Modified Clonal Selection Algorithm for Learning Qualitative Compartmental Models of Metabolic Systems

Wei Pang and George M. Coghill*
Department of Computing Science
University of Aberdeen
Scotland, United Kingdom, AB24 3UE
{wpang,gcoghill}@csd.abdn.ac.uk

ABSTRACT

In this paper, a modified Clonal Selection Algorithm (CSA) is proposed to learn qualitative compartmental models. Different from traditional AI search algorithm, this population-based approach employs antibody repertoire to perform random search, which is suitable for the ragged and multi-modal landscape of qualitative model space. Experimental result shows that this algorithm can obtain the same kernel sets and learning reliability as previous work for learning the two-compartment model, and it can also search out the target model when learning the more complex three-compartment model. Although this algorithm does not succeed in learning the four-compartment model, promising result is still obtained.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Induction*; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*

General Terms

Design, Theory

Keywords

Qualitative Model Learning, Artificial Immune System, Clonal Selection Algorithm

*The first author is also a lecturer in College of Computer Science and Technology, Jilin University, P.R.China, 130012 email: pangwei@jlu.edu.cn.

This work is partially supported by Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, P.R.China

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

1. INTRODUCTION

1.1 Qualitative Model Learning

Qualitative Model Learning (QML), as a branch of system identification, plays an important role in the fields of biology and physics. It involves extracting the qualitative structures (namely Qualitative Differential Equations, QDEs) of the systems from given qualitative data, which are often incomplete and imprecise. So it can be viewed as the inverse of Qualitative Simulation such as QSIM [10].

Some related research in this field has been done during the last two decades, such as GENMODEL [8], MISQ [12], QSI [14] and more recently, QSI-ILP [5]. However, the above systems have different limitations. GENMODEL cannot introduce hidden variables and perform dimensional analysis. QSI and MISQ often generate over-constrained models. None of these systems except QSI-ILP has performed systematic experiments which include conditions of all the subsets of complete data.

Furthermore, All of the above are based on straightforward logic reasoning approaches, which mainly depend on domain knowledge to find models. All these systems are based on QSIM representation, which cannot deal with fuzzy data easily. None of them except QSI-ILP has produced causally ordered models [9], which are physically or biologically meaningful.

1.2 Clonal Selection Algorithm

Inspired by the clonal selection theory [7] of immune system, the clonal selection algorithm [6] can be considered as a population-based evolutionary algorithm. CSA advocates that the accumulated blind mutation and affinity-based selection can lead to better solutions. Instead of two main genetic operators: crossover and mutation in Genetic Algorithm (GA), CSA uses hyper-mutation and re-selection to fulfill the search task. Classical GA usually tends to converge to a unique solution, which may be a local optimum, while CSA is particularly suitable for searching in a multi-modal problem space. Each individual (antibody) is in fact a local optimum searcher, and the hyper-mutation can be seen as a process of blind exploration in the search space. In the re-selection process, the proliferated antibodies are selected according to their affinity (fitness), and this is a greedy strategy. After certain generations, the antibodies will find different optima, probably including the global optimum.

In [4, 5], the well-posed model is defined by satisfying several criteria, such as the completeness, connection, causal or-

QDE	JMorven Differential Plane 0
f12=M+(c1)	func (dt 0 f12) (dt 0 c1)
fo=M+(c2)	func (dt 0 fo) (dt 0 c2)
q1=u - f12	sub(dt 0 q1)(dt 0 u)(dt 0 f12)
q2=f12-fo	sub(dt 0 q2)(dt 0 f12)(dt 0 fo)
c1'=M+(q1)	func(dt1 c1) (dt0 q1)
c2'=M+(q2)	func(dt1 c2) (dt0 q2)

Table 1: QDE and JMorven Description for CM2

dering. Given the training data and background knowledge about the model, there may be many well-posed models, and the target model is among them. In previous work [5], a simple generate-and-test strategy is used to learn the target model, that is, first a traditional search strategy (branch-bound algorithm) is used to find all the well-posed models systematically, then these models are tested for coverage by a qualitative reasoning engine. This approach works well when dealing with small-sized models, the completeness and correctness of the algorithm can be guaranteed. But when the model size and the number of variables are increased, resulting in the dramatically increasing in search space, traditional AI approach (such as branch-bound) will not be very efficient. This becomes one of the motivations to use CSA as an alternative search strategy.

2. MODEL REPRESENTATION

2.1 JMorven

In this paper, a more flexible qualitative reasoning engine, JMorven [2], is used to represent and verify qualitative models. JMorven, a Java implementation of the Morven framework [3], possesses all the benefits of QSIM and introduces many new features. The introduction of differential planes [15] and vector envisionment [11] make it possible to reason about more than two derivatives. By introducing fuzzy theory, JMorven uses fuzzy quantity spaces to specify the variables, and can perform fuzzy vector envisionment [3], which enables itself to deal with fuzzy qualitative data. All the above advantages make JMorven a suitable choice as a model representation and verification component in our work.

2.2 Compartmental Models of Metabolic Systems

Metabolic systems are often modeled by compartmental models (See Figure 1). "c1", 'c2", "c3" and "c4" are the concentration in the compartment, "f12", "f23" and "f34" denote the flow from one compartment to another, "u" and "fo" are the input and output flow respectively. In the two-compartment model, if "u" and "fo" do not exist, the model becomes a coupled closed system, denoted as model CM1 in this paper. CM2 is defined in a similar way. CM2-EX3 and CM2-EX4 are the extensions of CM2 with three and four compartments respectively. Table 1 shows the QDE and JMorven representation (0th differential plane) for CM2.

The "func" symbol in Table 1 denotes the Function constraint in JMorven. JMorven extends the M+ and M- constraint in QSIM by introducing a more general function constraint, in which two variables can have arbitrary mappings.

In order to simplify the problem and compare our work

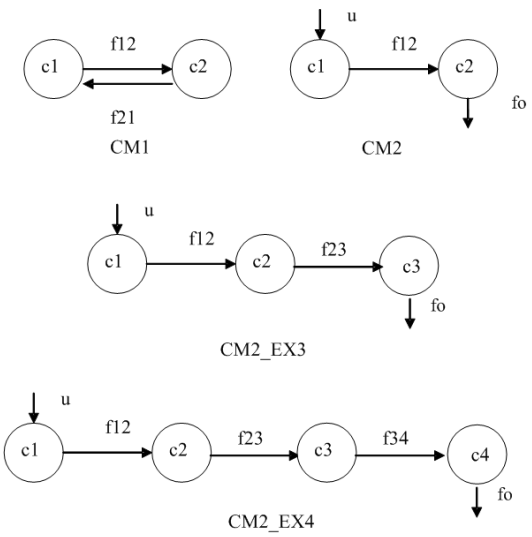

Figure 1: Compartmental Metabolic Models

Increase Mappings		Decrease Mappings	
neg	neg	neg	pos
zer	zer	zer	zer
pos	pos	pos	neg

Table 2: Increase and Decrease mappings

with previous ones, some assumptions, similar to those in [5], are imposed upon the models. That is, the compartmental models in our work are linear systems with constant coefficients in function relationship. The fuzzy quantity space of any variable includes only three values: negative, zero and positive. One reasonable quantity space is shown in Table 3. For all the observed variables, only the 0th derivative (magnitude) and the first derivative (change of direction) can be measured qualitatively. All the function relationship has the corresponding value (zer, zer), and there are only two kinds of function mappings as shown in Table 2. The models can be causally ordered, and are in canonical form as described in [9]. For simplicity and clarity, the rest of this paper will refer to *Inc* and *Dec* as the function constraints which have the increase and decrease mappings in Table 2.

2.3 One and a Half Differential Plane

Based on the above assumptions, a JMorven representation with "one and a half" differential planes is adopted to represent the models. Here "one and a half" means only the constraints in the 0th differential plane and part of the constraints in the 1st differential plane can be used to rep-

Quantity Name	a	b	alpha	beta
neg	-100	-1	0.5	0.5
zer	-1	1	0.5	0.5
pos	1	100	0.5	0.5

Table 3: Quantity Space

Differential Plane 0
C1: Inc (dt 0 f12)(dt 0 c1)
C2: Inc (dt 0 fo)(dt 0 c2)
C3: sub (dt 0 q1)(dt 0 u) (dt 0 f12)
C4: sub (dt 0 q2)(dt 0 f12)(dt 0 fo)
C5: Inc (dt 1 c1)(dt0 q1)
C6: Inc (dt 1 c2)(dt0 q2)
Differential Plane 1
C7: Inc (dt 1 f12)(dt 1 c1)
C8: Inc (dt 1 fo) (dt 1 c2)
C9: sub (dt 1 q1)(dt 1 u)(dt 1 f12)
C10:sub (dt 1 q2)(dt 1 f12)(dt 1 fo)

Table 4: JMorven Model for CM2

resent the model. In the 1st differential plane, the constraints which contain the 2nd derivative of a variable cannot be used, because only the information about 0th and first derivative of a variable are available. This form is equivalent to QSIM description for the purpose of comparison. Notice that M+ and M- are implemented by two function constraints in different differential planes: the corresponding values can be obtained from the mappings of the corresponding function constraint in the 0th differential plane, and the function constraints in the 1st differential plane determine the monotonically increasing or decreasing relation between two variables.

For example, the CM2 model is described in Table 4. In this description, Constraint C1 in the 0th differential plane and C7 in the 1st differential plane are equivalent to the constraint M+(c1 f12) in QSIM (Note the position difference). The following two constraints in the 1st differential plane are abandoned for the above mentioned reason:

C11: Inc (dt 2 c1)(dt1 q1)
C12: Inc (dt 2 c2)(dt1 q2)

3. BACKGROUND KNOWLEDGE

Before introducing the algorithm, some preliminary knowledge has to be described in detail. The constraints involved in this section are only in the 0th differential plane.

3.1 Some Concepts about Qualitative Modeling

The state variables in a causally ordered system are the variables that are directly affected by the integration operation, and is usually the output of the integrator.[15] Simply speaking, in a model with the canonical form, only the state variable can have a first derivative. The magnitude of a state variable cannot appear on the left side of any equation in the model. The exogenous variables are those variables determined from outside the model. All the non-exogenous variables are also called system variables.

In an experiment, the hidden variables are the unmeasured variables which lose both range and dimensional information. The number of hidden variables is often unknown, but it is reasonable to specify a maximum possible number of hidden variables. If the maximum number is less than the number of actual hidden variables, only "shallow" models will be induced; otherwise, unnecessary "deep" models may be found.

The model size in this paper is referred to as the number of the constraints in the model. The specification of model size is another factor that can influence the learning of the models.

In this paper, additional definitions about qualitative models are defined in order to illustrate the experimental result:

Complete Data: The qualitative states that are obtained from the complete envisionment [2].

Training Data: part of the complete data given in an experiment.

Cover_T model: Well-posed Models that can cover the given training data

OverGeneral_T model: Is a *Cover_T model*, and the data set it generates is a superset of the training data.

Cover_C Model: Is an *OverGeneral_T* model, and the data it generates include all the complete data.

OverGeneral_C Model: Is a *Cover_C* Model model, and the data set it generates is a superset of the complete data.

3.2 Inconsistent Constraints

An inconsistent constraint is a constraint that is inconsistent with the training data and consequently fails to pass the consistency check. The consistency check module employed here is the same as the one in JMorven, which uses the fuzzy interval algebraic operations. For example, constraint *X=Y-Z* is an inconsistent constraint when the current training data includes the following qualitative state: (X,Y,Z)=(pos, neg, pos),The quantities of "pos" and "neg" are taken from table 3.

3.3 Conflict Constraints

3.3.1 Conflict between two constraints

Two qualitative constraints *C1* and *C2* are conflicting if they fall into any of the following three conditions.

a. Logical Conflict: For example, the following two constraints are logically conflicting:

C3.1: Inc (dt 1 X) (dt0 Y)
C3.2: Dec (dt 1 X) (dt0 Y)

b. Redundancy: For example, the following two constraints:

C3.3: Inc (dt 0 X) (dt0 Y)
C3.4: Inc (dt 0 Y) (dt0 X)

C3.3 and C3.4 are actually describing the same relation if the causal ordering is ignored. It will be redundant if both of them appear in one model, and also the system can not be causally ordered.

Another example is:

C3.5: Sub (dt 0 a)(dt 0 b) (dt 0 c)
C3.6: Sub (dt 0 c)(dt 0 b) (dt 0 x)

x can be any variable in the model. The above two constraints can be substituted by a simpler relation: a=x, so they can not both appear in the same model.

c. dimensional inconsistence: The following condition is an instance of dimensional conflict:

C3.7: Sub (dt 0 Hid0)(dt 0 a) (dt 0 b)
C3.8: Sub (dt 0 c)(dt 0 Hid0) (dt 0 d)

Suppose both of these two constraints are dimensionally consistent individually, and the dimension of a and b is different from that of c and d. Hid0 is a hidden variable with undefined dimension. The confliction occurs because Hid0

can only have one dimension, either the same as a and b, or c and d.

3.3.2 Conflict Set of a Constraint

After the preprocessing phase of the algorithm, which will be introduced later, a candidate constraint set is obtained, denoted as ***FCS***. If $C1 \bowtie C2$ is used to represent that *C1* and *C2* are conflicting, the conflict set for a constraint *C1* can be defined as:

$$ConflictSet(C1) = \{C_i | C_i \in \boldsymbol{FCS}, C1 \bowtie C_i\}$$

3.3.3 Conflict involved more than two constraints

The conflict may involve more than two constraints, for example,

Inc(Hid0, Hid1), Hid0=a-b, a=Hid1-d.

Here the hidden variables Hid0 and Hid1 have the same dimension derived from the second and third constraint, resulting in no physical meaning for the first function constraint. Because the corresponding equation for the first constraint is:

Hid0= k* Hid1

In this equation, k must have a dimension if we make the assumption that there is no gain or amplifier in the system under study. So the dimension of Hid0 and Hid1 cannot be the same.

3.4 Defining Constraints and Search Space Partition

3.4.1 Defining Constraint

The defining constraint for a variable with specified derivative (or variable/derivative for short) is the constraint in which the variable/derivative appears in the leftmost position.

For instance, constraint sub (dt1 X) (dt0 Y) (dt0 Z) is one defining constraint for the first derivative of variable X. All derivatives of an exogenous variable and 0th derivative of a state variable do not have defining constraints.

3.4.2 Referring Constraint

The referring constraint of a variable/derivative is the constraint in which the variable/derivative appears in any position except the leftmost position.

For example, Sub(dt0 Y)(dt0 X)(dt0 Z) is a referring constraint for both 0th derivative of variable X and 0th derivative of variable Z.

3.4.3 Dependency Set of a Constraint

For a certain variable/derivative, all its referring constraints depend on its defining constraints in a causal ordering context. If constraint C1 depends on C2, then this relation is denoted as follows:

$$C1 \rightarrow C2$$

Suppose the candidate constraint set is ***FCS***, the dependency set for a constraint C1 is defined as:

$$Dependency(C1) = \{C_i | C_i \in \boldsymbol{FCS}, C1 \rightarrow C_i\}$$

For example, constraint sub (dt0 X)(dt0 Y)(dt0 Z), the dependency set of this constraint may contain the following constraints:

Inc (dt0 Y)(dt0 A)

Dec (dt0 Z)(dt0 B)

In a causally ordered model, a constraint cannot appear before any of its dependency constraints, because only after the defining constraint of a variable/derivative appears, can other constraints refer to this variable/derivative.

Theorem 3.1

Based on all the assumptions we have made upon the models, in the 0th differential plane, a well-posed model defined in [5] must include one and only one defining constraint for each of the system variables with either 0th or first derivative.

Proof: Suppose X is a non-exogenous variable in the model. If X is a state variable, according to the definition of state variable, there must be a defining constraint for the first derivative of X. If X is not a state variable, and the model does not include any defining constraint for the zero derivative of X, then no referring constraints for X can be included in the model, resulting in the exclusion of X from the model. This is contradictory considering the completeness principle of well-posed models, stating that the model must include all the system variables. So a well-posed model must include at least one defining constraint for each of the system variables.

On the other hand, if a model includes more than one defining constraint for the same variable, it also cannot be causally ordered. Consequently Theorem 3.1 is sound.

Corollary 3.1

The model size of a target model equals to the number of system variables (including hidden variables) in the model.

4. ALGORITHM DESCRIPTION

4.1 Preprocessing Phase

First we introduce the preprocessing phase of the algorithm, this includes four modules: Constraint Generation, Constraint Filtering, Calculation of Conflict Set and Dependency Set, and Constraint Set Partition.

4.1.1 Constraint Generation

The constraint generation is similar to GENMODEL [8] except performing an additional dimensional check [1]. In this phase, given all the observed variables, maximum number of possible hidden variables, maximum number of derivatives for each variable (2 in our problem), range and dimension (if available) for each derivative, and all possible constraint types (Subtract, Inc and Dec in this paper), the constraint generator will generate all the possible constraints, denoted as Initial Candidate Constraint Set (***ICCS***).

4.1.2 Constraint Filtering

Second, all the constraints in ***ICCS*** will be checked for consistency by the constraint filter. The inconsistent constraints defined in Section 3.1 will be filtered out. After this phase, a filtered constraint set (***FCS***) is obtained. Given complete behaviors of the systems, ***FCS*** will have the minimum size; otherwise, the size of ***FCS*** may be very large because some inconsistent constraints may not be filtered out provided incomplete data.

4.1.3 Calculation of Conflict Set and Dependency Set

In this phase, for each constraint in ***FCS***, we calculate the conflict set (Section 3.3.3) and dependency set (Section 3.4.3) and store the result into two matrixes: *ConflictMatrix* and *DependencyMatrix*. They will be used for later CSA.

4.1.4 Constraint Set Partition

FCS is divided into several subsets, each of these subsets contains all the defining constraints for the same variable. If S_i is the subset containing the defining constraints for a hidden variable, an "empty" constraint ϕ is appended on this subset: $S_i = S_i \bigcup \{\phi\}$. ***DS*** is a set that takes each of these subsets as an element, denoted as ***DS***$= \{S_n\}$ (n=1 to N) N is the number of variables (including hidden variables). For example, in the CM2 model, a subset for variable f12 may contain the following constraints:

Inc (dt0 f12) (dt0 c1)
Dec (dt1 f12) (dt0 c2)
Sub (dt0 f12) (dt0 fo)(dt0 u)

4.2 Clonal Selection Algorithm for Searching the Model Space

The basic idea is that each antibody is encoded to represent a potential model, and in the process of evolutionary search the antibodies are expected to find the target models.

It should be pointed out that not all the antibodies can find the target models. Some antibody may find a local optimum (for example, a well-posed model cannot cover all the training data), and stay at that position for a intolerable long time, because it cannot find its neighbor with better affinity. To avoid the unwanted domination of these antibodies in the antibody repertoire, each antibody is attached with a survival time. If an antibody exceeds its survival time, it will be replaced by a randomly generated antibody. This can enhance the search ability of the algorithm.

To save computational cost, all the well-posed models found during the search are recorded into the memory pool, which is implemented by a hash set. This can avoid repeated computation.

A model repository is constructed to store all the candidate models found by the algorithm, including the *Cover_T*, *OverGeneral_T*, *Cover_C* and *OverGeneral_C* models. This repository is updated by the newly found candidate models in each generation.

4.2.1 Antibody Encoding

The intuitive integer encoding method is adopted in CSA. The antibody is composed of several slots, each of which corresponds to a different S_i in ***DS***, which is introduced in Section 4.1.4. Each slot will be assigned to an integer number, which indicates a qualitative constraint selected from different S_i in ***DS***. The correctness of this encoding strategy is guaranteed by Theorem 3.1 and Corollary 3.1. Figure 2 shows an example of antibody encoding for CM2. In this antibody, S_0, S_1, S_2 and S_i contains all the defining constraints for "f12", "fo", "c1" and "Hid0", respectively.

Note there is an "empty" constraint in the last subset S_i, because this subset contains all the defining constraints for the hidden variable Hid0. The "empty" constraint introduced here is to deal with the redundant hidden variables. When the number of maximum possible hidden variables is greater than that of the hidden variables the system actually has, some generated hidden variables cannot be introduced to the system. For example, a system has two hidden variables, but the maximum possible number of hidden variables is 3, resulting in the generation of three subsets in ***DS*** for these three hidden variable. The target model in fact chooses constraints from only two of these subsets; for another subset, the target model will select the "empty" constraint.

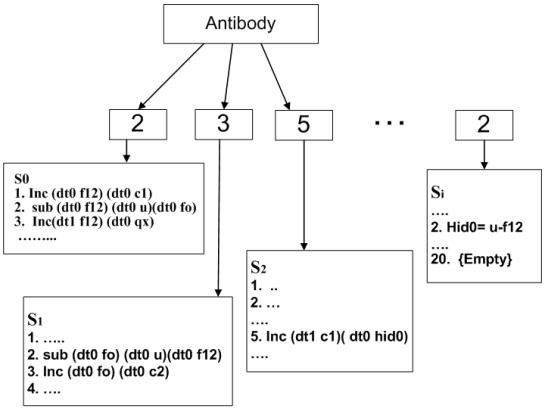

Figure 2: Antibody Encoding for CM2

4.2.2 Hyper-mutation

Each antibody in the repertoire will be cloned to several copies. All the cloned antibodies, which form a temporary population, will undergo the hyper-mutation scheme.

The hyper-mutation is also modified due to the modification of the antibody encoding. That is, for each slot of the antibody, its value will be replaced by a randomly generated integer with a high probability. The range of this randomly generated integer is from 1 to N, N is the number of the constraints in the corresponding subset S_i. This means each constraint in the model will be replaced by another constraint in the same defining subset S_i. Each mutated antibody can be seen as a neighborhood of the original antibody.

4.2.3 Affinity Evaluation

The following criteria, similar to the one in [5], are set up to evaluate the model which an antibody represents:

a. *Model Confliction*: This module will calculate the number of conflicting constraints in the model. The calculation is based on the *ConflictMatrix*, which is obtained from the preprocessing phase.

b. *Dimensional Consistency*: This module can detect the dimensional inconsistency described in Section 3.3.4.

c. *Model Language*: Check whether the model has included all types of the constraints.

d. *Model Connection*: Check whether a model is connected, a model is connected if all intermediate (hidden) variables appear in at least two qualitative constraints.

e. *Model Completeness*: Check whether the model has included all the variables provided by the training data.

f. *Model singularity*: No disjoint sub-model in the model.

g. *Causal Ordering*: Whether the model can be causally ordered. In our work, Iwasaki's causal ordering algorithm [9] is slightly modified to check the causal ordering of a model.

The dependency relation among constraints can be obtained from the *DependencyMatrix*, which is constructed during the preprocessing phase.

h. *Coverage*: Check whether the model can cover all the training data. The JMorven package is tailored and slightly modified as an embedded module to support the model coverage test.

If a model satisfies criteria a⌣g, it is referred to as a well-posed model. Because of the relatively expensive computational cost of JMorven simulation, the model coverage test h is arranged in the final stage, only the well-posed models are allowed to be simulated by JMorven.

The above criteria establish a scoring system for evaluating the affinity of an antibody. The more criteria a model satisfies, the higher score it will get. For example, for the *Model Confliction* criterion, if there is no conflicting constraint in the model, a full weighted score will be added to the affinity value. Otherwise, the score can be calculated as follows:

$$ConflictScore = W1 * (1 - NumberOfConflicting/ModelSize)$$

In the above formula, W1 is the weight, *NumberOfConflicting* indicates the number of conflicting constraints in this model. The scores for other criteria can be calculated in a similar way. The weights for all the criteria are currently the same.

4.2.4 Re-selection

The re-selection process is based on the above-mentioned scoring system. The antibodies, which have higher scores and have not exceeded the survival time, will be selected to form a new generation of population.

4.3 Pseudo Code of The CSA Algorithm

The pseudo code is quite straightforward and the framework of the algorithm is basically the same as the original CSA. Notice the stop criteria of the algorithm depend on different experiments, which will be described later.

Step 1: Initialization

parameter setting: The hyper-mutation probability ρ; maximum running time: *MaxTime*; maximum generation: *MaxGen*; population size: *PopSize*; clonal size: *ClonalSize*; survival time for each antibody: *SurviveTime*.

Repertoire Initialization: Randomly generating *PopSize* antibodies to construct the initial antibody repertoire (population). Initializing the *modelrepository*.

Step 2: Evolutionary Iteration

While (Stop Criteria not satisfied), iterate the following steps:

Step2-1 Selection: All the antibodies in the population are selected for further operations.

Step2-2 Clonal Expansion: Each antibody in the population are cloned for *ClonalSize* copies, and all these copies are stored to a temporary population *tempPop*.

Step2-3 Hyper-Mutation: All the antibodies in the temporary population undergo the hyper mutation process. Note, we will keep one copy unchanged for each of the original antibodies.

Step2-4 Affinity Evaluation: for each antibody in *tempPop*, calculate the affinity.

Experiment ID	Given Variable	Hidden Variable	Specified State Variable
CM1	c1, c2, f12, f21	qx	None
CM2	c1, c2, f12, fo, u	q1, q2	None
CM2-EX3-E1	c1, c2, c3, f12, f23, u, fo, q1, q2	q3	c1, c2, c3
CM2-EX3-E2	c1,c2, c3,f12, f23,u,fo, q1	q2, q3	c1, c2, c3
CM2-EX3-E3	c1,c2, c3, f12,f23,u,fo	q1,q2,q3	c1, c2, c3
CM2-EX4	c1, c2, c3,c4, f12, f23, f34, fo, u	q1, q2 q3, q4	c1, c2 c3, c4

Table 5: Experimental Conditions

Step2-5 Update Model Repository: Record the newly found *Cover_T*, *Cover_C*, *OverGeneral_T* and *OverGeneral_C* models during the evaluation process.

Step 2-6 Reselection: After evaluation, *PopSize* best antibodies are selected from the *tempPop*, forming a new generation antibody repertoire. If an antibody's surviving time exceeds the *SurviveTime*, it will be replaced by a new randomly generated antibody.

5. EXPERIMENTAL RESULTS

The reliability and scalability of the algorithm are tested by different experiments. The conditions of all the experiments are described in Table 5. For simplifying the problem, in all the experiments, the input u is supposed to {pos, zer}, and the maximum possible number of hidden variables is assumed to equal to the actual hidden variables.

5.1 Reliability Test

For simple models with small number of training data, CM1 and CM2, the learning reliability of the algorithm is tested by providing all the subsets of the complete data to test the experiments. So there are $2^N - 1$ experiments, N is the number of the complete data. Each of these experiments takes one of the subsets of the complete data as training data. Thus, the kernel set and learning curve, defined in [5], can be obtained. The traditional AI search algorithm will also be performed to test the correctness of the result.

For CM1 and CM2, the repertoire sizes are 10 and 100 respectively, and the clonal size is 10. The survival time of the antibody is 100 generations. There are two stop criteria for each of the subset training data experiments: One is the running time exceeds 60 seconds; the other is that the algorithm searches out a model that is a *Cover_T* model, but is not the target model (This means that we cannot discriminate the target model from other *Cover_T models*). The algorithm will stop if either of the above is satisfied. The second criterion is actually a fail-fast strategy, resulting in saving computation time.

Figure 3 shows the learning reliability of the algorithm in CM1 and CM2 experiments. For CM1, The elements in the kernel set are all pairs:

(1,2) (1,3) (1,5) (2,3) (2,4)

The number in the pairs stands for the State ID in the complete envisionment, which is shown in Appendix A. This means for learning CM1, the above pairs and all subsets

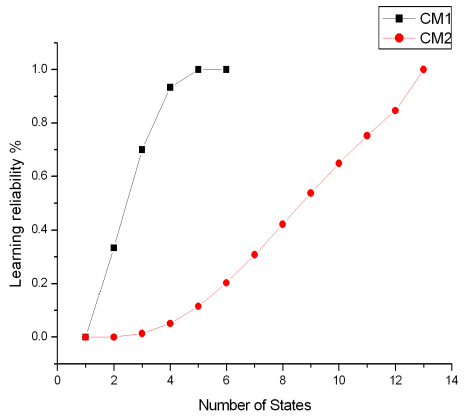

Figure 3: Learning Reliability of CM1 and CM2

including these pairs can successfully learn the target model. For CM2, The kernel set found is as follows:

(0,2,5) (0,2,7) (0,2,11) (0,2,13) (0,4,7)
(0,4,5,6) (0,4,6,11) (0,4,6,13)

A traditional AI search algorithm, backtracking algorithm with forward checking [13], is also performed on learning CM1 and CM2, indicating that the above kernel set and learning reliability are correct.

Figure 4 and Figure 5 shows two example experiments for CM1 and CM2 experiments with two sates. The qualitative states in these two experiment are both (0, 1). These two experiments are forced to run 80 and 1000 generations respectively, to demonstrate the evolutionary search process of the algorithm.

5.2 Scalability Test

For CM2-EX3 and CM2-EX4, the scalability of the algorithm is tested by providing complete data but with different hidden variables.

For all the experiments in CM2-EX3 and CM2-EX4, the hyper-mutation probability is 0.5. Clonal Size is 10. The stop criterion of the algorithm is: The algorithm finds at least one *Cover_T* model. The complexity of the experiments and other parameter settings are shown in Table 6.

In CM2-EX3-E1, E2 and E3, there are 68, 48, and 48 qualitative states in the complete envisionment respectively. In CM2-EX4, there are 164 states. The CSA can find one solution in all the three CM2-EX3 experiments, and the solution it finds is both *Cover_T* and target model. For CM2-EX4, the algorithm cannot find the any *Cover_T* models after 42,122 seconds, 100,000 generation. This is because the complexity of the model, but we still obtain 988 well-posed models.

Experiment ID	Search Space Size	Repertoire size	Survival Time (Gen)	First Solution Found (Gen)
CM2-EX3-E1	1.06×10^8	100	100	5217
CM2-EX3-E2	1.31×10^{10}	1000	1000	201368
CM2-EX3-E3	2.31×10^{11}	1000	1000	131624
CM2-EX4	3.65×10^{17}	1000	1000	N/A

Table 6: parameter setting and Model Complexity

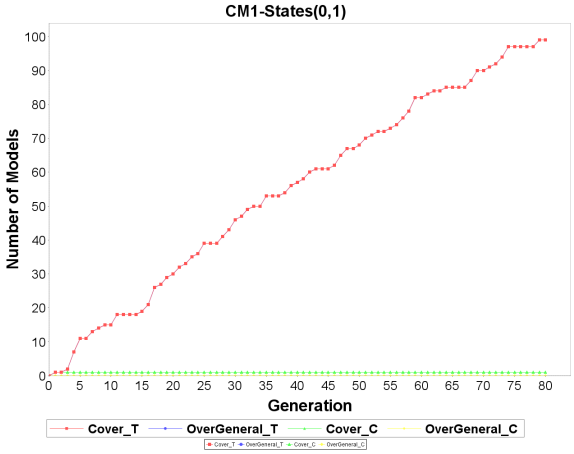

Figure 4: A Typical Run for CM1

5.3 Conclusions and Future Work

In this paper, an alternative approach is proposed to deal with the same problem as in [4] and [5]. The contributions of our work are twofold:

First, a modified clonal selection algorithm is proposed to learn the qualitative compartmental models. The reliability and scalability of the algorithm are evaluated. Experimental results indicate our algorithm can achieve the same reliability and obtain the same kernel sets as the traditional AI algorithm. Experiments on more complex models shows that the algorithm can also get promising results. Second, JMorven, a more flexible model representation is adopted. This is not only an alternative representation method, but also has the potential ability to deal with fuzzy data and reason about more than two derivatives.

Future works will involve the followings:

First, local search can be integrated into the algorithm. The convergence speed of pure evolutionary algorithm will slow down when dealing with extreme complicated problems (CM2-EX4, for example), and local search can accelerate the search process. One possible local search strategy is that each antibody is refined by traditional AI search algorithms (such as backtracking method) in a limited computational cost.

Second, the precision of the model can also be improved by adding more quantities in the quantity space. Currently only three quantities are used. Under this circumstance, the learning task will become more challenging.

6. REFERENCES

[1] R. Bhaskhar and A. Nigam. Qualitative physics using

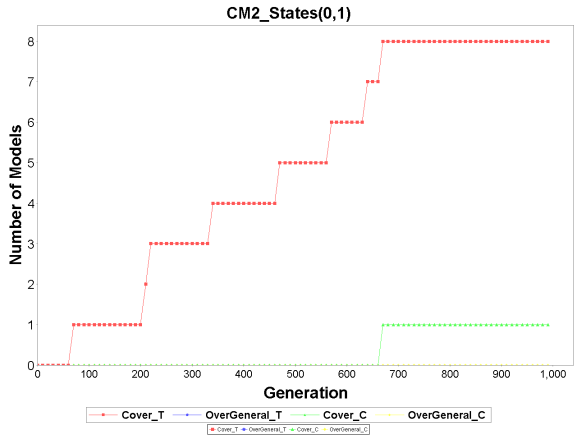

Figure 5: A Typical Run for CM2

dimensional analysis. *Artificial Intelligence*, 45:73–111, 1990.

[2] A. M. Bruce and G. M. Coghill. Parallel fuzzy qualitative reasoning. In *Proceedings of the 19th International Workshop on Qualitative Reasoning*, pages 110–116, Graz, Austria, 2005.

[3] G. M. Coghill. *Mycroft: A Framework for Constraint based Fuzzy Qualitative Reasoning*. PhD thesis, Heriot-Watt University, September 1996.

[4] G. M. Coghill, S. Garrett, and R. D. King. Learning qualitative metabolic models. In *European Conference on Artificial Intelligence (ECAI'04)*, pages 445–449, Valencia, Spain, 2004.

[5] G. M. Coghill, S. M. Garrett, A. Srinivasan, and R. D. King. Qualitative system identification from imperfect data. Technique Report AUCS/TR0501, Department of Computing Science, University of Aberdeen, 2004.

[6] de Castro and V. Zuben. The clonal selection algorithm with engineering applications. In *Proceedings of GECCO,Workshop on Artificial Immune Systems and Their Applications*, pages 36–39, Las Vegas, USA, July 2000.

[7] F.Burnet. *The Clonal Selection Theory of Acquired Immunity*. Cambridge University Press, Cambridge, 1959.

[8] D. T. Hau and E. W. Coiera. Learning qualitative models of dynamic systems. *Machine Learning*, 26:177–211, 1993.

[9] Y. Iwasaki and H. A. Simon. Causality in device behavior. *Artificial Intelligence*, 29:3–32, 1986.

[10] B. Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, Cambridge, MA, 1994.

[11] A. Morgan. *Qualitative behaviour of Dynamic Physical Systems*. PhD thesis, University of Cambridge, 1988.

[12] B. L. Richards, I. Kraan, and B. Kuipers. Automatic abduction of qualitative models. In *National Conference on Artificial Intelligence*, pages 723–728, 1992.

[13] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach 2nd Edition*, chapter 5, pages 142–146. Prentice Hall, 2003.

[14] A. C. Say and S. Kuru. Qualitative system identification: deriving structure from behavior. *Artificial Intelligence*, 83:75–141, 1996.

[15] M. Wiegand. *Constructive Qualitative Simulation of Continuous Dynamic Systems*. PhD thesis, Heriot-Watt university, May 1991.

7. APPENDIX A

The JMorven Model for CM1:

Differential Plane 0
C1: Inc (dt 0 f12)(dt 0 c1)
C2: Inc (dt 0 f21)(dt 0 c2)
C3: sub (dt 0 qx)(dt 0 f12) (dt 0 f21)
C4: Inc (dt 1 c1)(dt0 qx)
C5: Dec (dt 1 c2)(dt0 qx)

Differential Plane 1
C6: Inc (dt 1 f12)(dt 1 c1)
C7: Inc (dt 1 f21)(dt 1 c2)
C8: sub (dt 1 qx)(dt 1 f12)(dt 1 f21)

Complete Fuzzy Vector Envisionment for CM1. c1={pos, neg} means that the zero derivative of c1 is "positive" while the first derivative of c1 is "negative".

State ID	c1	c2	f12	f21
0	{zer , zer}	{zer , zer}	{zer , zer}	{zer , zer}
1	{zer , pos}	{pos , neg}	{zer , pos}	{pos , neg}
2	{pos , neg}	{zer , pos}	{pos , neg}	{zer , pos}
3	{pos , zer}	{pos , zer}	{pos , zer}	{pos , zer}
4	{pos , pos}	{pos , neg}	{pos , pos}	{pos , neg}
5	{pos , neg}	{pos , pos}	{pos , neg}	{pos , pos}

8. APPENDIX B

Compete Envisionment for CM2, supposing inflow u={pos, zer}

State ID	c1	c2	f12	fo
0	{zer , pos}	{zer , zer}	{zer , pos}	{zer , zer}
1	{zer , pos}	{pos , neg}	{zer , pos}	{pos , neg}
2	{pos , zer}	{zer , pos}	{pos , zer}	{zer , pos}
3	{pos , pos}	{zer , pos}	{pos , pos}	{zer , pos}
4	{pos , neg}	{zer , pos}	{pos , neg}	{zer , pos}
5	{pos , zer}	{pos , zer}	{pos , zer}	{pos , zer}
6	{pos , zer}	{pos , pos}	{pos , zer}	{pos , pos}
7	{pos , zer}	{pos , neg}	{pos , zer}	{pos , neg}
8	{pos , pos}	{pos , zer}	{pos , pos}	{pos , zer}
9	{pos , pos}	{pos , pos}	{pos , pos}	{pos , pos}
10	{pos , pos}	{pos , neg}	{pos , pos}	{pos , neg}
11	{pos , neg}	{pos , zer}	{pos , neg}	{pos , zer}
12	{pos , neg}	{pos , pos}	{pos , neg}	{pos , pos}
13	{pos , neg}	{pos , neg}	{pos , neg}	{pos , neg}