A Genetic Representation for Dynamic System Qualitative Models on Genetic Programming: A Gene Expression Programming Approach

Ramiro Serrato Paniagua[1], Juan J. Flores Romero[1], and Carlos A. Coello Coello[2]

[1] División de Estudios de Posgrado, Facultad de Ingeniería Eléctrica, Universidad Michoacana de San Nicolás de Hidalgo, Morelia 58000, México
[2] CINVESTAV-IPN, Computer Science Department, México City 07360, México
raspacorp@yahoo.com.mx, juanf@umich.mx, ccoello@cs.cinvestav.mx

Abstract. In this work we design a genetic representation and its genetic operators to encode individuals for evolving Dynamic System Models in a Qualitative Differential Equation form, for System Identification. The representation proposed, can be implemented in almost every programming language without the need of complex data structures, this representation gives us the possibility to encode an individual whose phenotype is a Qualitative Differential Equation in QSIM representation. The Evolutionary Computation paradigm we propose for evolving structures like those found in the QSIM representation, is a variation of Genetic Programming called Gene Expression Programming. Our proposal represents an important variation in the multi-gene chromosome structure of Gene Expression Programming at the level of the gene codification structure. This gives us an efficient way of evolving QSIM Qualitative Differential Equations and the basis of an Evolutionary Computation approach to Qualitative System Identification.

Keywords: Genetic Representation, Genetic Programming, Gene Expression Programming, Qualitative Reasoning, QSIM, System Identification, Evolutionary Computation.

1 Introduction

A Qualitative Model is a qualitative description of a phenomenon, that description is useful for answering particular questions about it; those questions are focused on the phenomenon-behavior's qualitative characteristics. The phenomena could be Dynamic Systems in the real world. QSIM is a formalism that allows us to model qualitatively a Dynamic System, it has a firm mathematical foundation and is easy to understand. In this work QSIM is used for representing Qualitative Models in a novel Genetic Representation. The final goal of this representation is to be part of an Evolutionary Algorithm which will perform System Identification (System Identification is the task of discovering a model from observations [6], also called Model Learning). The proposed Genetic Representation is easy to implement because it does not use any complex data structure (such as nonlinear pointer-based trees or

A. Gelbukh and A.F. Kuri Morales (Eds.): MICAI 2007, LNAI 4827, pp. 30–40, 2007.

graphs), it uses a linear chromosome representation based on Gene Expression Programming (GEP) created by Ferreira [4]. Nevertheless, our representation makes important variations which give it the capability of storing Qualitative Differential Equations in QSIM form.

There is few work on Qualitative System Identification using the QSIM representation and Evolutionary Computation. Alen Varsek [2] built a Qualitative Model Learner based on QSIM that uses Genetic Programming (GP). He uses a binary tree representation where the leaves are QSIM constraints and branching points form the hierarchical structure, trees have different sizes and number of leaves in a given interval. The genetic operators used are the classic GP operators described in [3]. The genetic representation is not discussed enough in Varsek's work. He does not describe the Function and Terminal sets used in his GP algorithm. In the examples used in Varsek's paper there are no constraints with corresponding values, it is commented in the article that this assumption is for simplifying the model. However, Varsek does not specify clearly if this choice was made because of a limitation of the representation, a poor performance of the evolution process, an excessive cost in time and space, or any other aspect. Corresponding values give the QSIM constraints an important expressive power; by using them we can introduce pieces of previous knowledge about the system being modeled, and thus more likely to improve the learned-models accuracy.

Another approach on learning Qualitative Models is the work by Khoury, Guerin and Coghill [7]. They built a semi-quantitative model learner using GP, using a tree representation. They do not use QSIM for the model representation but a combination of Fuzzy Vector Envisionment and Differential Planes. By using a Framework called ECJ, they use Automated Defined Functions (ADFs), which allow them to reuse pieces of the tree in any branch of it. The Terminal set contains all the types of leaves in the tree, which can be Ephemeral Random Constants, variables in the form of fuzzy vectors, and finally ADFs that allow the algorithm to embed restrictions. The Function set contains arithmetic operators as well as ADFs. Basically, the tree representation structure is as follows: the root of the tree is a main ADF function, whose number of arguments define the number of branches at the first depth-level in the tree and therefore, the number of constraints encoded. After the first depth level, there are the subtrees that represent the constraints. The trees have not a fixed size, but have a maximum length; they do not describe in detail the genetic operators used; they only comment about the use of crossover and reproduction, so it should be assumed they used the Koza's definitions. In the conclusion they mention that GP is a costly method from the point of view of computational resources, very probably the evaluation of the fitness function is the main factor for that computational cost. Also, another factor could be the nonlinear pointer tree representation if used, the use of a Java based GP framework instead of one based in C/C++ could be another factor.

Section 2 presents the concepts that serve as basis for the development of the present work. Section 3 describes the proposed genetic representation. Section 4 defines the genetic operators that can be applied to the individuals.

2 Background

Evolutionary Computation is based fundamentally on Darwin Natural Evolution. Natural Evolution can be seen as a natural population-based optimization process [1], where the stronger individuals are those who are more adapted to their environment and are a product of that optimization process. Those strong individuals have a bigger probability to survive in their environment and as a consequence, to propagate their genetic information through the population in the next generations. In Evolutionary Computation, individuals are possible solutions to problems that human beings want to solve, commonly those problems are engineering or mathematical problems which are not easy to solve using other techniques. Natural evolution does not change the individuals characteristics at the level of their phenotype, it works in their genetics. This indirect change gives this process a powerful exploring mechanism due to the Pleiotropy and Poligeny [1] effects and many others present in the genes coding-decoding process. In Evolutionary Computation there are some paradigms that make use of evolution at the level of genetics, two of them are: Genetic Algorithms and Gene Expression Programming.

One of the main aspects in the evolution process operation is the Genetic Representation. Genetic Representation is the way nature encodes the phenotypic characteristics in the genes, therefore the Genetic Representation must be expressive enough to encode every possible phenotypic characteristic for that specie of individuals. In the same way, Evolutionary-Computation Genetic Representation has to be sufficiently expressive to encode every possible solution for an specific problem in its search space. Another aspect that the Genetic Representation has to satisfy, is the one related to the correct applicability of genetic operations, in Evolutionary Computation it is also desired that genetic operators can be easily implemented.

2.1 Genetic Programming

GP is an Evolutionary Computation Paradigm whose aim is to deal with the problem of Program Induction [3]. That is, the discovery, from the search space containing all possible computer programs, of a program that produces some desired output when presented with some particular input. A wide variety of problems can be expressed as problems of program induction. This means that a computer program could be seen as a generic representation form, for possible solutions to those problems. So, computer programs may represent a formula, a control strategy, a video game, a mathematical model, etc. Computer programs are hierarchical structures that can fit a tree form, that tree is called "computer program parse tree", GP population individuals are computer programs parse trees. GP representation is thus a non-linear non-fixed length structure. In this paradigm there is not a clear separation between the phenotype and the genotype; an individual functions simultaneously as genome and phenome.

2.2 Gene Expression Programming

GEP is a genotype/phenotype evolutionary algorithm [4]. Its representation is a fixed length multigenic linear chromosome, where the genes have a special structure composed of a head and a tail. It is important to notice that the fixed length affects the

genotype of the individuals, but the decoded individuals (Computer Programs parse trees) can have different sizes and shapes. The GEP individuals encode parse trees of computer programs like in GP but GEP evolutionary process works at the level of the genotype. Ferreira [4] proposes the use of a set of genetic operators: Replication, Mutation, IS Transposition, RIS Transposition, Gene Transposition, 1-Point Recombination, 2-Point Recombination, Gene Recombination (Recombination is also called Crossover). As Ferreira comments [4], the advantages of a Genetic Representation like the one in GEP are the following. The chromosomes are simple entities: linear, compact, relatively small, easy to manipulate genetically. The genetic operators applied to them are less restricted than those used in GP for example, the mutation operator in GP differs from point mutation in nature in order to guarantee the creation of syntactically correct programs (as observed by Ferreira in [4]). The implementation of mutation in GP as shown in [3] first randomly selects a node from the parse tree, then the node and the sub-expression tree below the node are replaced with a randomly generated tree.

The coding in the GEP chromosomes is named Karva language, the genes in the chromosome contain entities called open reading frames (ORFs) whose length defines the length of the sub-expression tree encoded in the gene. The ORFs length could be equal or less than the length of the gene, this allows the possibility of encoding trees with different sizes and shapes. But if the encoded tree is not always using all the space in the gene, what is the function of those non-coding regions? These regions allow the algorithm the modification through genetic operators of the chromosome without restrictions, because the size and structure of the genes remains constant despite the size of the encoded sub-expression tree. Also, these non-coding regions can store important genetic information that can emerge again in the evolutionary process.

2.3 QSIM

QSIM is a representation for Qualitative Differential Equations (QDEs); QSIM is also an algorithm for qualitative model simulation. In this paper we will focus on the QSIM representation. QDEs are abstractions of differential equations and differential equations are as well abstractions or models of the real world Physical Systems. QDEs are Qualitative Models, which are general descriptions of the qualitative characteristics and behaviors of a physical phenomenon. These models express states of incomplete knowledge and can be used to infer useful conclusions about the behaviors of that phenomenon.

In the QSIM representation [5], a QDE is a 4-tuple <V,Q,C,T>, where V is a set of qualitative variables (this variables represent reasonable functions of time); Q is a set of quantity spaces one for each variable in V; C is a set of constraints applying to the variables in V; T is a set of transitions which define the domain of applicability of the QDE. The quantity space of a variable is a totally ordered list of important values that serve as qualitative-regions boundaries, those values are called landmark values or simply landmarks. The qualitative constraints are relationships among the qualitative variables in the QDE. There is a basic repertoire of constraints in QSIM. Figure 1 lists those constraints and their meanings.

In Figure 1, the points between brackets are the corresponding values (brackets indicate they are optional), which are tuples of landmark values that the variables can take in some constraint; in other words, the point where the constraint is satisfied. In the description of the basic set of constraints shown in Figure1, there are some constraints that do not use corresponding values, these are the derivative and the constant constraints.

(add x y z [(a1 b1 c1) (a2 b2 c2) ...]) iff (t) x(t) + y(t)= z(t) and (i) ai + bi = ci {**corresponding values**}
(mult x y z [(a1 b1 c1) (a2 b2 c2) ...]) iff (t) x(t) · y(t) = z(t) and (i) ai · bi = ci
(minus x y [(a1 b1) (a2 b2) ...]) iff (t) y(t) = - x(t) and (i) bi = - ai
(d/dt x y) iff (t) y(t) = (d/dt) x(t)
(constant x)
(M+ x y [(a1 b1) (a2 b2) ...]) iff (t) y(t) = f(x(t)) where f belongs to the set of reasonable monotonously increasing functions and (t)(i) x(t) = ai iff y(t) = bi
(M- x y [(a1 b1) (a2 b2) ...]) iff (t) y(t) = f(x(t)) where f belongs to the set of reasonable monotonously decreasing functions and (t)(i) x(t) = ai iff y(t) = bi

Fig. 1. The QSIM constraints basic set

3 Genetic Representation

Genetic Programming (GP) [3] and Gene Expression Programming (GEP) [4] are Evolutionary Algorithms that evolve computer programs. They differ in the representation and in the form of their genetic operators; both of them use two element sets that form the alphabet used in their representation. Those sets are F the set of functions and T the set of terminals [3][4].

The GEP representation encodes an expression tree, like in GP. The difference between them is that GP individuals are directly the computer-programs parse trees while GEP uses a phenotype-genotype approach, the genotype is structured by a multi-gene chromosome and the phenotype is the computer-program parse tree. Each gene encodes a sub-expression tree (a piece of the computer-program parse tree) using the Karva notation, which is just a width-first linearization of the sub-expression tree (see [4]), the full expression tree (the computer-program parse tree) encoded in the chromosome is the result of linking the sub-expressions in each gene using a link function [4] (i. e. A function used to join the sub-expression trees in the decoding process). Each gene in the GEP Genetic Representation has a two-part structure which is formed by a head and a tail [4], this structure guarantees that every gene decodes to a valid sub-expression tree; the head contains functions or terminals while the tail contains only terminals, see Figure 2.

The name for the proposed representation is "Qualitative Model Genetic Representation" (QMGR).

In GP and GEP there are 2 sets F and T whose union contains all possible elements in an individual tree or chromosome; in QMGR we have F, T and a third one L. The F set is formed of the QSIM constraints; T contains the qualitative variables given by the user; L is the set of all landmark values of the variables quantity spaces in the model. The union of these three sets contains all possible elements in a QMGR chromosome.

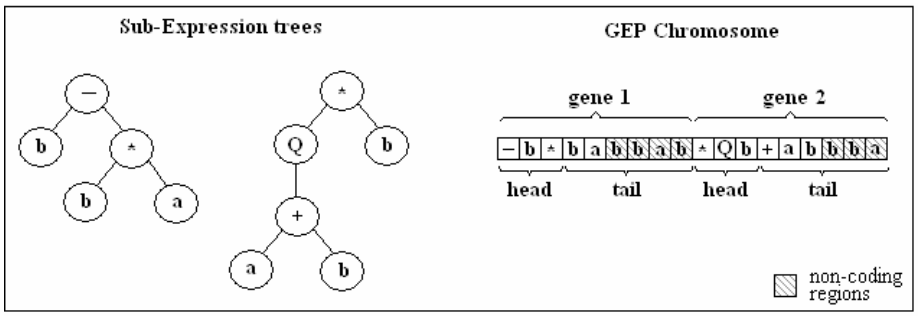

Fig. 2. GEP Genetic Representation

For QSIM models it is not needed to use the Karva notation because those models do not have a tree form necessarily. The structure of the chromosome in the proposed Genetic Representation has a head-tail structure as GEP, we added a third part called CV because it contains the corresponding values of the QSIM constraints. We fixed the head's length to one because it is always encoded one QSIM constraint in a gene (i. e. the head of each gene stores the name of a constraint). The length of the tail is determined by the maximum arity of the functions in F since the tail contains the arguments applied to the constraint defined in the head; Equation 1 determines the length of the CV.

$$cvLength = maxA * numCV \quad (1)$$

where numCV is the number of corresponding values to be encoded; maxA is the greatest arity of all functions in the F set. Table 1 shows a comparative between GEP and QMGR chromosome's structre.

In QMGR it is encoded one QSIM constraint per gene, therefore the number of constraints in an individual is the same as the number of genes. The user has to make a good choice of the number of genes he would use, as well as the other parameters in the algorithm.

Figure 3 shows how a set of QSIM constraints is encoded in a chromosome using QMGR. For the easy reading and for efficiency in storing the chromosome's linear structure, we use a mapping between the name of a QSIM constraint and a one-character symbol to be stored in the chromosome (this could be also used with the variables and the landmarks if needed) see Figure 3.

This representation provides a generic structure that allows encoding restrictions of any arity and any number of corresponding values, which are encoded in the CV region of each gene. To encode constraints that do not use corresponding values, we need a form of representing the absence of them. In a QMGR individual all genes in a chromosome have the same structure, for example if we have a maximum arity of 3 in the F set and a number of corresponding values equal to 2, the length of the CV will be 6 by using the equation (1). The CV length is a parameter that affects all the individuals in an instance of the QMGR. We define an instance of QMGR as the set which contains all the individuals being processed or studied, the F, T, and L sets, as well as the structural characteristics of the chromosomes in that instance, those are:

the head, tail, CV and gene lengths, the number of corresponding values, the number of genes and the chromosome length.

To solve the problem of encoding constraints that do not use corresponding values or that have a less number of them than the given as parameter, we can introduce an element to the L set which represents the absence of a corresponding value. This element could be represented as the symbol "#" and called "null element". Thus, the chromosome-decoding parser will know when to omit the creation of corresponding values if it finds a null character. This approach allows QMGR to encode constraints that do not use corresponding values or to modify the number of them. This null element can be inserted in the individuals randomly during the initial population creation in an evolutionary algorithm. We also use another approach for dealing with the absence of corresponding values by means of the crossover operator, this approach is described in section 4.1.

4 The Genetic Operators

The role of genetic operators in an evolutionary computation algorithm is to serve as the evolutionary-process engine. Genetic operators allow the evolutionary process to explore and exploit the search space. The genetic-operators main goal is the introduction of genetic diversity in the population of individuals in an evolutionary process. We propose three genetic operators for using with QMGR individuals, one- and two- point crossover and mutation.

Table 1. GEP and QMGR chromosome's structure comparative

Structure's part	GEP	QMGR
Head	Contains symbols from F U T; the length can be defined of any size.	Contains symbols from F; the length is fixed to 1.
Tail	Contains symbols from T; the length is a function of the head's length and the maximum arity of all functions in F. $t=h(n-1)+1$ Where t is the length of the head, and n is the maximum arity.	Contains symbols from T; the length is always equal to the maximum arity of the constraints in F.
CV	Not existent.	Contains symbols from L; its length is a function of the number of corresponding values and the maximum arity of the constraints in F, see equation 1.

4.1 Crossover

The crossover genetic operator can be easily applied to QMGR individuals. The structure of QMGR allows us to implement the crossover operators defined in GEP and in Genetic Algorithms; this feature can be generalized to n-point crossover. The QMGR structure guarantees that any offspring derived from the crossover operation will encode a valid QSIM-QDE, this is inherited from the GEP representation [4]. The one- and two- point crossover operators are illustrated in Figure 4.

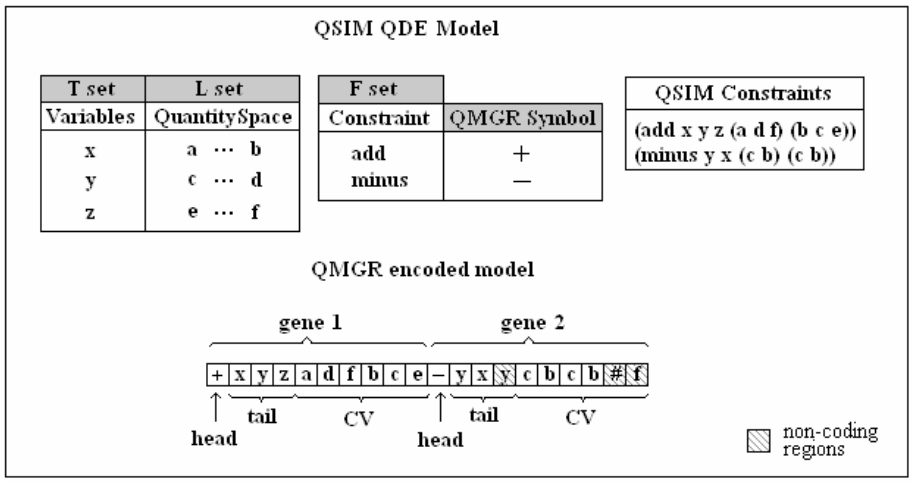

Fig. 3. QMGR Genetic Representation

It is necessary to observe that crossover can generate offspring with non-valid corresponding values. For example, let us suppose we have the following 1-gene individuals in an instance of QMGR, with the following chromosome structural-parameters: CV length equal to 6, maximum arity equal to 3 and number of corresponding values equal to 2.

0 1 2 3 4 5 6 7 8 9
\+ x y z a c e b d f
\- y z z d e c e a c

The quantity spaces of the variables are:
x: a … b
y: c … d
z: e … f

The elements in the positions of the CV in each individual gene are valid landmarks in the quantity space of their respective variables. Therefore, all the decoded corresponding values in the CVs will be valid.

If we select a crossover point between position 4 and 5 the offspring will be the following.

```
0 1 2 3 4 5 6 7 8 9
+ x y z a e c e a c
- y z z d c e b d f
```

Analyzing the genes of these offspring it can be seen that in individual one, the two corresponding values are (a, e, c) and (e, a, c). In the first corresponding value, "e" and "c" are landmarks which not belong to the quantity spaces of the respective variables, these variables are "y" and "z"; in the second corresponding value of the first individual, none of the landmarks belong to the quantity spaces of the corresponding variables. Thus both corresponding values of the CV from the first offspring are invalid. The corresponding values of the second individual are (d, c) and (e, b). The first element of the corresponding values for this individual corresponds to the "y" variable and the second element corresponds to the "z" variable. We observe that the first element of the first corresponding value "d" belongs to the quantity space of "y" which is the corresponding variable for this element, but "e" does not satisfy this rule. Neither "c" which is the second element of the first corresponding value nor "b" which is the second element of the second corresponding value, belongs to the quantity space of the "z" variable. Thus both corresponding values in the second offspring are not valid.

In the example, none of the offspring has a valid corresponding value. The problem of generation of invalid corresponding values through crossover, which also means the generation of non-valid individuals, can be seen as an advantage. It can be used as another (see section 3) way of destroying and varying the number of corresponding values in a constraint. To do that, the chromosome-decoding parser must detect when a symbol in the CV does not belong to the quantity space of the corresponding variable, and therefore not decoding it, jumping to the next one, and so on.

The examples in Benjamin Kuipers' book [5] indicate, that the percentage of constraints that use corresponding values is very small; that means, in general, that when we model a physical system we rarely know the points the functions pass through. The probability assigned to the crossover operators in Genetic Algorithms, GP and GEP is usually high. If we use QMGR in an evolutionary algorithm and use a high probability value for crossover, it will be very likely that this operator will destroy a lot of corresponding values through generations, resulting this in a continuously decreasing percentage of individuals with valid corresponding values. This behavior allows the learned models to be more similar to those in the real world.

4.2 Mutation

It is possible to use the point mutation operator on QMGR but we suggest to restrict the operator in the following case: when the chromosome position to be mutated is located in the CV of the genes, this is, stores a corresponding value, it should change the stored value to other that belongs to the set L and also, to the quantity space of the variable referred by this corresponding value. The reason for this restriction is, for conserving the number of valid corresponding-values in that gene. As seen in section

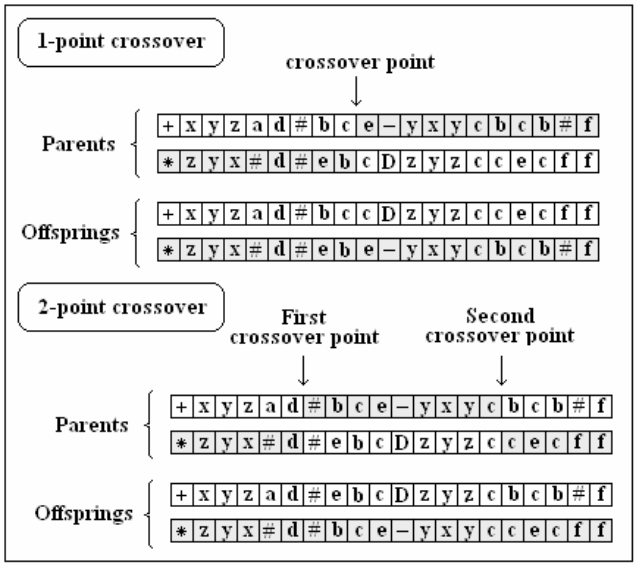

Fig. 4. Crossover operators

4.1 the corresponding values are destroyed when they contain values not existent in the quantity space of the referred variable. Crossover in QMGR has a high corresponding-values destruction power, so it is necessary to avoid that destructiveness in the mutation operator.

There are two more restrictions in the application of the mutation operator, but these ones are mandatory for obtaining valid mutated individuals. When the position to be mutated is located in the tail of any gene, it has to change the stored value only to one element of T. Finally when the position to be mutated is located in the head of any gene, it has to change the stored value only to one element of F.

5 Conclusion

In this paper we presented a new Genetic Representation called QMGR. It allows the easy application of genetic operators like crossover and one-point mutation. It uses a linear fixed length multigenic chromosome, which can encode qualitative models of different sizes. QMGR is designed to encode QDEs in the QSIM qualitative representation. QMGR is efficient because it does not need the use of non-linear pointer-based data structures. QMGR's structure encodes the QDEs in a natural form, storing each constraint in one gene in the QMGR chromosomes. This representation can be used in an evolutionary algorithm aggregating a fitness evaluation function. That evaluation function could use the QSIM simulation algorithm for generating the behaviors of the learned models, those behaviors can then be compared to the observations of the system to be modeled. Current research work deals with the implementation of an Evolutionary Algorithm that completes the system identification process at the qualitative level.

Acknowledgments. The second author acknowledges support from CONACyT project No. 51729. The third author acknowledges support from CONACyT project No. 45683-Y.

References

1. Fogel, D.B.: An Introduction to Simulated Evolutionary Optimization. IEEE Transactions on Neural Networks 5(1) (January 1994)
2. Varsek, A.: Qualitative Model Evolution. IJCAI, 1311–1316 (1991)
3. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
4. Ferreira, C.: Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. Complex Systems 13(2) (2001)
5. Kuipers, B.: Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge. MIT Press, Cambridge (1994)
6. LJung, L.: System Identification Theory for the User. Prentice Hall, USA (1999)
7. Khoury, M., Guerin, F., Coghill, G.M.: Finding semi-quantitative physical models using genetic programming. In: The 6th annual UK Workshop on Computational Intelligence, Leeds, 4-6 September, 2006, pp. 245–252 (2006)