The Halt Condition in Genetic Programming

José Neves, José Machado, Cesar Analide, António Abelha, and Luis Brito

Department of Informatics, University of Minho, Braga, Portugal
{jneves,jmac,analide,abelha,lbrito}@di.uminho.pt

Abstract. In this paper we address the role of divergence and convergence in creative processes, and argue about the need to consider them in Computational Creativity research in the Genetic or Evolutionary Programming paradigm, being one´s goal the problem of the Halt Condition in Genetic Programming. Here the candidate solutions are seen as evolutionary logic programs or theories, being the test whether a solution is optimal based on a measure of the quality-of-information carried out by those logical theories or programs. Furthermore, we present Conceptual Blending Theory as being a promising framework for implementing convergence methods within creativity programs, in terms of the logic programming framework.

Keywords: Computational Creativity, Genetic or Evolutionary Programming, Extended Logic Programming, Quality-of-Information, Conceptual Blending Theory.

1 Introduction

While the discussion around the phenomenon of creativity runs about fundamental issues like clarification of concepts, evaluation, psychological factors or philosophical questions, the quest for creativity in Artificial Intelligence (AI) has begun, raising its unavoidable subjects such as knowledge representation, search methods, domain modelling, just to name a few. In this paper, we propose a theme around the subject of modelling creativity, from the point of view of the process. We start by considering the divergence/convergence characteristics of the creative process as an argument for the need of divergent methods that, at some point, are able to detect a convergent solution as a way of goal accomplishment. Although this may seem the description of search methods in general, it is clear that we may deal with wider amplitudes of divergence in tasks that demand higher creativity. These tasks do not necessarily have to present a particular form or be of a specific kind. However, the quest for a previously unseen and correct solution is surely expected. It is a solution that traditional methods do not seem to find. We think that some qualitative jump must be made in AI, such that classical methods become more able to diverge or at least to combine with other processes, in order to enter the realms of creativity.

On the other hand, *Genetic Programming (GP)* may be seen as one of the most useful, general-purpose problem solving techniques available nowadays. It has been used to solve a wide range of problems, such as symbolic regression, data mining, optimization, and emergent behavior in biological communities. *GP* is one instance of

J. Neves, M. Santos, and J. Machado (Eds.): EPIA 2007, LNAI 4874, pp. 160–169, 2007.

the class of techniques called evolutionary algorithms, which are based on insights from the study of natural selection and evolution. Evolutionary algorithms solve problems not by explicit design and analysis, but by a process akin to natural selection. An evolutionary algorithm solves a problem by first generating a large number of random problem solvers (programs). Each problem solver is executed and rated according to a fitness metric defined by the developer. In the same way that evolution in nature results from natural selection, an evolutionary algorithm selects the best problem solvers in each generation and breeds them. Genetic programming and genetic algorithms are two different evolutionary algorithms. Genetic algorithms involve encoded strings that represent particular problem solutions. These encoded strings are run through a simulator and the best strings are mixed to form a new generation. Genetic programming, the subject of this article, follows a different approach. Instead of encoding a representation of a solution, *GP* breeds executable computer programs, here defined in terms of logic programs or logical theories.

2 The Problem

A *genetic* or *evolutionary algorithm* applies the principles of evolution found in nature to the problem of finding an optimal solution to a solver problem. In a *genetic algorithm* the problem is encoded in a series of bit strings that are manipulated by the algorithm; in an *evolutionary algorithm* the decision variables and problem functions are used directly. A drawback of any evolutionary algorithm is that a solution is *better* only in comparison to other, presently known solutions; such an algorithm actually has no concept of an *optimal solution*, or any way to test whether a solution is optimal. This also means that an evolutionary algorithm never knows for certain when to stop, aside from the length of time, or the number of iterations or candidate solutions, that one may wish to explore. In this paper it is addressed the problem of the *halt condition in genetic programming*, where the candidate solutions are seen as evolutionary logic programs or theories, being the test whether a solution is optimal based on a measure of the quality-of-information carried out by those *logical theories* or programs.

2.1 The Past

For example, let us consider the case where we had a series of data that was produced from a set of water's reservoir results taken over time. In this case we have the values that define the output of the function, and we can guess at some parameters which might inter-operate to produce these values - such as a measure of the acidity or alkalinity of the water's reservoir, the dissolved oxygen, the *nitrates*, the *phosphates*, the *chlorophyll*, and so on. We might like to predict how the water's reservoir results will fare in the future, or we may want to fill in some missing data in the series. To do so, we need to find the relationship between the parameters which will generate values as close to the observed values as possible. Therefore, our *optimum* will be fit to the observed values.

To give a very different example, consider the problem a glazier might have in deciding how best to cut up a large sheet of glass in order to achieve the minimum

wastage if a known number of different sized panes are to be produced from the sheet. The function is therefore the area of glass needed, and the parameters are the sizes of the panes required. The *optimum* is the minimum area of glass left over after all the panes have been cut out. Whereas traditional search algorithms seek to search such problem spaces in a linear fashion, one search route at a time, i.e., *Genetic Algorithms* maintain a population of candidate solutions, all competing against one another. At each iteration of the GA search engine, each candidate solution is evaluated against the optimality criteria specified, and assigned a measure of goodness. All the candidate solutions are then replaced with new candidates by a reproduction process which seeks to combine parts of one solution with parts of another. By a stochastic process, only the better candidates are selected to perform in this process. As a result, some of the good candidate solutions will be replicated, and some new candidates will be produced based on the existing ones. This forms a kind of directed search with controlled stochastic effects to provide exploration where needed.

The GA operates upon a *population* of candidate solutions to the problem. These solutions can be held in the type of their parameter representation. For example, if the candidate solutions were for a function optimisation problem in which the function took a fixed number of floating point parameters, then each candidate could be represented as an array of such floating point numbers.

Clearly, we want to search only the most promising search paths into the population, although we must remain aware that sometimes non-promising search paths can be the best route to the result we are looking for. In order to work out which are the most promising candidates, we evaluate each candidate solution using a user supplied evaluation function. In general, this assigns a single numeric *goodness measure* to each candidate, so that their relative merit is readily ascertained during the application of the genetic operators. Undoubtedly, the amount of meaning and the interpretation that can be gleaned from this single value, is crucial to a successful search [3].

2.2 The Future

With respect to the computational paradigm it were considered extended logic programs with two kinds of negation, classical negation, ¬, and default negation, *not*. Intuitively, *not p* is true whenever there is no reason to believe *p (close world assumption)*, whereas ¬*p* requires a proof of the negated literal. An extended logic program (program, for short) is a finite collection of rules and integrity constraints, standing for all their ground instances, and is given in the form:

$$p \leftarrow p_1 \wedge \ldots \wedge p_n \wedge not\ q_1 \wedge \ldots \wedge not\ q_m; \text{ and}$$
$$? \ p_1 \wedge \ldots \wedge p_n \wedge not\ q_1 \wedge \ldots \wedge not\ q_m, \ (n,m \geq 0)$$

where *?* is a domain atom denoting falsity, the p_i, q_j, and p are classical ground literals, i.e. either positive atoms or atoms preceded by the classical negation sign ¬ [7]. Every program is associated with a set of abducibles. Abducibles can be seen as hypotheses that provide possible solutions or explanations of given queries, being given here in the form of exceptions to the extensions of the predicates that make the program.

These extended logic programs or theories stand for the population of candidate solutions to model the universe of discourse. Indeed, in our approach to *GP*, we will not get a solution to a particular problem, but rather a logic representation (or program) of the universe of discourse to be optimized. On the other hand, logic programming enables an evolving program to predict in advance its possible future states and to make a preference. This computational paradigm is particularly advantageous since it can be used to predict a program evolution employing the methodologies for problem solving that benefit from abducibles [8,9], in order to make and preserve abductive hypotheses. It is on the preservation of the abductive hypotheses that our approach will be based to present a solution to the problem of *The Halt Condition in GP*.

Designing such a selection regime presents, still, unique challenges. Most evolutionary computation problems are well defined, and quantitative comparisons of performance among the competing individuals are straightforward. By contrast, in selecting an abstract and general logical representation or program, performance metrics are clearly more difficult to devise. Individuals (i.e., programs) must be tested on their ability to adapt to changing environments, to make deductions and draw inferences, and to choose the most appropriate course of action from a wide range of alternatives. Above all they must learn how to do these things on their own, not by implementing specific instructions given to them by a programmer, but by continuously responding to positive and negative environmental feedback.

In order to accomplish such goal, i.e., to model the universe of discourse in a changing environment, the breeding and executable computer programs will be ordered in terms of the quality-of-information that stems out of them, when subject to a process of conceptual blending [10]. In blending, the structure or extension of two or more predicates is projected to a separate blended space, which inherits a partial structure from the inputs, and has an emergent structure of its own. Meaning is not compositional in the usual sense, and blending operates to produce understandings of composite functions or predicates, the conceptual domain, i.e., a conceptual domain has a basic structure of entities and relations at a high level of generality (e.g., the conceptual domain for journey has roles for traveler, path, origin, destination). In our work we will follow the normal view of conceptual metaphor, i.e., metaphor will carry structure from one conceptual domain (the source) to another (the target) directly.

Let i ($i \in \{1,\ldots,m\}$) denote the predicates whose extensions make an extended logic program or theory that model the universe of discourse, and j ($j \in \{1,\ldots,n\}$) the attributes for those predicates. Let $x_j \in [min_j, max_j]$ be a value for attribute j. To each predicate it is also associated a scoring function $V^i_j\,[min_j, max_j] \rightarrow 0\ldots1$, that gives the score predicate i assigns to a value of attribute j in the range of its acceptable values, i.e., its domain (for the sake of simplicity, scores are kept in the interval 0…1), here given in the form *all(attribute exception list, sub expression, invariants)*. This states that *sub expression* should hold for each combination of the exceptions to the extension of the predicate of the attributes in the *attribute exception list* and the *invariants*. This is further translated by introducing three new predicates. The first predicate creates a list of all possible value combinations (e.g., pairs, triples) as a list of sets determined by the domain size (and the invariants). The second predicate recurses through this list, and makes a call to the third predicate for each exception

combination. The third predicate denotes *sub expression* and is constructed accordingly. The quality of the information with respect to a generic predicate *K* is therefore given by $Q_K=1/Card$, where *Card* denotes the cardinality of the exception set for *K*, if the exception set is not disjoint. If the exception set is disjoint, the quality of information is given by:

$$Q_K = \frac{1}{C_1^{Card} + \ldots + C_{card}^{card}}$$

where C_{card}^{card} is a card-combination subset, with *card* elements.

The next element of the model to be considered, it is the relative importance that a predicate assigns to each of its attributes under observation; w^i_j stands for the relevance of attribute j for predicate i (it is also assumed that the weights of all predicates are normalized, i.e. [1],

$$\sum_{1 \le j \le n} w^i_j = 1, \text{ for all i.}$$

It is now possible to define a predicate's scoring function, i.e., for a value $x = (x_1,\ldots,x_n)$ in the multi-dimensional space defined by the attributes domains, which is given in the form:

$$V^i(x) = \sum_{1 \le j \le n} w^i_j \, V^i_j(x_j).$$

It is now possible to measure the quality of the information that stems from a logic program or theory, by posting the $V^i(x)$ values into a multi-dimensional space, whose axes denote the logic program or theory, with a numbering ranging from 0 (at the center) to 1. For example, one may have what is illustrated by Figure 1, where the dashed area stands for the quality of information that springs from an extended logic program or theory P, built on the extension of 5 (five) predicates, here named as $p_1 \ldots p_5$. It not only works out which are the most promising extended logic programs or theories to model the universe of discourse, making the *Halt Condition in Genetic Programming*.

As an example, let us now consider the case referred to above, where we had a series of data that was produced from a set of water's reservoir results taken over time. In this case we will not have the values that define the output of the function, but a measure of the quality of the water's reservoir (Program 1).

pH(january,0.32).

¬ pH(X,Y) ←
not pH(X,Y) ∧
not exception$_{pH}$(X,Y).

¬ (pH(X,Y) ∧ Y ≥ 0 ∧ Y ≤ 1). / This invariant states that pH takes values on the interval 0…1/

¬ ((exception$_{pH}$(X,Y) ∨ exception$_{pH}$(X,Z)) ∧ ¬ (exception$_{pH}$(X,Y) ∧ exception$_{pH}$(X,Z))).

/This invariant states that the exceptions to the predicate pH follow an exclusive or/

Program 1. The extended logic program for pH with respect to January

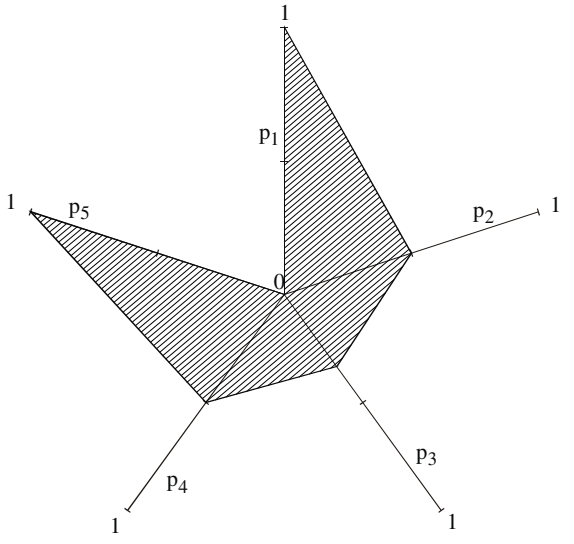

Fig. 1. A measure of the quality-of-information for logic program or theory P

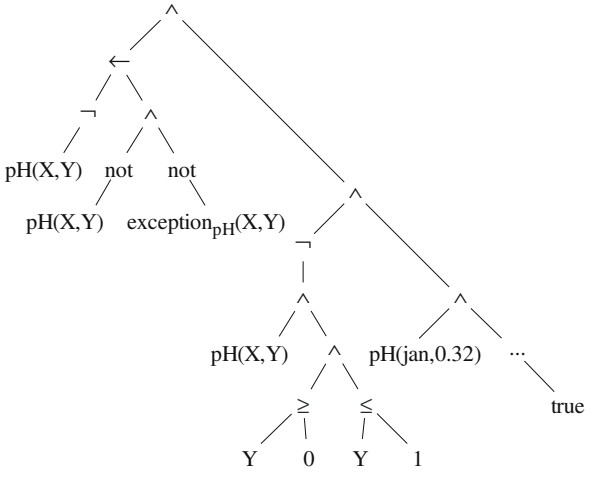

Fig. 2. The evolutionary logic program for pH with respect to January

Therefore we will not guess at some parameters which might inter-operate to produce these values - such as a measure of the acidity or alkalinity of the water's reservoir, the *dissolved oxygen*, the *nitrates*, the *phosphates*, the *chlorophyll*, and so on - but to have such parameters aggregated and giving rise to a set of predicates, which will be given in the form (for the sake of simplicity it will be considered only three predicates, namely those denoting the acidity or alkalinity of the water's reservoir (i.e., the *pH*), the dissolved oxygen (i.e., the *dO*), and the phosphates (i.e.,

the *tP*) whose extensions, with respect to January are given, as it is depicted below, in Program 1 (Figure2), Program 2 (Figure 3), Program 3 (Figure 4) [2].

dO(january,dO).

$\neg dO(X,Y) \leftarrow$
  not dO(X,Y) $\wedge$
  not $exception_{dO}(X,Y)$.

$exception_{dO}(X,Y) \leftarrow$
  dO(X,dO).

$\neg (dO(X,Y) \wedge Y \geq 0 \wedge Y \leq 1)$.

/This invariant states that dO takes values on the interval 0…1/

Program 2. The extended logic program for dO with respect to January

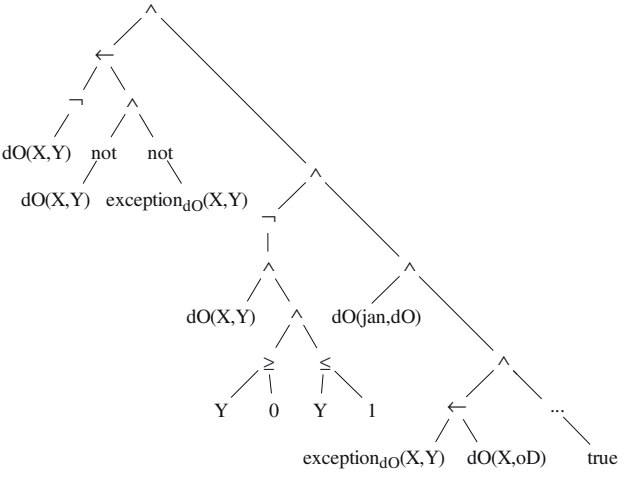

Fig. 3. The evolutionary logic program for dO with respect to January

tP(january,0.21).

$\neg tP(X,Y) \leftarrow$ */The closed word assumption is being softened/*
  not tP(X,Y) $\wedge$
  not $exception_{tP}(X,Y)$.

$\neg (tP(X,Y) \wedge Y \geq 0 \wedge Y \leq 1)$. */This invariant states that pH takes values on the interval 0…1/*

Program 3. The extended logic program for tP with respect to January

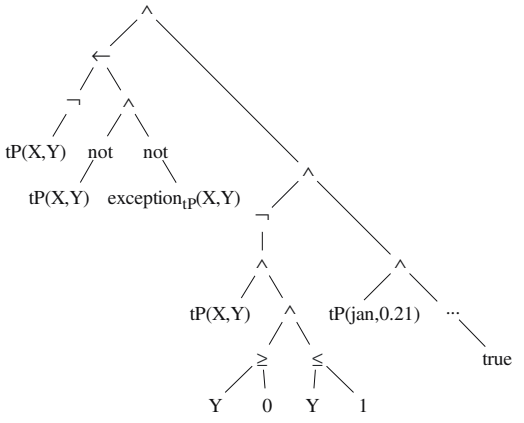

Fig. 4. The evolutionary logic program for tP with respect to January

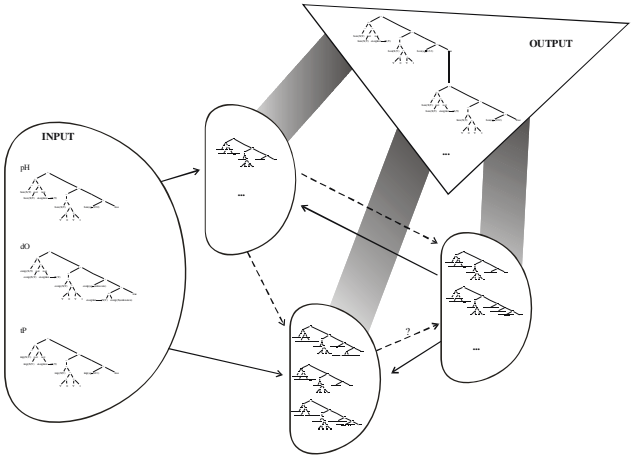

Fig. 5. A blended of the extensions of the predicates pH, dO and tP

Considering what it is illustrated by Figure 5, we might now want to predict how the water's reservoir results will fare in the future, or we may want to fill in some missing data into the series. To do so, we need to evolve the logic theories or logic programs, evolving the correspondent evolutionary logic programs, according to the rules of GP, resulting in what it is depicted by Figures 6,7 [4].

What we are doing, is to put evolution to work for us as well. Indeed, in the present case there is nones change on the quality-of-information when one moves from March to April, i.e., the computational process must stop, once the halt condition is accomplished (Figure 7).

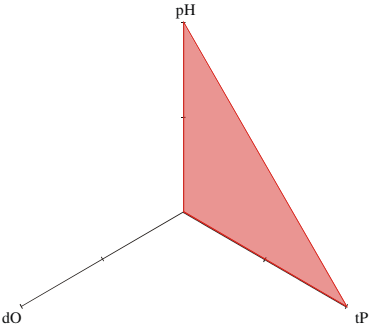

Fig. 6. A measure of the reservoir water's quality for January is given by the paint area

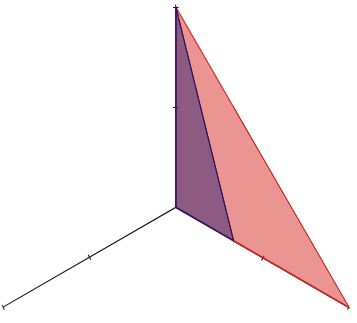

Fig. 7. A measure of the evolution of the reservoir water's quality from January to February

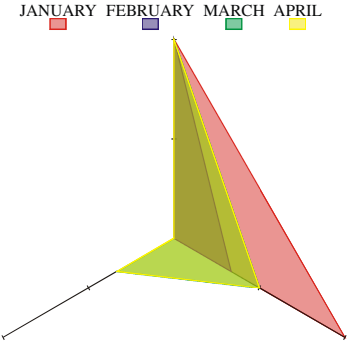

Fig. 8. A measure of the evolution of the reservoir water's quality from January to April

3 Conclusions

This paper shows how to construct a dynamic virtual world of complex and interacting populations, entities that are built as evolutionary logic programs that compete against one another in a rigorous selection regime, where the halt condition of the search process is formalized. In order to produce the optimal solution to a

particular problem, one must evolve the logic program or theory that models the universe of discourse, in which fitness is judged by one criterion alone, the quality-of-information.

Clearly, we work out:

What is the model?

A model in this context is to be understood as the composition of the predicates that denote the objects and the relations that may be established between them, that model the universe of discourse.

What parameters are we seeking to discover?

The extensions of predicates of the kind just referred to above [5].

What do we mean by optimal?

By optimal we mean the logic program or theory that models the universe of discourse and maximizes its quality-of-information factor [6].

How can we measure and assign values to possible solutions?

Via mechanical theorem proving, and program blending [10].

References

[1] Jennings, N.R., Faratin, P., Johnson, M.J., Norman, T.J., Brien, O., Wiegand, M.E.: Journal of Cooperative Information Systems 5(2-3), 105–130 (1996)

[2] Angeline, P.J.: Parse Trees. In: Bäck, T., et al. (eds.) Evolutionary Computation 1: Basic Algorithms And Operators, Institute of Physics Publishing, Bristol (2000)

[3] Rudolph, G.: Convergence Analysis of Canonical Genetic Algorithms. IEEE Transactions on Neural Networks, Special Issue on Evolutionary Computation 5(1), 96–101 (1994)

[4] Teller, A.: Evolving programmers: The co-evolution of intelligent recombination operators. In: Kinnear, K., Angeline, P. (eds.) Advances in Genetic Programming 2, MIT Press, Cambridge (1996)

[5] Analide, C., Novais, P., Machado, J., Neves, J.: Quality of Knowledge in Virtual Entities. In: Encyclopedia of Communities of Practice in Information and Knowledge Management, pp. 436–442. Idea Group Inc., USA (2006)

[6] Mendes, R., Kennedy, J., Neves, J.: Avoiding the Pitfalls of Local Optima: How topologies can Save the Day. In: ISAP 2003. Proceedings of the 12th Conference on Intelligent Systems Application to Power Systems, IEEE Computer Society, Lemnos, Greece (2003)

[7] Neves, J.C.: A Logic Interpreter to Handle Time and Negation in Logic Data Bases. In: Proceedings of ACM 1984 Annual Conference, San Francisco, USA (October 24-27, 1984)

[8] Kakas, A., Kowalski, R., Toni, F.: The role of abduction in logic programming. In: Gabbay, D., Hogger, C., Robinson, J. (eds.) Handbook of logic in Artificial Intelligence and Logic Programming, vol. 5, pp. 235–324. Oxford University Press, Oxford (1998)

[9] Kowalski, R.: The logical way to be artificially intelligent. In: Toni, F., Torroni, P. (eds.) Proceedings of CLIMA VI. LNCS (LNAI), Springer, Heidelberg (2006)

[10] Turner, M., Fauconnier, G.: Conceptual Integration and Formal Expression. Johnson, M.: Journal of Metaphor and Symbolic Activity 10(3) (1995)