza 92].

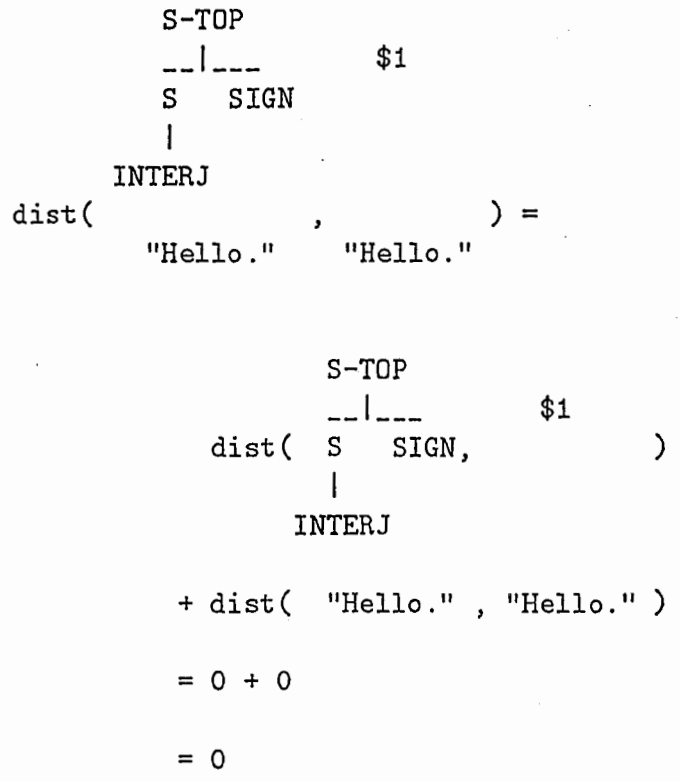


Figure 5: Proximity score with tree part unknown

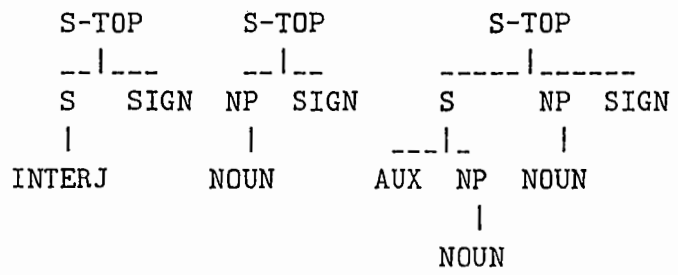


Figure 6: Three trees

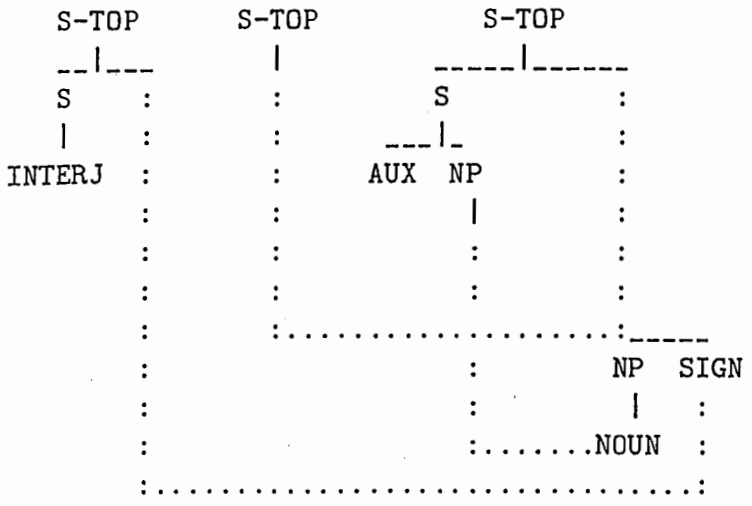


Figure 7: The trees factored

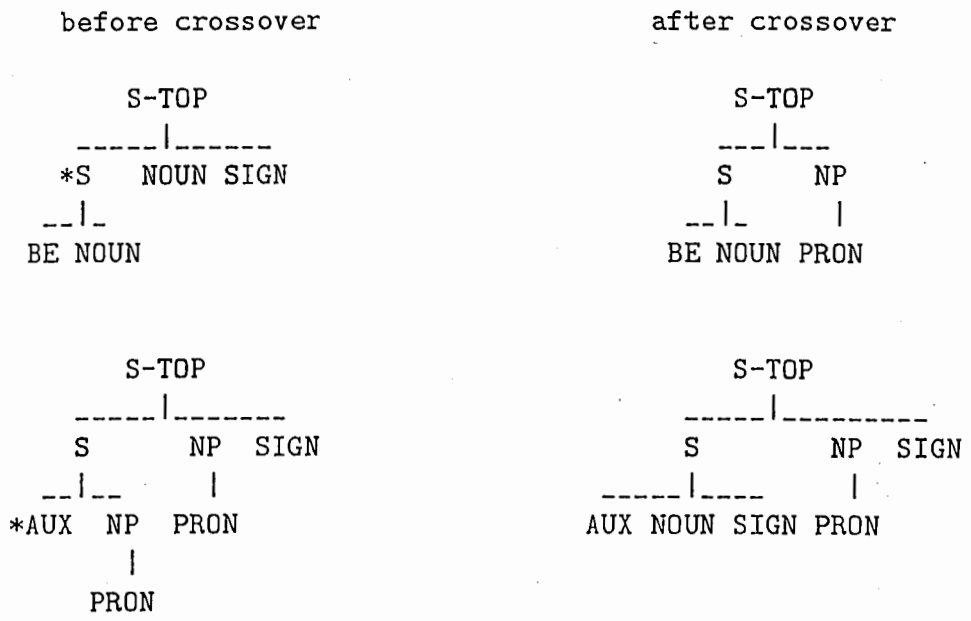


Figure 8: Crossover on trees (crossover points are marked by a *)

3 The program

This section describes the application program in natural language processing. We show how analysis and generation are performed, and also another kind of operation, called non-directional completion. In fact, analysis and generation are only particular cases of this third operation.

The basic object is a board, *i.e.* a pair of a pattern string and a pattern tree. The program works with a database of examples, that is a set of boards each of which associates a syntactic tree with a sentence.

The input to the program is a board. The program, running according to the genetic algorithm principles we described above, attempts to build a new board from pieces in the data base in order to output a board whose distance to the input is as small as possible.

3.1 Analysis

If the board given as a goal consists of a sentence with an undefined tree, that is, a variable, the effect of the program is to build a tree corresponding to the given sentence. This is *analysis*. This comes from the fact that a tree reduced to a variable has a distance equal to zero to any tree. As crossover works both on the string part and on the tree part, the string output from the program may happen to be slightly different from the input string. Figure 9 shows an example of analysis.

3.2 Generation

If the board given as a goal consists of a tree with an undefined string, that is a variable, the effect of the program is to build the sentence corresponding to the given sentence. This is *generation*. The reason is similar to the one given for analysis: a string reduced to a variable has a distance equal to zero to any other string. Again, the output tree may be slightly different from the input tree. Figure 10 shows an example of generation.

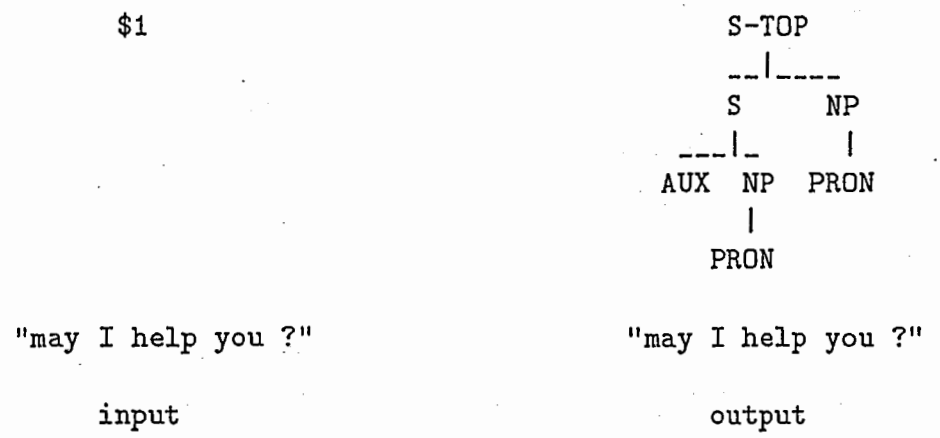


Figure 9: Analysis

3.3 Non-directionality

An important property we wanted for the system is the one described in [Lepage 91] as *non-directionality*. This is the possibility of giving a partially defined tree and a partially defined string as input. The job of the program is to build a complete association from an incompletely specified one. This operation is original and, to our knowledge, it has never been described for natural language processing. It is illustrated in Figure 11.

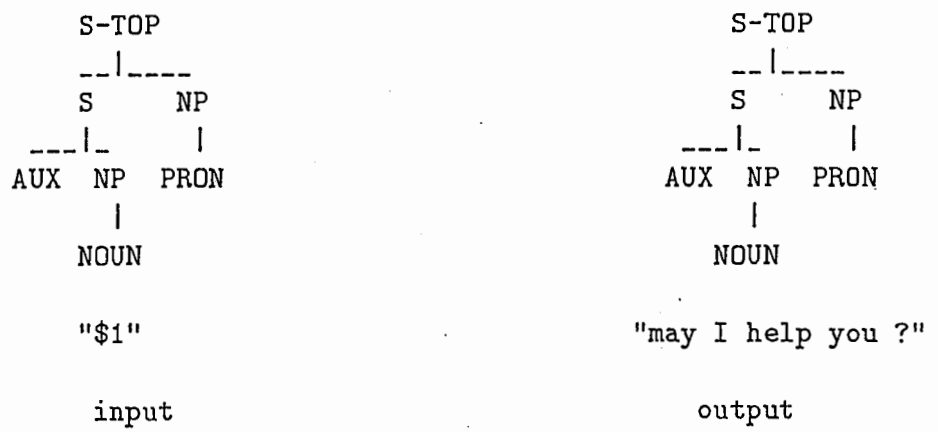


Figure 10: Generation

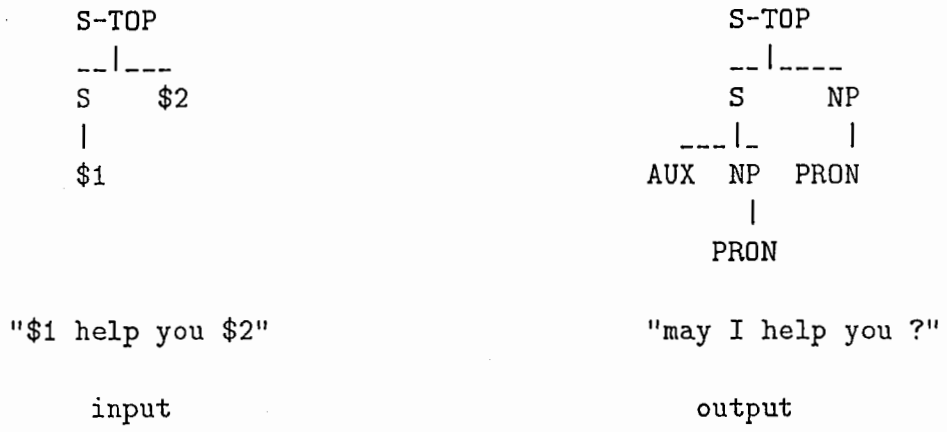


Figure 11: Non-directionality

4 The data

4.1 Nature of the data

Our experiments require that linguistic descriptions (syntactic structures) be provided for input strings. We used the results produced by the ASURA system. However, if the structures were created by hand, a linguist would have to draw a syntactic tree for each sentence of the data base. This may appear to be a disadvantage in comparison with example-based translation systems, since it is often assumed that such systems require only input strings, and themselves do all the linguistic structure-building work. But this conception is incorrect. In TDMT [Furuse and Iida 92b] for example, a thorough inspection of a corpus allowed the extraction, the collection and the abstraction of the relevant patterns in order to build the linguistic core of the system. This task *is* a linguistic task.

4.2 Provenance of the data

The data consist of 116 boards, which come from the English analysis of dialogues A-B, 1-10 of the ATR corpus. Figure 12 shows such a board.

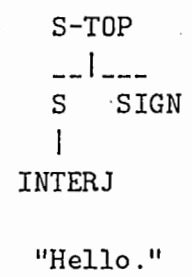


Figure 12: A board

In this database, the average length of a sentence is 24.8 characters and the average weight of a tree is 29.2 characters.

4.3 Extraction of the data

The English analysis of the ASURA system delivers a feature structure for each sentence. This feature structures encapsulate three kinds of information: syntactic, semantic and pragmatic. The syntactic part is the one which interests us. Unfortunately, HPSG representations do not deliver explicit syntactic trees; they encode them in feature structures using the following conventions: a mother node is put under its natural place as leftmost node and is dominated by an M label. During this operation, all its daughter become its sisters and they are dominated by nodes numbered D1, D2, ..., in sequence.

Because of this unnatural representation, one needs a program to carry out the transformation illustrated in Figure 13.

This program takes the form of five tree-transformation rules listed in Figure 14. For the sake of comprehension, variables beginning with a dollar sign are forest variables, those with a colon are node variables. These rules are applied in unique mode (*i.e.* not recursively) in a postorder traversal.

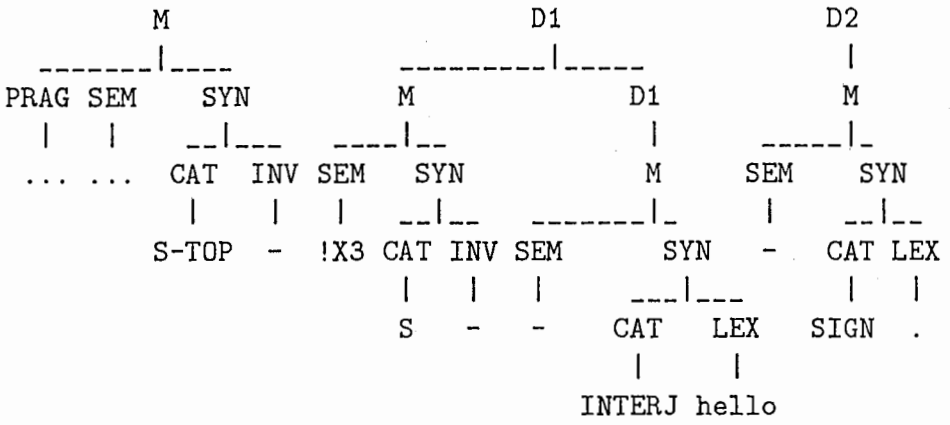
The tree-transformational program works as follows:

- relabel all D1, D2, ... as D. To do this, first relabel D1 as D, and then, recursively, all successors of a D node as D.
- reduce a feature substructure to the CAT value it contains.
- elevate the mother node to its natural position.
- erase all remaining D nodes.

a feature structure:

```
 [[M [[PRAG ...]
     [SEM ...]
     [SYN [[CAT S-TOP]
           [INV -]]]]]
 [D1 [[M [[SEM !X3]
          [SYN [[CAT S]
                [INV -]]]]]
      [D1 [[M [[SEM -]
               [SYN [[CAT INTERJ]
                     [LEX hello]]]]]]]]
 [D2 [[M [[SEM -]
          [SYN [[CAT SIGN]
                [LEX '.']]]]]]]]
```

viewed as a forest:



reduced to its syntactic part:

Figure 13: From HPSG feature structure to syntactic tree

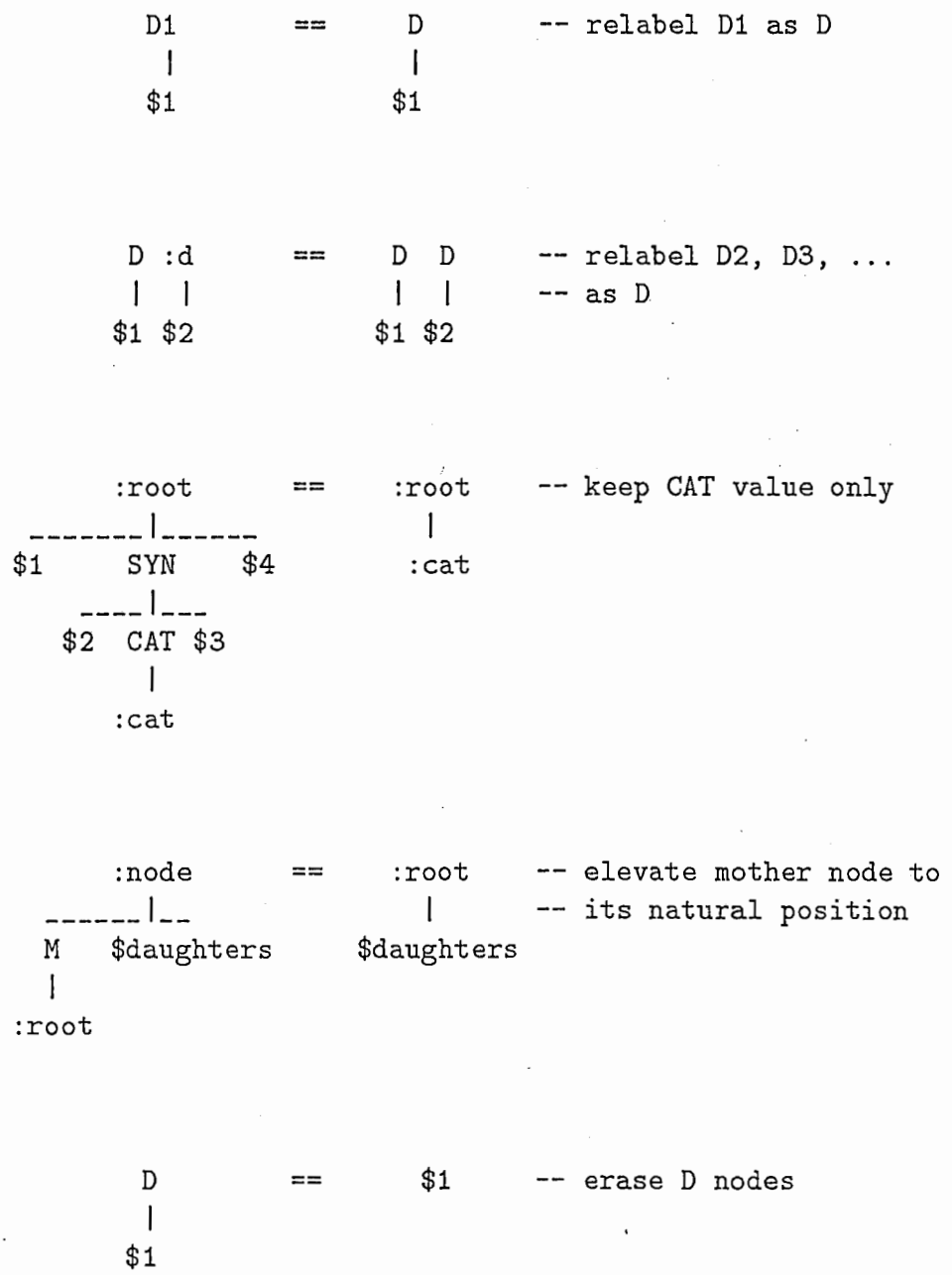


Figure 14: Tree-transformational program for the extraction of the syntactic part from a feature structure

5 Experiments

5.1 A trace

Here is a very simple trace of the program. Figure 15 draws the genealogy of the analysis result for the sentence "*Hello*". The output is the right tree for the normalised sentence "*hello .*" The trace is to be read in the following way: each line is a board, a parenthesised syntactic structure with a sentence. The output board appears vertically between its parent-boards which are shifted to the right by one tabulation. This trace is a genealogic tree with an particular individual as a stem (leftmost board) and its parents, grand-parents, and so on, as branches (above and below).

```
Number of generations: 3
A goal individual:     $1 "Hello"
A population:          #include "base"

Best:  S-TOP(S(INTERJ),SIGN) "hello ."

                              S-TOP(S(INTERJ),SIGN) "hello ."
                       S-TOP(S(INTERJ),SIGN) "hello ."
                              S-TOP(S(ADV),SIGN) "yes ."
               S-TOP(S(INTERJ),SIGN) "hello ."
                              S-TOP(S(ADV),SIGN) "yes ."
                       S-TOP(S(ADV),SIGN) "yes ."
                              S-TOP(S(ADV),SIGN) "yes ."
 S-TOP(S(INTERJ),SIGN) "hello ."
                              S-TOP(S(INTERJ),SIGN) "hello ."
                       S-TOP(S(INTERJ),SIGN) "hello ."
                              S-TOP(S(ADV),SIGN) "yes ."
               S-TOP(S(INTERJ),SIGN) "hello ."
                              S-TOP(S(NP(PRON),VP(VERB)),VP(VERB))
                              "I see ."
                       S-TOP(S(NP(PRON),VP(VERB)),VP(VERB)) "I ."
                              S-TOP(S(ADV),SIGN) "yes ."
```

Figure 15: A trace for analysis

5.2 Analysis

We measured the quality of analysis on a data base of 116 elements on sixty elements. For each board in the database, the string part only was retained and given as input to the program with an undefined associated tree. Every 3 generations, the best board built in the population is compared with the original one by applying the board distance. This yields a measure of the quality of the best board (a difference of one means a difference of one character in the string or in the syntactic tree).

Figure 16 shows the results.

- In abscissa are the generation steps (30 generations in total);
- in ordinate, the sentences of the database are ranked from the shortest to the longest one;
- in the vertical dimension, the quality of the best board is given.

5.3 Generation

Similarly to analysis, Figure 17 gives the quality of the results obtained.

- In abscissa are the generation steps (30 generations in total);
- in ordinate, the syntactic trees of the database are ranked from the smallest to the biggest one;
- in the vertical dimension, the quality of the best board is given.

5.4 Non-directionality

For non-directionality, we built manually a base of partially undefined boards from the database used in the previous experiment. Figure 18 shows the results obtained.

- In abscissa are the generation steps (30 generations in total);
- in ordinate, the partially defined boards are ranked from the smallest to the biggest one;
- in the vertical dimension, the quality of the best board is given.

5.5 Runtime

The main criticism of the genetic algorithm is that it consumes excessive time. To give an idea of how slow our program is, we made some measurements. Figure 19 shows the runtime measures for analysis on the data base of 116 elements. Figure 20 is for generation.

- In abscissa are generation steps (30 generations in total);
- in ordinate, the number of the sentence (resp. the syntactic tree) ranked by size;
- in the vertical dimension, the time, in seconds, needed to produce a result.

These figures show that the performance of the genetic algorithm run on a serial machine (a SPARC workstation, 96 Mips) are unacceptable.

However, we think that the time argument is not a valid one, when considering that genetic algorithms are in essence parallel, and that they are supposed to be implemented on parallel architectures.

The simplest design would be to assign one processor per individual in a generation. Then, running selection and crossover in parallel, the overall run time is simply proportional to crossover and the number of generations. We think that this kind of architecture, which is in fact the natural one for this kind of algorithm, would make the run times acceptable.

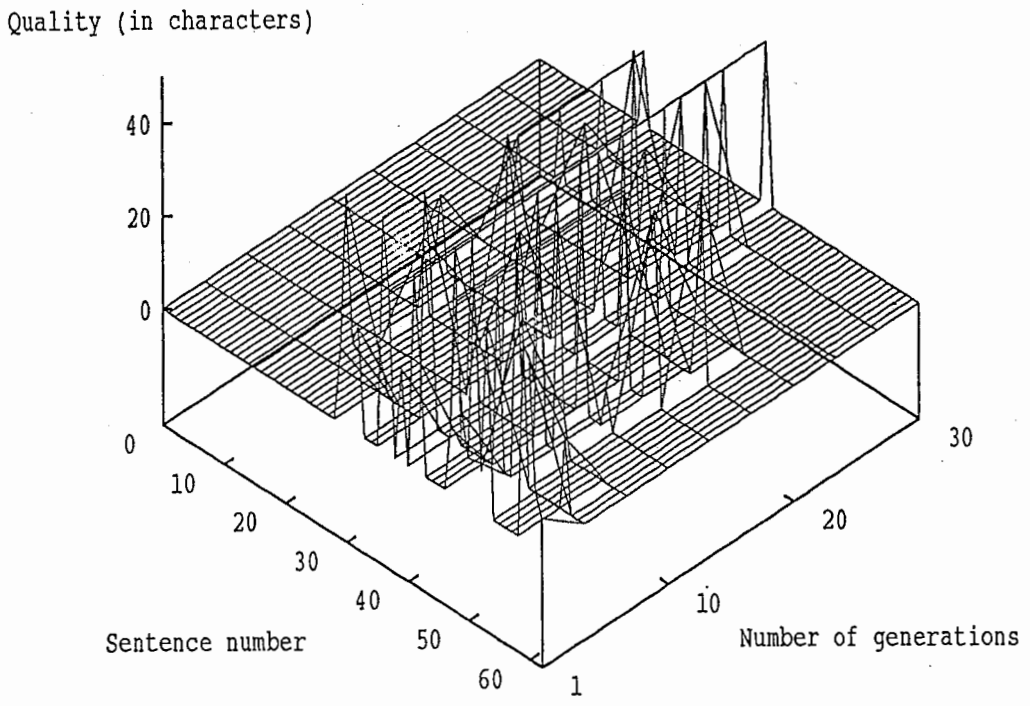


Figure 16: Analysis results

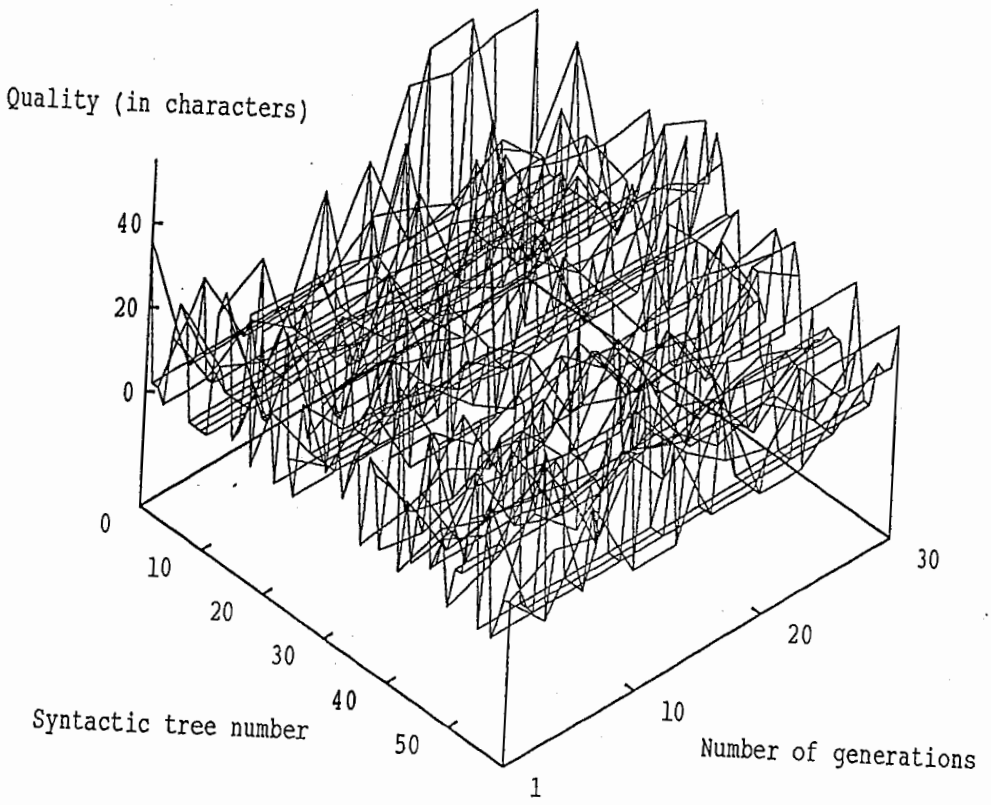


Figure 17: Generation results

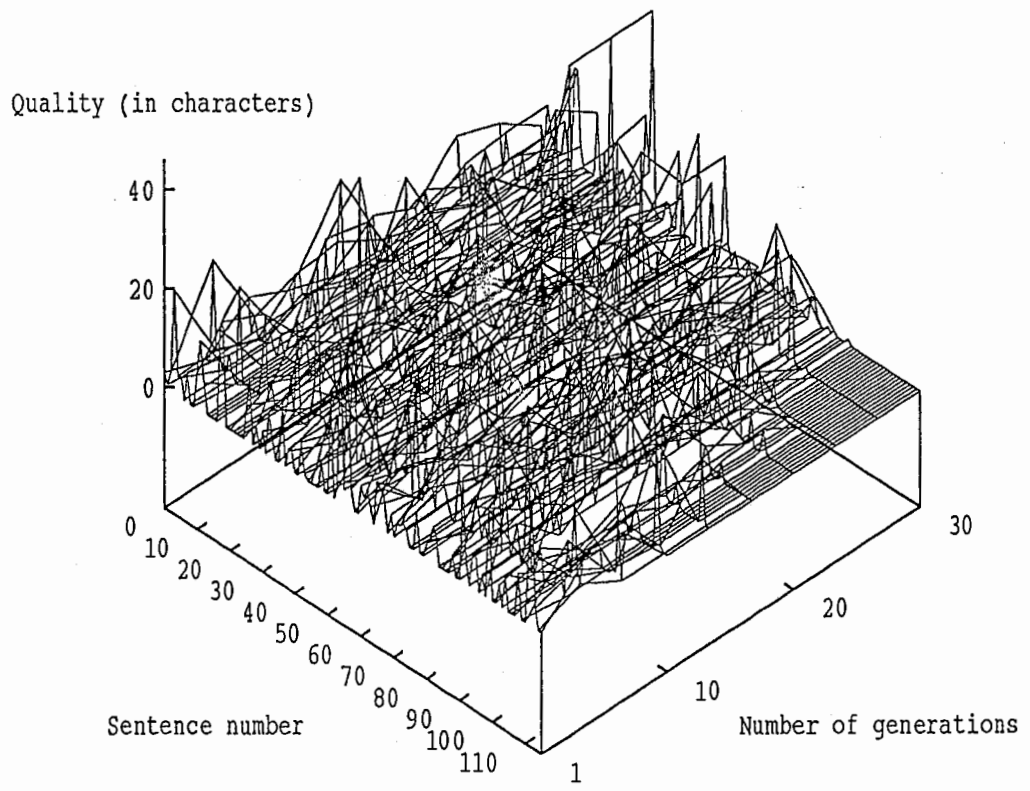


Figure 18: Results for non-directionality

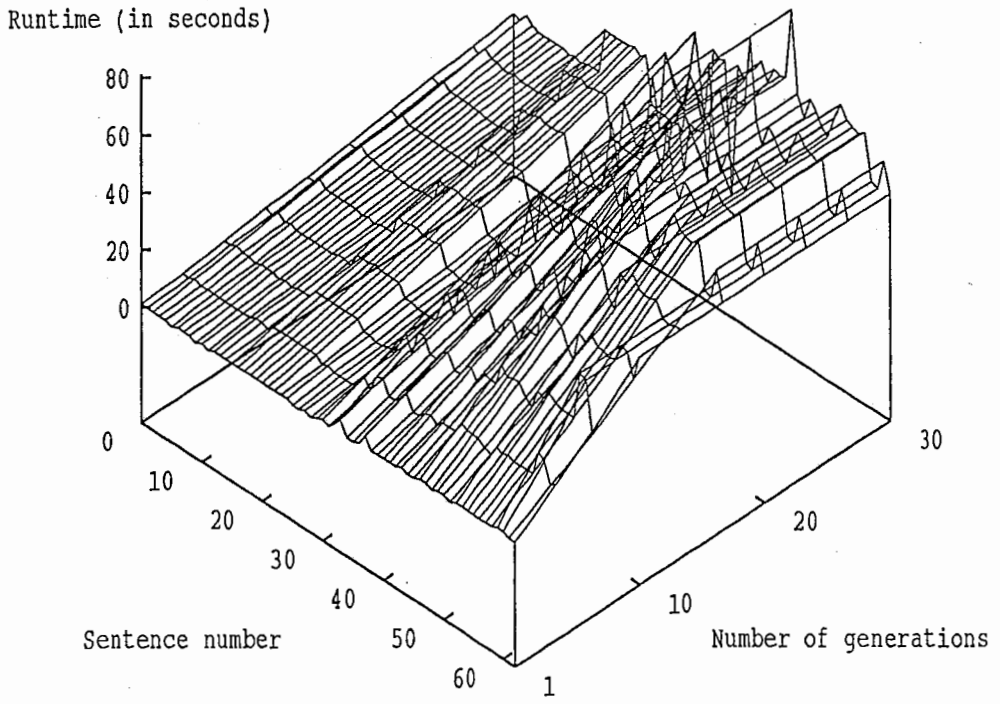


Figure 19: Runtime values for analysis

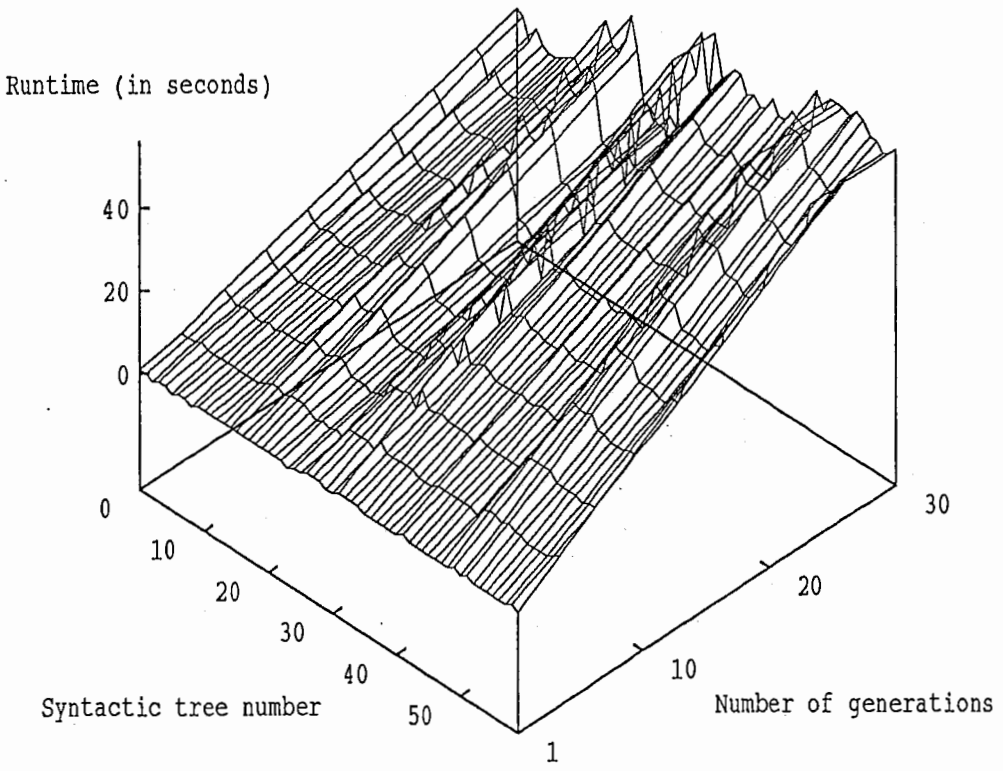


Figure 20: Runtime values for generation

6 Analysis of the results

Below we will discuss the results obtained for analysis and generation and leave aside those for non-directional completion.

6.1 Quality

Analysis Roughly, about thirty generations are needed to produce an almost correct result in analysis. The following array shows the average quality and the average runtime every third generation.

generation	quality	time
1	8.98	1.81
4	4.30	6.79
7	2.16	12.16
10	3.08	17.02
13	3.68	21.86
16	2.14	26.46
19	1.71	29.49
22	2.03	32.32
25	0.81	34.41
28	0.81	34.41

Generation The generation quality results seem rather disorderly and poor, as shown in the following table. They stabilise at a ten character difference in average. The average weight of an output is 55 characters (about 25 characters for a sentence and approximately 30 characters for a tree). Since about 11 characters are wrong in the average output, we have an average error of 1/5.

generation	quality	time
1	15.60	1.12
4	12.23	5.13
7	11.87	9.10
10	10.60	13.22
13	10.75	17.32
16	13.85	21.72
19	11.75	26.25
22	10.97	30.88
25	11.45	35.15
28	11.07	38.92

However, in fact, further inspection of the results helps to explain these disappointing scores. Two phenomena seem to be involved here.

Firstly, nearly all syntactic trees in the data base correspond to more than one different sentence. For example `"yes."`, `"no."` on one hand, or `"I am with Ken Brown."`, `"I attend with my wife."` on another hand, have the same syntactic trees respectively.

The apparently bad results are due to the assessment of such boards. From the data base, one board is extracted, `S-TOP(S(ADV),SIGN) "yes ."` for instance. The tree part only is retained and the following board is taken as input to the program: `S-TOP(S(ADV),SIGN) "$1"`. The output may be `S-TOP(S(ADV),SIGN) "no ."` which is a perfectly valid result, since both sentences have the same syntactic trees associated with them in the database. But assessment is done by comparing the output to the original board. Hence, for the current example, the quality is estimated `dist("yes .", "no .")`, *i.e.* 3, although it should have been zero.

The second phenomenon is that the program seems to hesitate between possible results. It delivers outputs like: `"I is the conference office ."`, which is a mixture of two valid sentences.

These two phenomena explain mostly the noisy shape of the quality results for generation.

6.2 Runtime

Generation is slightly slower than analysis. Generation takes 39 seconds in average for 30 generations, whereas analysis needs only 35 sec-

onds.

We suspect that this difference is due to the ambiguity of trees in relation to sentences. Another explanation might be the relative simplicity of trees in the data base and their resemblance. In other words, the trees might not be distinct enough.

Conclusion

This report has shown the application of an optimisation technique to natural language processing tasks, *i.e* analysis and generation. This technique offer some advantages.

Non-directionality The engine builds a complete sentence and its complete associated syntactic tree from a partially specified sentence and a partially specified tree. Analysis and generation turn out to be only particular cases of this general operation.

Robustness The main flaw of rule-based systems using context-free parsers is that they often fail to deliver a solution for trivial reasons such as a word missing in a dictionary. In contrast, the system described here ensures an answer in any case.

Evaluation Natural language processing lacks methods to assess its results. The introduction of distance calculations is a step toward evaluation. Our system self-evaluates itself when delivering a result: this is the fitness of the output in the genetic parlance.

Some criticisms can still be addressed to the current technique.

Granularity When performing crossover, the system does not establish any link between the string side and the tree side. It would be better to know which part of the string corresponds to which part of the tree, and to cut only according to these correspondences. Crossover would then be given a linguistic meaning.

Normalisation Experiments carried out with input sentences from outside the data base have shown that the system has a "normalising" effect. Outputs are cast to resemble sentences and trees from the database. This can be seen as a desirable effect if looking for normalisation, or as a negative effect if a free-input system is wanted.

As this work is only a first experiment, open questions still remain. They have been only skimmed over when experimenting with the system. They range from the use of dictionaries in such a framework to the representativeness of an example data base and its effect on the results obtained.

References

[de Garis 91] Hugo de Garis
Genetic Programming, chap. 8
in Pr. Brank Soucek (ed) *Neural and Intelligent Systems Integration*, Wiley , 1991.

[Furuse and Iida 92b] Furuse Osamu and Iida Hitoshi
An Example-based Method for Transfer-driven Machine Translation
Proceedings of the fourth International Conference on Theoretical and Methodological Issues in Machine Translation TMI-92, pp 139-150, Montréal, 1992.

[Goldberg 89] David E. Goldberg
Genetic Algorithms in Search, Optimization, and Machine Learning
Addison Wesley Publishing Company, 1989.

[Koza 92] John R. Koza
Genetic Programming - On the Programming of Computers by Means of Natural Selection
MIT Press, 1992.

[Lepage 91] Yves Lepage
Parsing and Generating Context-Sensitive Languages with Correspondence Identification Grammars
Proceedings of the Pacific Rim Symposium on Natural Language Processing, Singapore, November 1991, pp. 256-263.

[Lepage et al. 92] Yves Lepage, Furuse Osamu and Iida Hitoshi
Relation between a pattern-matching operation and a distance:
On the path to reconcile two approaches in Natural Language Proceessing
Proceedings of the First Singapore International Conference on Intelligent Systems, Singapore, November 1992, pp. 513-518.

[Lepage 92b] Yves Lepage
Easier C programming
Some useful objects
ATR report TR-I-0294, Kyoto, November 1992.

[Lepage 92c] Yves Lepage
Easier C programming
Dynamic programming
ATR report TR-I-0295, Kyoto, November 1992.

[Nagao 84] Nagao Makoto
A framework of a Mechanical Translation between Japanese and English by Analogy Principle
in *Artificial intelligence and Human Intelligence*,
Elithorn A. and Banerji R. eds., Elsevier Science Publishers, 1984.

[Selkow 77] Stanley M. Selkow
The Tree-to-Tree Editing Problem
Information Processing Letters, Vol. 6, No. 6, December 1977, pp. 184-186.

[Vauquois & Chappuy 85] Bernard Vauquois and Sylviane Chappuy
Static grammars: a formalism for the description of linguistic models
Proceedings of the Conference on Theoretical and Methodological Issues in Machine Translation, Colgate University, pp 298-322, Hamilton, New York, August 1985.

[Wagner & Fischer 74] Robert A. Wagner and Michael J. Fischer
The String-to-String Correction Problem
Journal for the Association of Computing Machinery, Vol. 21, No. 1, January 1974, pp. 168-173.