

Combining Genetic Programming and Semi-quantitative Representation to Learn Models of Physical Systems

Mehdi Khoury, Frank Guerin and George Macleod Coghill

Department of Computing Science

University of Aberdeen

Aberdeen AB243UE, Scotland, UK

Phone: +44 1224-272747, Fax: +44 1224-273422

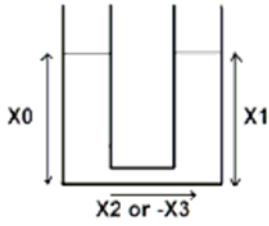
email: {mkhoury,fguerin,gcoghill}@csd.abdn.ac.uk

ABSTRACT: Scientific discovery implies exploring a vast search space of possible hypotheses in the hope of finding a model befitting the available data. Our present aim is to use Genetic Programming (GP) and a representation involving both crisp numbers and fuzzy quantity spaces to learn models of simple physical systems from a set of imperfect and incomplete data. We use the ECJ framework, to learn models of increasing complexity (u-tube, coupled tanks, and cascading tanks). The best fitness is obtained when a model covers all positive examples. Results show that the system can approximate the target models and that the use of a weighted fitness function seems to accelerate the learning process.

KEYWORDS: Genetic Programming, Semi-Quantitative Modelling, Representation, Machine Learning.

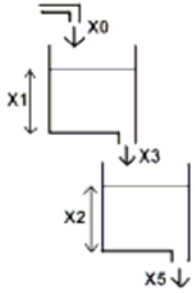
INTRODUCTION

Finding the underlying structure of a system from the observation of its states is a complex task which poses two problems; how to represent such a system, and how to learn the correct model in the most efficient way. Regarding the problem of symbolic representation, we can observe two different approaches in the literature. One involves quantitative modeling. Traditionally used in scientific discovery on systems such as the influential BACON [15], it allows a great degree of precision in the mapping between the model description and an individual data point. Another approach involving Qualitative Reasoning (QR) was developed in the early eighties (De Kee[8], Forbus[9]). It arose from the necessity of obtaining more understandable and relevant information from relatively noisy and imperfect data sets. The QSIM formalism, developed by Kuipers[14], has been influential in this area. It presents constraints in the form of qualitative differential equations where the variables are represented by both their qualitative magnitude (an interval within the given range) and their rate of change (increasing, decreasing or steady). Further developments in fuzzy sets (Zadeh[20]), were also used in more recent QR systems such as FuSiM[18]. Fuzzy representation allows us to analyse data sets with imprecise and ambiguous information but brings accuracy, which is defined as “the ability to guarantee that all the data points are contained within a model “ [16]. An even more recent system called Morven [5] [4] builds on this and introduces the concept of fuzzy vector envisionment which reasons about a non-fixed number of fuzzy derivatives. We use this Morven representation in a semi-quantitative way, which is to say that we use it in conjunction with crisp numbers converted to fuzzy values. This approach is distinct from the semi-quantitative QR system described by Herbert Kay[10]. Regarding the problem of exploring the search space to reach a satisfying solution, most of the QR model learners have been using Inductive Logic Programming(ILP) : Coira’s GENMODEL [7], GOLEM [17] [1] , ILP-QSI [6] among others. While ILP is a systematic method well fitted for exploring a narrow search space, GP might be adequate for exploring a broad search space. Very little work has been done in the area using GP. Varsek [19] used GP, but purely on the basis of QSIM formalism. A very interesting approach was adopted by Koza et al [13] , but with a completely different formalism closer to crisp symbolism. Using the Morven formalism with a GP learner therefore constitutes a reasonably novel approach. We will describe in section 2 the detail of the formalism used in our semi-qualitative models. Then, in section 3, we will examine the GP learning process, and finally we will describe our experiment and discuss the results in section 4.



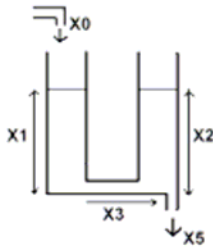
$$\begin{cases} X0 = inputvalue \\ X1 = (2 \times l) - X0 \\ X2 = k \times (X0 - X1) \\ X3 = minus(X2) \end{cases}$$

Figure 1: U-Tube and Target Model



$$\begin{cases} X0 = inputvalue1 \\ X1 = inputvalue2 \\ X2 = inputvalue3 \\ X3 = k1 \times (X1) \\ X4 = X0 - X3 \\ X5 = k2 \times (X2) \\ X6 = X4 - X5 \end{cases}$$

Figure 2: Cascaded Tanks and Target Model



$$\begin{cases} X0 = inputvalue1, \\ X1 = inputvalue2, \\ X2 = inputvalue3, \\ X3 = k1 \times (X2 - X1), \\ X4 = X0 - X3, \\ X5 = k2 \times (X2), \\ X6 = X4 - X5 \end{cases}$$

Figure 3: Coupled Tanks and Target Model

THE SEMI-QUANTITATIVE REPRESENTATION

A model can be broadly defined as a set of constraints describing the relationships linking variables. Our present aim is to learn models of simple physical systems, and in the longer term biological systems. In such systems, especially the latter, there is a need to deal with imprecision and ambiguity. This is why, in practice our model looks like a system of equations engineered to support fuzzy representation. We make use of a fuzzy 4-tuple parametric representation and a mixture of differential planes and fuzzy vector envisionment. Constants are crisp numbers converted to fuzzy values. Operators are designed to provide the Fusim type of fuzzy arithmetic.

FUZZY 4-TUPLE PARAMETRIC REPRESENTATION

The Fusim formalism uses the Fuzzy 4-Tuple Parametric Representation which approximates fuzzy number curves while allowing the concept of degree of membership. A 4-tuple is described using four values a , b , α and β which define the fuzzy number as shown graphically in figure 4 ([2], page 53]). This is very useful for describing real numbers, real intervals, fuzzy numbers and fuzzy intervals.

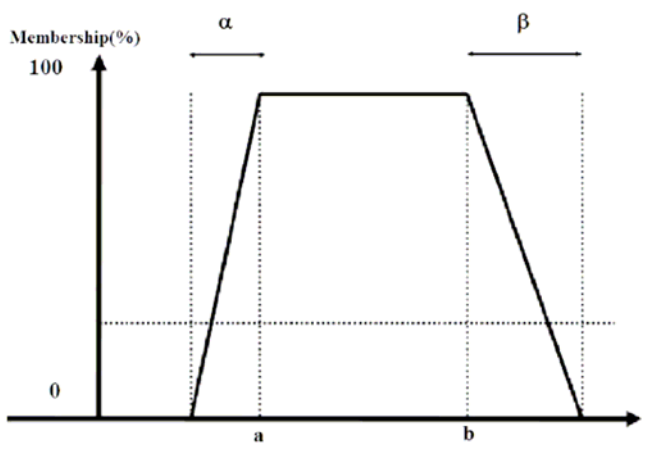


Figure 4: Fuzzy 4-Tuple Parametric Representation

COMBINING FUZZY ARITHMETIC, FUZZY VECTOR ENVISIONMENT AND DIFFERENTIAL PLANES

We apply standard arithmetic operators to fuzzy tuples in the way of Fusim [18]. The table 1 shows how basic arithmetic operators are applied to Fuzzy 4-tuple parametric representation ([2], page 54).

Operation	Result	Conditions
$m \times n$	$ac, bd, a\gamma + c\tau - \tau\gamma, b\delta + d\beta + \beta\delta$	$m >_0 0$ and $n >_0 0$
	$ad, bc, d\tau - a\delta + \tau\delta, -b\gamma + c\beta - \beta\gamma$	$m <_0 0$ and $n >_0 0$
	$bc, ad, b\gamma - c\beta + \beta\gamma, -d\tau + a\delta - \tau\delta$	$m >_0 0$ and $n <_0 0$
	$bd, ac, -b\delta - d\beta - \beta\delta, -a\gamma - c\tau + \tau\gamma$	$m <_0 0$ and $n <_0 0$
	$m = \langle a, b, \tau, \beta \rangle$ and $n = \langle c, d, \gamma, \delta \rangle$	

Table I.: One example of FuSim fuzzy arithmetic: the multiplication operator

The QSIM formalism implies a monotonic function representation. Morgan [3] introduced multiple derivatives for variables. This allows us to not only observe the rate of change of a variable, but also its curvature. This is possible with QSIM only if the user explicitly defines which variable is the derivative of another. It is an automated process in Vector Envisionment. If a variable is defined by a vector of length three, it implies that the variable is defined by its magnitude and first and second derivatives. In terms of models of simple physical system, this gives information not only about magnitude, but also about speed and acceleration. The concept of Vector envisionment was extended to the fuzzy domain by Morven. There are multiple derivatives per variable, but these are fuzzy values. The combination of Fuzzy Vector Envisionment and Differential Planes gives us the ability to use a powerful Qualitative formalism. We can control the number of derivatives for each variable, and thus simplify equations involving these derivatives. For example, when we apply the multiplication operator to two variables, it not only involves the magnitudes, but also their derivatives. Therefore, if tuples A and B each have two derivatives such that their vectors are:

$$A = [a, \dot{a}, \ddot{a}] \text{ and } B = [b, \dot{b}, \ddot{b}], \text{ then } AB = [(ab), (\dot{a}b + \dot{b}a), (\ddot{a}b + \ddot{b}a)] \quad (1)$$

Table II.: Derivative of the product of two Vectors

SETUP AND IMPLEMENTATION OF THE GENETIC PROGRAMMING MODEL LEARNER

The choice was made not to create the GP learner from scratch, but rather to use an open source package already available. The ECJ platform version 1.5 using the Java programming language was selected. Thus, we were able to concentrate on the core of the problem without having to worry about code optimization and the implementation of

classics such as ADFs2. The main problems encountered were more a matter of choice than programming design: the choice of the function and terminal sets, the fitness function, and the generic parameters.

CHOICE OF THE FUNCTION AND TERMINAL SETS

The terminal set contains all the types of leaves of the GP tree and the function set contains all the different types of nodes.

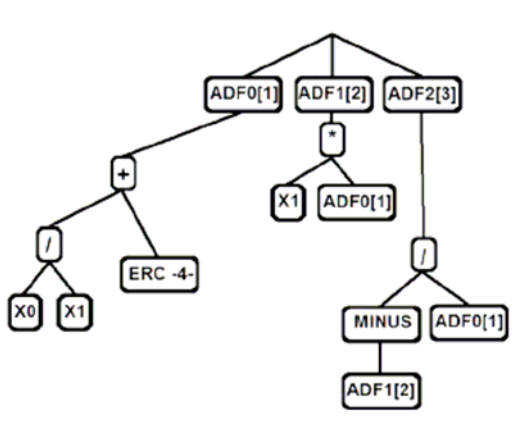


Figure 5: A Model made of three constraints

The Function Set

There are two types of functions: arithmetic operators and the ADF's function defining branch.

- The arithmetic operators are: +, -, ×, ÷, inv, and the unary operator minus.
- The ADF's [12] function defining branch: there is one ADF main function which accepts as many terminal arguments as there are constraints. These constraints can be reused as building blocks to build subsequent and more complex constraints. If there are n constraints, then the ADF function will accept as arguments sub-trees named from ADF0[1] to ADFn-1[n]. A model containing three constraints with the following will be represented as a GP tree similar to the randomly selected sample in Figure 5.

The Terminal Set

The terminal set contains the leaves of the tree. There are three types of terminals in our learner:

- An Ephemeral Random Constant [11] (ERC). It generates constants at random.
- The variables: each variable is in fact a Vector Variable as previously defined by Fuzzy Vector Envisionment, and therefore consists of a set of 4-tuples fuzzy subsets. The first fuzzy tuple represents the qualitative magnitude of the variable, and the next tuples its different orders of derivations. The number of degrees of derivations can be set up by the user, but may drastically influence the number of computations.
- ADFs: they can be reused with operators and other ADFs. This way, the second constraint can make use of the first constraint; the third one can make use of the first and second constraints, and so on.

THE FITNESS FUNCTION

We have two sets of variables; the positive data used in the learning examples, and the data produced by the newly generated model. We first introduce the input variables from the positive examples in the newly generated model, and then calculate what endogenous variables it creates. We then measure the distance between learning data and generated variables. The distance between fuzzy variables used in fitness evaluation is calculated following a method provided by Shen and Leitch[18]. We also introduce fitness calculation with incomplete data by considering that the distance between two variables of unknown magnitude or between a known and unknown magnitude is null. This works the same way for derivatives. The best fitness is zero, and the worst, infinity. Two different cases of fitness evaluation are examined in this paper:

- the non-weighted fitness evaluation where the overall sum of the distances for all learning examples is used to evaluate the raw fitness.
- the weighted fitness function where the raw fitness is not just the linear sum of distances, but is weighted by the number of correct constraints found in the system. For n examples to learn from, on the first $n - 1$ examples, we take away from the base fitness the proportion of constraints satisfied in the generated model. On the last example n , we use the non-weighted fitness evaluation. Not doing so seems to over-score the fitness, meaning that incorrect individuals would be evaluated as perfectly fit. In the end, the learner manages to find a satisfying solution with a gain of performance (see experiment results). Our model has a raw fitness α (a positive number), and satisfies δ constraints over μ . We learn over n examples. The weighted fitness ϕ will be:

$$\phi = \sum_{i=1}^{n-1} \alpha \times (\delta / \mu) \quad (2)$$

GENERIC GP PARAMETERS

In general, most of the experiments were done with a population from 20000 to 40000 individuals, this number being found pragmatically sufficient after several attempts to solve problems of limited complexity such as our models. All experiments were run on one machine (a Pentium 3 GHz with 2GB of RAM). The maximum number of generations was fixed to 200. The system first evaluates the individuals using the Koza Standardized Fitness. The results are then displayed using Adjusted Fitness into a scale from 0.0 exclusive (worst) to 1.0 inclusive (best) in order to facilitate the reading and emphasize the differences between the very good individuals. The top part of the individuals are generated using the 'full' method while the constraints are generated using the 'Rampant half-and-half'. There is no elitism. The method of selection in use is tournament selection, and the default tournament size is 7. Probability of crossover is 90%, and probability of reproduction is 10%.

THE EXPERIMENTS AND RESULTS OBTAINED

Using easy-to-learn "dummy" models, we will compare the number of generations needed to learn a model while using a normal and a weighted fitness function. We will also measure the average time per generation as a function of several parameters. Finally, we will analyse the performance of the system when trying to find the three models while varying the population size. Models which don't reuse constraints (ADFs are not reused) seem easier to find for the learner as they generally take fewer generations. A possible explanation may be that the models being simpler, it is easier to generate a matching tree. Anyway, these "dummy" models are ideal for testing. The following graph describes the average number of generations needed over five runs to find a dummy model depending on the chosen fitness function. The models have from four to nine constraints. We compare results obtained with a normal fitness function and with a weighted fitness function (see Figure 6).

COMPARING NORMAL FITNESS WITH WEIGHTED FITNESS

The weighted fitness function seems to enhance performance compared to normal fitness as it reduces the number of generations by on average 46% for a 4 to 7-constraints model (see Figure 6). From this observation, we advance the hypothesis that if we consider the fitness of a model as the linear sum of the fitnesses of its components (here, the constraints), then the convergence to a solution is slower than if the fitness is the weighted sum of the fitness of its components (detail of weighted fitness described in Section 3.2).

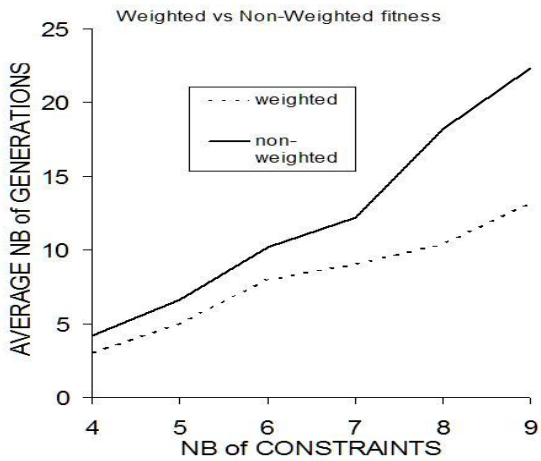


Figure 6: Time in function of 2 parameters

TIME ANALYSIS

Figure 7 shows the average time per generation it takes to find a dummy model, depending on two parameters: the size of learning set, and the number of constraints. As we can see, the time seems to increase linearly with both parameters. This might suggest that time is not an exponential but rather a polynomial if not linear function of the size of the learning set, and of the number of constraints present in the target model.

Time performance per generation

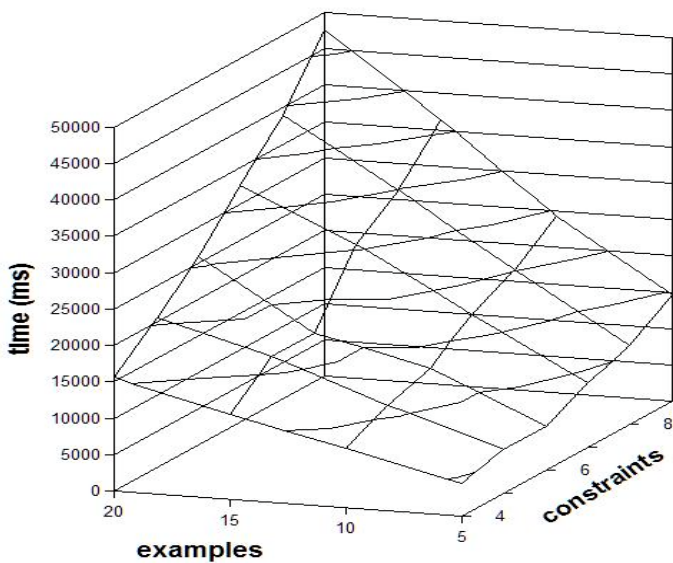


Figure 7: Weighted Fitness vs. Normal Fitness

PERFORMANCES WHILE LEARNING THE U-TUBE, CASCADED TANKS AND COUPLED TANKS.

Considering the GP parameters, there is a 90% chance for a new potential solution to be generated from a crossover, and a 10% chance to be generated from mutation. And as the larger the population, the greater is the number of possible combinations of two parents, so the greater is the diversity of solutions produced 90% of the time. It is therefore not too surprising that the efficiency of the learner seems to be linked to the size of the population. Performance is analysed by trying to answer several questions. Does the learner find a solution with the present population size? If so, how many generations does it take and how similar is the generated solution to the target model? If no solution is found, the population size is increased and we will start the same process again. A model is a set of constraints defining the

relationships between measured variables. In the simple physical systems we are examining, these variables (starting with the letter X) are either the level of water measured at one place, or the flow of water between one place or another place. Generated constants are linked to level (starting with a letter l), or flow (starting with a letter k) of water. The u-tube model is shown in figure 1. In this example: X0 is the level of water on the left side (our input value), X1 is the level of water on the right side, X2 is the flow of water from left to right, X3 is the flow of water from right to left, l is a constant expressing the level of water when there is equilibrium, and k is a constant influencing the flow between both sides of the tube (for example the diameter of the pipe). Similarly, the slightly more complex models of cascaded tanks and coupled tanks are shown respectively in figure 2 and 3. During the experiments, the learner did sometimes find the exact target model (constants included), but mostly approximations to it(in the case of the U-tube, generally one constraint in four is slightly deviant). Here is one example of such an approximation:

$$\left\{ \begin{array}{l} X0 = \text{inputvalue} \\ X1 = (2 \times l) - X0 \\ X2 = X1 - ((2 \times k) - X0)^2 \\ X3 = \text{minus}(X2) \end{array} \right.$$

We see two possible explanations for this oddity:

- The use of a limited number of examples might enlarge the space of possible solutions. In this experiment, we learn from a set of only ten fuzzy “data points”. The obtained model might be perfectly adapted for these examples, but it also might not work if we add other examples.
- It may also be because of the relative complexity of constraints that reuse other constraints and also generate constants.

Depending on the complexity of the target model, the size of the population was sometimes insufficient to find a solution in the imparted number of generations. This drove us to increase the population size when no solution was found after five runs. The table 2 is describing each model, and giving the population size and number of generations required to converge to a solution. These results could suggest that the minimum population size required is particularly dependant on the number of constraints reused in each system.

Model	Constraints	Constants	Input values	Reused var.	Pop size	Generations
u-tube	4	2	1	4	20000	9
cascaded	7	2	3	6	30000	18
coupled	7	2	3	7	40000	39

Table III.: Possible factors influencing the population size

CONCLUSION

We have observed how factors such as the number of constraints, the size of the learning set, and the population size parameters could influence the performance of the model learner. We also noticed the positive impact of using a weighted fitness function during evaluation. Further work might be done to see if this result can be generalized. We are also driven to look for different ways to optimise the learner. GP may explore a broad search space more efficiently, but it remains a costly method on the point of view of computational resources. Further work is needed to find ways to minimise such costs. One interesting direction to investigate would be how to reuse models already found into the learning process by integrating them in the starting population. We did not implement logic and recursive operators in the function and terminal sets. Adding these features to the Morven formalism might enhance an already powerful representation. Assuming that the GP approach approximates models nicely, future work may involve a merging between GP and ILP. This would allow us to explore more efficiently the narrow search space that stands between an approximation and a precise model.

REFERENCES

- [1] I. Bratko, S. Muggleton, and A. Varsek. Learning qualitative models of dynamic systems. In *ML*, pages 385–388, 1991.
- [2] A. Bruce. *JMorven: A Framework for parallel non-constructive qualitative reasoning and fuzzy interval simulation*. PhD thesis, Department of Computer Science, University of Aberdeen, 2006.
- [3] G. M. Coghill. *Vector envisionment and compartmental systems*. Master’s thesis, University of Glasgow, 1992.
- [4] G. M. Coghill. *Mycroft: a framework for Constraint Based Fuzzy Qualitative Reasoning*. PhD thesis, Heriott-Watt University, 1996.
- [5] G. M. Coghill and M. Chantler. *Mycroft: a framework for qualitative reasoning*. In *Proceedings of the Second International Conference on Intelligent Systems Engineering*, pages 49–55, Hamburg, Germany, 1994.
- [6] G. M. Coghill, S. M. Garrett, A. Srinivasan, and R. D. King. *Qualitative system identification from imperfect data*. Technical report, University of Aberdeen, 2002.
- [7] E. Coiera. *Learning qualitative models from example behaviours*. In *Proceedings of the Third Qualitative Physics Workshop*, Stanford University, Palo Alto, Aug. 1989.
- [8] J. de Kleer. *Multiple representations of knowledge in a mechanics problem-solver*. In *IJCAI*, pages 299–304, 1977.
- [9] K. Forbus. *Qualitative reasoning about physical processes*. In *Proceedings of IJCAI’81. International Joint Conferences on Artificial Intelligence*, 1981.
- [10] H. Kay, B. Rinner, and B. Kuipers. *Semi-quantitative system identification*. *Artif. Intell.*, 119(1-2):103–140, 2000.
- [11] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [12] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.
- [13] J. R. Koza, W. Mydlowec, G. Lanza, J. Yu, and M. A. Keane. *Reverse engineering and automatic synthesis of metabolic pathways from observed data using genetic programming*. Technical Report SMI-2000-0851, Stanford Medical Informatics, Nov. 7 2000.
- [14] B. Kuipers. *Qualitative simulation*. *Artificial Intelligence*, 29(3):289–338, 1986.
- [15] P. Langley. *BACON: A production system that discovers empirical laws*. In *IJCAI*, pages 344–344, 1977.
- [16] R. Leitch, Q. Shen, G. Coghill, and M. Chantler. *Choosing the right model*. In *IEE Proceedings - Control Theory and Applications*, pages 435–449, 1999.
- [17] S. Muggleton and C. Feng. *Efficient induction of logic programs*. In *ALT*, pages 368–381, 1990.
- [18] Q. Shen and R. Leitch. *Fuzzy qualitative simulation*. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(4):1038–1061, July-Aug. 1993.
- [19] A. Varsek. *Qualitative model evolution*. In *IJCAI*, pages 1311–1316, 1991.
- [20] L. A. Zadeh. *Fuzzy sets*. *Information and Control*, 8:338–353, 1965.