Genetic Logic Programming

Thomas R Osborn, Adib Charif, Ricardo Lamas and Eugene Dubossarsky

School of Computing Sciences
University of Technology, Sydney
PO Box 123
BROADWAY 2007
AUSTRALIA
email: osborn@socs.uts.edu.au

Abstract: Genetic Logic Programming (GLP) is a new method which applies the Genetic Algorithms paradigm to Declarative Programming - specifically to evolve populations of Prolog programs. This paper examines GLP applied to Natural Language Understanding to illustrate the power, issues and limitations of GLP.

Populations of Prolog query interpreters evolve to respond more correctly to queries about the Aesop fable: "The Fox and the Crow". The interpreters process parsed text and consult a general knowledge-base. The gene pool consists of a large set of Prolog rules and facts which are tentatively proposed as being 'useful' for interpretation. Essentially, interpreters act as an interface between queries, knowledge bases and the text. Closure and termination are addressed at the level of design of the gene pool, and various Prolog options.

Fitness amounts to a score on a high school like "comprehension test", with special care to deal with redundant and dependent answers, and with an eye to rewarding correct higher-level abstractions.

1. Introduction

Whereas Genetic Algorithms [4] essentially determine an optimal parametrisation of a template, Genetic Programming evolves formal programs (expression trees) to approximate a desired set of behaviours [6]. We extend the notions of both approaches to evolve declarative programs, where the denotational semantics of the genes contribute directly to solutions, rather than their operational semantics (GP) or numeric value semantics in the context of a template (GA). The conception is straight-forward, but the path is treacherous - fitness must relate to high-level abstractions about program behaviour, to reward utility and cohesiveness of the program, and gene pools must obey some restrictions to ensure termination and some senses of closure.

Prolog was chosen as a vehicle for this study, which we refer to as Genetic Logic Programming, and the domain of Natural Language Understanding (NLU) was used for evaluation. All NLU methods are inherently discrete and semantic, and have strengths and weaknesses. Some related work on dialogue optimisation was performed by Nettleton and Garigliano [7].

Our project aimed to exploit the combined behaviours of available methods while averring the undesirable behaviours. Broadly, existing methods match logical pattern through grammatical transformation of the text, supported by lexicons, frames and "exceptional cases" lists. Indeed, our study mirrors the history of natural language understanding where many efforts have evolved and combined to address more abstract and more detailed concerns. "The second reason to believe natural language understanding is entering a new phase is the development of a wide range of theories describing particular aspects of multi-sentence discourse. This includes models that apply general world knowledge about actions and communication to integrate multiple sentences into a coherent aspects of discourse itself". From "The Handbook of Artificial Intelligence" [2].

Prolog implementations of NLU consist of included files of: logical rules for grammars and parsing; logical rules for post-processing of basic parses; and, frames and facts for semantic support. We applied a traditional DCG (Definite Clause Grammar) approaches for these parsings, but broaden the variety of grammars to cover structural information about semantic values of physical, temporal, psychological states [3]. In effect, the parser's role was to supply, uninterpreted, as much relational and term information from the text as possible. Post-processing involved complex processing of the parsed text subject to constraints in the semantic support. We evolved populations of sets of post-processing rules.

An individual interpreter performs by attempting to provide answer to queries which the researchers consider germane to a "good understanding" of the text. Logic programs correspond to a natural query answering system. To score the performance, we measure the difference between the right (human adjudicated "correct" or formal) answers and the system's performance on the same questions. This corresponds in education of humans to the dreaded and obtuse "Comprehension Tests" from school. The inclusion of human opinion in the comprehension test design and also the arrangements for scoring implies that the fitness function favours behaviours which are subjectively correct, rather than exhaustively correct. Even the expertise of the researchers in designing the tests is subject to evolution through some kind of guided trial and error.

Naturally, most questions have many right answers. Some right answers are essentially synonyms of each other - the convey the same information - while others are logically correct but a long way from "interesting" or "relevant" to understanding of the meaning or significance of the text.

Furthermore, some answers from the system may be wrong. This may result as an artefact of the interpretation process, deficiencies in the knowledge-bases, information missed by the parsers and also, stochastically wrong responses from unclear or ambiguous information.

Some questions may not be answered well. Few or none of the correct answers may be found, either because the knowledge-base does not support the queries, or the interpreter's rules may not be sufficient to compute many of the answers. In any case, specifying the fitness function in terms of the information content favours a "focussed" text interpreter - focussed on the human way of taking in information from text.

2. Technical Details

Initially, we gathered models of NLU as basic fodder for the gene pool. In addition to syntactic grammars, we derived semantic grammars to cover physical and logical constraints and some psychological mechanisms (eg, motivations). Somewhat similar implementations have been devised by Sgall *et al* [8], Allen [1], Holland *et al* [5]. Decomposition of the NLU models yielded a large set of rules. These rules constitute the genes of the gene pool (1 gene = Fact, Prolog rule or set of Prolog rules). Our gene pool consisted of about 300 rules. The overall scope of the project was restricted by tailoring the genes/rules to the specific *terms* in the fable, and concepts required to support those terms. This restriction does not appear impose artificial restrictions on understanding *this* fable, and future studies on other text should probably only require augmentation of the gene pool - we shall see.

The first generation of chromosomes (bit strings) is created randomly. So a file of rules for NLU query-answering is created. For example,

```
chromosome  [0,0,1,0,1,1]  &
pool  [Rule1,2,3,4,5,6]
     ==>  interp[Rule3,5,6].
```

Fitness tests the capacity of the chromosome to answer a fix set of questions, and give a score to each chromosome. After text interpretation and fitness evaluation of all the chromosomes, a new generation is created using the highest scoring several chromosomes and (bit) mutation and cross-over. Selection probability is proportional to the fitness score.

Figure 1 illustrates the basic architecture for GLP applied to NLU.

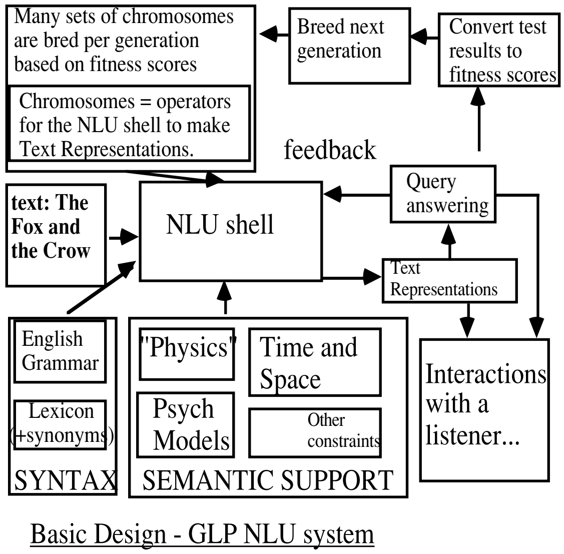

Figure 1: The Architecture of GLP for NLU

3. Semantic Grammar and Parser

The DCGs approach (Definite Clause Grammar) which allow rewriting of rules, describes a treelike representation of each sentence. The result of each parser is a compound term containing the semantics value of the text.

For example, the Semantics Analysis of the sentence "the fox walked under the branch" is:

```
sentence( agent_phrase( agent( mammal(
fox)), desc_agent( definite)), event(
move_event( walked), features( past, 3,
sing)), compl( mod_and_location(
preposition_loc( under), location( plant(
tree), feat( definite))))).
```

This semantic parsing version of DCG offers flexibility in accessing textual information, as syntactical and some low level context are made explicit *before* the GLP attempts to model the implicit semantic contents. For example, to find the location X of the fox, there should be an agent(mammal(fox)) and a location(X), related by an event (eg, a movement event).

More complicated questions require access to the world knowledge, and the lexicon and the other form of analysis - to which various genes have hooks. For example: To know *'fully'* "why the crow caw"ed in the Fable, we need to know the initial beliefs of the crow and, the fox about the crow, and about himself. We need to know the planning of the fox and what steps he took to execute his plan, and how the belief of the crow was changed which let him open the mouth to caw where the cheese fell to the ground and was snatched up by the fox.

The chaining of genes (declared in chromosomes) - gounded in the parsed text, plus semantic support - supplies the evolving test interpreters.

The *correctness* criterion for answers (however intricate) came *a priori* from human expectations of what is right or wrong - ie, a human 'teacher' devises the correct answers. However frequently the list of correct answers needed to be expanded as the system reminded us of other *rational* chains of reasoning through the application of exhaustive evaluation.

The issue of non-termination was addressed by a genetic engineering approach. All genes which could lead to non-termination (via infinite regression, or other looping behaviour) were culled from the gene pool. This was fairly easy to do, as the gene pool itself was a valid interpreter. Non-termination could have been addressed via a time-limit fitness penalty.

Table 1: Selected typical genes - Genes relating to location and physical relations between things.

```
%******* loc - genes about location *****
loc(L,Y) :-           setof(X,locat(X,Y),L).
locat(X,Y) :-         loc(X,Y,_).
loc((Y,part_of,X),Y,init_time) :-        part_of(Y,X).
loc(X,Y,T) :-         inside(X,Y,T).
%*******18 loc genes omitted
loc((Y,P,X),Y,T) :-sentence(list_clauses(
      head_clause(agent_phr(agent(mammal(Y)),_),
      _,body(mod_and_loc(prep_loc(P),loc(
      compon(X,_),_)),_),T),_,_,_),_).
loc((Y,above,X),Y,T) :-
      above(Y,X,T); above(Y,X,init_time).

%*******above **********
above0(X,Y,T) :-
      sentence(head_clause(agent_phr(agent(bird(
      X)),_),_,body(mod_and_loc(prep_loc(on),
      loc(compon(Y,_),_)),_),T),_,_).
above1(X,Y,T) :-    above0(X,Z,T), on(Z,Y,_).
above2(X,Y,T) :-    above0(X,Z,T), under(Y,Z,T).
%*******15 above genes omitted
above((L,above,Y),Y) :-      setof(X,above1(X,Y),L).
above1(X,Y) :-       above(X,Y,T).
```

Table 2: Sample of genes relating to perception.

```
%******* see ************
see0(X,Y,T) :-        sentence(_,tail_clause(_,
      head_clause(theme_phr(theme(pers_pro
      (she)),_,refer_to(agent(bird(Y),_))),event
      (prim_aux(was),_,perc_event(seen),_),body
      (prep_way(by),agent_phr(agent(mammal
      (X)),_)), T)),_).
see(X,L,T) :-  setof(Y,(see0(X,Y,T); see1(X,Y,T);
      see2(X,Y,T); see3(X,Y,T); see4(X,Y,T);
      see5(X,Y,T)),L).
```

Table 3: Sample of genes relating to physical actions

```
%********* did ************
did(X,Y,L,T) :-       sentence(list_clauses(head_clause
      (agent_phr(agent(mammal(X)),_),event
      (move_event(Y),_        ,_,T),_,_,_),_),
      place(L,X,T).
did(X,Y,L,T) :-       agent(X), event(Y,T), dir(L,X,T).
did(X,Y,L,T) :-       agent(X), event(Y,T), dir((P,A),X,T),
      loc(L,A,_).
%******* many did genes omitted
```

Table 4: Some information processing genes.

```
%*******why genes ********
why(AW,did(X,Y,_)) :-        sentence(list_clauses(_,
      _,reply_cond(tail_clause(comp_conj(Z1),
      stim_reply(prep_caus(Z2),event(perc_event
      (Z3),_,_),agent(_,_,refer_to(mammal(Z4))),_),
      comp_conj(Z5),head_clause(agent(pers_pro(
      Z6),_,_),event(mod_aux(Z7),_,speak_event(
      Z8),_),_,_)),head_clause(agent(_,_,refer_to(
      agent(bird(X),_))),_,_,_,stim_reply(_,event(
      speak_event(Y),_),_,_),_))),_),
      AW = [Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8].
why(AW,did(X,Y,_)) :-   believe(A,is(X,Z,_)),
      prop(Z,X,_),tel(A,X,Z), satisfy(Y,is(X,Z,_)).
```

Table 5: Genes relating to decisions by agents.

```
%********* agent genes********
agent(X) :-  sentence(head_clause(agent_phr
      (agent(bird(X)),_),_,_),_,_).
agent(X) :-sentence(_,tail_clause(_,head_clause
      (_,_,body(_,agent_phr(agent(mammal(X)),
      _)),_)),_).
agent(X) :-   sentence(list_clauses(_,head_clause(
      agent_phr(agent(mammal(X),_)),_,_,_),_,_),_).
%********some psychological genes***** domin(X,Y,T) :-
      believe(Y,quality(queen,X),T).
believe(X,Z,T) :-     agent(X), know(X,Z,T).
know(X,Z,T) :-        agent(X), see(X,Z,T); prop(Z,X,T).
know(X,Z,T) :-        agent(X), see(X,Y,T); prop(Z,Y,T).
satisfy(X,Y,T) :-     quality(X,Y,T), like(Y,X).
prop(Z,X,_) :-        quality(Z,X).
prop(Z,X,T) :-        sentence(head_clause(agent_phr
      (agent(bird(X)),agent_desc(_,C1,prop_list
      (shape(C2),prop_list(color(C3),_)))),_,_,_),
      _,T), Z = [C1,C2,C3].
%*******lots omitted*******
```

Table 6: Genes for miscellaneous cognitive support.

```
is4(X,(Y,P,B),T) :- sentence(list_clauses(_,_,_,
      head_clause(_,_,agent_speech(_,_,_,_,_,_,
      cond_phr(_,necess_phr(cor_conj(then),_,
      head_clause(theme_phr(_,_,refer_to(agent_phr
      (agent(bird(X)),_))),_,body(theme(title_import
      (Y),_,prep_part(P),animal(B),_)),T)))))),_).
is(X,Y,T) :- is0(X,Y,T); is1(X,Y,T); is2(X,Y,T).
imp_qual(X,Y,T) :-is0(X,Y,T).
app_qual(X,Y,T) :-    is1(X,Y,T); is2(X,Y,T).
means0(X,Y) :-      X == looks -> (app_qual(Z,Z1,_),
      Y = [Z,Z1]); Y = 'I dont know'.
means(X,L) :-    setof(C,means0(X,C),L).
```

This representative list of genes above includes example of logic interacting rules to deduce physical, grammatical or psychological facts. We can deal with *any* situation or question by adding logical rule to the genes pool. The two design criteria for genes was that the 'looked like they might have been relevant, and that non-terminated recursions were avoided by anticipation. The Prolog compiler was set to take non-instantiated terms (eg, in Quintus Prolog, this is via 'unknown_predicate_handler, or 'on_exception') in the body of a rule as failure, rather than error (abort).

Temporal reasoning is inferred from clausal structure of the text by the semantic parser. Fornunately, fables use this convention to signify occurence of events.

One example of a fitness test is :

Q1: locat(X,crow,time(1,1);
Q2: locat(X,crow,time(2,1));
Q3: locat(X,crow,time(4,3));
Q4: locat(X,fox,time(2,1));
Q5: locat(X,fox,time(4,3));
Q6: locat(X,cheese,time(4,3));
Q7: locat(X,branch,time(3,3));
Q8 believe(X,fox,time(1,2));
Q9: believe(X,fox,time(4,3));
Q10: need(X,crow,time(2,4));
Q11: plan(X,fox,time(1,2));
Q12: plan(X,fox,time(4,3)).

Most of these questions have many answers, for example, **Q2: [(crow, on, branch), (crow, on, tree), (crow, above, branch), (crow, above, ground), (crow, above, fox), ...].** The last of these answers is NOT true for Q1, since the fox position is unknown initially.

Likewise, **locat(X, cheese, time(Y,Z))** varies when the cheese falls, and the **need()** and **plan()** predicates change throughout the story. Fitness measures - how the answers are scored is a subjective judgement of how which of the answers are synonyms, and only to reward first occurrences.

Any wrong answers are penalised equally (whether synonyms or not, since their incorrectness invalidates any arguments of 'concurrent sentencing').

Presence of correct answers to a question implies that the chromosome included a valid combinations of rules which utilise the knowledge base, and the parse-tree expressions of the story.

Incorrect answers imply some kind of chainings of rules which *together* utilise some terms incorrectly. This may be because some genes are too general, or they may simply perform a wrong action.

Missing right answers may be a result of genes which are too generalised, or simply that the gene was not present on the chromosome. *Some* combinations of rules may satisfy one set of questions, but fail or answer incorrectly, some other questions.

The gene pool was revised over time, by adding more specialised (hopefully better) variants of genes which seemed to cause this kind of defect - although the accused gene was not omitted from the gene pool.

Recent experiments with GLP have attempted to mutate genes by generalising, specialising and splitting the rules which constitute the rules. In this approach, the variants are attached to the same gene index, modelling the allele concept.

4. Typical Performance

A range of experiments were performed based on several fitness tests. Figure 2 illustrates the learning curve for the system.

Convergence was quite rapid, by GA standards. This may have arisen in part from weeding out of intrinsically poor individual genes early on. The deleterious gene combinations would constitute a second phase of evolution.

A simple interpretation of this graph may suggest that one valid Logic Program was being selected, and that the lower scores arose from poorer variants.

This did not appear to be the case. Inspection of high scoring individual text interpreters showed that there were many combinations of genes (perhaps 'races') which performed well.

The problem of cross-over between races has not been investigated as yet. However, it would seem that with the multiplicity of high scoring *distinct* individua, there could be some frustration of evolution from 'blending' our search trajectories

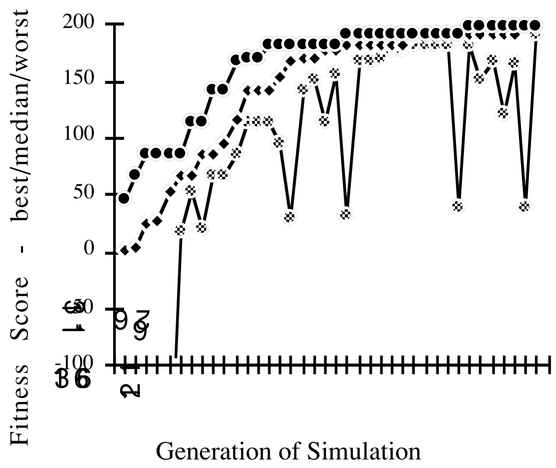

Figure 2: Learning Curve - performance on fitness over time. Default (null response) = -999. Perfect score (solutions achieve by the entire gene pool, with all incorrect responses deleted) was just under 200 - depending on the design of the fitness test).

The particular fitness test which yielded figure 2 included a great variety of types of questions - queries about location, time, information, actions, motivation, and cognition. The questions were specific, and deliberately chosen so they weren't comprehensive, in order to identify any generalisation which the system achieved.

When the later generation interpreters were interrogated with novel questions - queries not found in the gene pool, answers often appeared to be consistent with human beliefs about the fable.

This suggests at the level of semantics that there was some emergence or selection of abstraction (ie, semantic concept selection).

Genes which semantically were intermediate between questions and answers were effectively higher level semantic concepts from which a variety of questions could be answered. These concepts were not generated by the system although they may be in a system which also used the mutation paradigm outlined earlier. The concepts were selected, after having already been declared in the gene pool.

However, some experiments utilised fitness tests which were *restricted* in semantics, for example, they may have only tested knowledge about physical location and time. The later generation interpreters generally were poor at answering questions about motivation, possession, and belief. This is hardly surprising.

But with a fitness test mainly aimed at motivation, queries about location, time, and information were answered fairly well. This is consistent with the idea that spatio-temporal and basic information and perception details are necessary to answer more elaborate queries. However, these experiments took longer to converge.

In a program engineering sense, one valuable aspect of this approach to gene pool design is that you don't have to devise excellent operators - you simply devise many of them, and never throw one away (except to avoid non-termination).

5. Conclusion

GLP proved effective at evolving Logic Programs which yielded correct query responses according to a pre-determined fitness 'comprehension test'.

Subjective evaluation using unseen queries was not uniformly correct, but demonstrated a high proportion of correct responses. The extent of good answers for unseen questions appears to be consistent with logical entailment of the novel question with those queries provided in the fitness tests. Nevertheless, significant generalisation appeared to emerge in the better scoring individuals from later generations.

Currently, the design of fitness tests appears to be a crucial feature of GLP. The basic approach has been to assess the validity of a Logic Program by inspired testing.

As in Test Design in Education, the test designer needs to have a good model for the knowledge being learnt, in order for the learning system to make progress. Principles of parsimony (of concepts) and possibly other ways to introduce feed-back may be useful in future studies in GLP, especially in the domain of NLU.

Later research will attempt to broaden the base of our NLU analysis as well as applying the GLP (and other declarative Genetic Programming) as well as to other domains.

Other areas requiring attention include: elaboration of the mutation operator (to include rule specialisation, generalisation, and splitting); and, a theoretical basis for the fitness measure based on an information theoretic analysis of the underlying (Bayesian) belief networks which support correct answers to the fitness queries.

6. References

[1] Allen J A, 1987. Natural Language Understanding. Benjamin/Cummings, California.

[2] Barr A, Cohen P R and Feigenbaum E A, 1989 Handbook of Artificial Intelligence, Addison-Wesley.

[3] Charif A, 1994 Genetic Logic Programming for Natural Language Understanding, MSc thesis, University of Technology, Sydney.

[4] Grefenstette J J,1987. Genetic Algorithms and Their Applications: Proc. 2nd International Conference on Genetic Algorithms.

[5] Holland J, Holyoak K J, Nisbett R E and Thagard P R, 1987, Induction. MIT Press.

[6] Koza J R, 1993, Genetic Programming on the programming of computers by means of natural selection. The Mit Press, Cambridge MA.

[7] Nettleton DJ and Garigliano R, Dialogue Optimisation in the LOLITA Natural Language Processor using Evolutionary Algorithms. Technical Report (Durham), and Private Communication.

[8] Sgall P, Hajicova E, and Panevora J, 1986, The Meaning of The Sentence in its Semantic and Pragmatic Aspects, MIT Press.