Advanced Experiments for Learning Qualitative Compartment Models

Wei Pang and **George M. Coghill**
Department of Computing Science
University of Aberdeen
Scotland, United Kingdom, AB24 3UE

Abstract

In this paper, the learning of qualitative two-compartment metabolic models is studied under the conditions of different types and numbers of hidden variables. For each condition, all the experiments, each of which takes one of the subsets of the complete qualitative states as training data, are tested one by one. In order to conduct the experiments more efficiently, a backtracking algorithm with forward checking is introduced to search out all the well-posed qualitative models as candidate solutions. Then these candidate solutions are verified by a fuzzy qualitative engine JMorven to find the target models. Finally the learning reliability and kernel set under different conditions is calculated and analyzed.

1. Introduction

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 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 (Kuipers 1994)).

Some related research in this field has been done during the last two decades, such as GENMODEL (Hau & Coiera 1993), MISQ (Richards, Kraan, & Kuipers 1992), QSI (Say & Kuru 1996) and more recently, QSI-ILP (Coghill *et al.* 2004). All these systems are based on QSIM representation. However, the above systems have different limitations. GENMODEL can not introduce hidden variables and perform dimensional analysis. QSI often generate over-constrained models. None of these systems except QSI-ILP performs systematical experiments which include conditions of all the subsets of complete data. None of these systems have analyzed the influence of different hidden variables on the learning reliability of the system.

2. Model Representation

2.1 JMorven

In this paper, a more flexible qualitative reasoning engine, JMorven (Bruce & Coghill 2005), is used to represent and verify qualitative models. JMorven, a Java implementation of the Morven framework (Coghill 1996), possesses all the benefits of QSIM and introduces many new features. The introduction of differential planes (Wiegand 1991) and vector envisionment (Morgan 1988) 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 (Coghill 1996), which enables it to deal with fuzzy qualitative data. In addition, the utilization of parallel techniques makes JMorven more efficient. All the above advantages make JMorven a better choice as a model representation and verification component in our work.

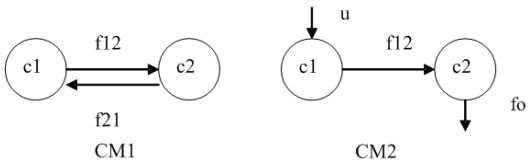

Figure 1: Two-Compartment Metabolic Models

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

2.2 Compartmental Models of Metabolic Systems

Metabolic systems are often modeled by two-compartment models (See Figure 1). In the two-compartment model, if input *u* and output *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. Table 1 shows QDE and JMorven representation (0th differential plane) for CM2.

The "func" symbol in Table 1 denotes the Function con-

Increase Mappings	
neg	neg
zer	zer
pos	pos

Decrease Mappings	
neg	pos
zer	zer
pos	neg

Table 2: Increase and Decrease mappings

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

straint 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 with previous research, some assumptions, similar to those in (Coghill *et al.* 2004), are imposed upon the models. That is, the compartment models in our work are linear systems with constant coefficients in the functional relationships. The fuzzy quantity space of any variable includes only three values: negative, zero and positive. One reasonable quantity space is shown in Table 3. The meanings of the variables "a","b","alpha" and "beta" are described in (Shen & Leitch 1993). For all the observed variables, only the zero derivative (magnitude) and the first derivative can be measured qualitatively. All the function relationships have 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 (Iwasaki & Simon 1986). 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

The concept of *differential plane* in qualitative context was first proposed in (Wiegand 1991). The zeroth differential plane constains the constraints, which can construct a model used for numerical simulation. The constraints in a higher differential plane are obtained by differentiating the corresponding constraints in the preceding differential plane.

Based on the assumptions in previous section, 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 represent the model. In the 1st differential plane, the constraints which contain the 2nd derivative of a variable can not be used, because only the information about zero 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.

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

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.

3.0 Some Concepts about Qualitative Modeling

The state variables in a causally ordered system are the variables that are directly effected by the integration operation, and is usually the output of the integrator.(Wiegand 1991) Simply speaking, in a model with canonical form, only the state variable can have first derivative. The magnitude of a state variable can not 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 number of possible hidden variables. If the maximum number is less than the number of actual hidden variables, only "shallow" models will be induced; otherwise, unnecessarilly "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.

3.1 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 we 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 current training data include the following qualitative state: (X,Y,Z)=(pos, neg, pos),The quantities of "pos" and "neg" are taken from table 3.

3.2 Conflict Constraints

Two qualitative constraints *C1* and *C2* are conflicting if they are logically or dimensionally inconsistent, or redundant so that a simpler qualitative constraint can be derived from these two constraints by algebraic operations. We use $C1 \bowtie C2$ to represent that *C1* and *C2* are conflicting. The details about conflict constraints will be illustrated by section 3.2.1 $\sim$ 3.2.4.

3.2.1 Conflict between two function constraints

Two function constraints *C1* and *C2* are conflicting if they satisfy any of the following two conditions:

- a. Logical Conflict: They have the same variable/derivatives in the corresponding positions of the constraints but they have different function mappings, i.e., one is *Inc* and the other is *Dec*
- b. Redundancy: They have the same variable/derivatives and same function mappings but the variables/ derivatives appear in different positions of the constraints.

In condition a, these two constraints are actually logically inconsistent, because in a physical or biological system, the relation between two variables can not have different mappings. For example, the following two constraints are conflicting:

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

C3.1 means the first derivative of variable X and zero derivative of Y has increasing relationship, but C3.2 means these two variables has decreasing relation, this is contradictory in logic.

In condition b, these two constraints are in fact the same constraint. Considering 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 actually describe 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 cannot be causally ordered. So C3.3 and C3.4 are conflicting constraints.

3.2.2 Conflict between subtract constraints

The detection of a conflict between two subtract constraints is more complicated than that in function constraints. After generalization, the following seven conditions are listed without considering the dimensional consistency:

a: a=b-c , c=b-x (x can be any variables in the system)
b: a=b-c, d=b-c (a, d can be any variables in the system)
c: a=b-c, d=c-b (a, d can be any variables in the system)
d: a=b-c, b=a-x (x can be any variables in the system)
e: a=b-c , c=a-b
f: a=b-c , b=c-x (x can be any variables in the system)
g: a=b-c , c=x-a (x can be any variables in the system)

The constraints in the above conditions are either contradictory or can be replaced by a simpler constraint.

Apart from the above conditions, the dimensional conflict may occur between two subtract constraints when there exist variables with undefined dimension, such as hidden variables. The following condition is an instance of dimensional conflict:

h: Hid0=a-b, c=Hid0-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 conflict occurs because Hid0 can only have one dimension, either the same as a and b, or c and d.

3.2.3 Conflict Set of a Constraint

After the preprocessing phase of the algorithm, which we will introduce later, a candidate constraint set is obtained, denoted as ***FCS***. The conflict set for a constraint C1 is defined as:

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

As we have introduced before, this conflict relation is binary which only involves two constraint.

3.2.4 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 can not be the same.

3.3 Defining Constraints and Search Space Partition

3.3.1 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 zero derivative of a state variable do not have defining constraints.

3.3.2 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 zero derivative of variable X and zero derivative of variable Z.

3.3.3 Dependency Set of a Constraint

For a certain variable/derivative, all its referring constraints depend on its defining constraints in causal ordering context. If constraint C1 depend 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:

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

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 can not appear before any of its dependency constraints, because only after the defining constraint of a variable/derivative appears, can other constraints refers 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 (Coghill *et al.* 2004) must include one and only one defining constraint for each of the system variables (zero 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 can not 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

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

4.1 Constraint Generation

Constraint generation is similar to GENMODEL (Hau & Coiera 1993) except that it performs an additional dimensional check(Bhaskhar & Nigam 1990). 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***).

4.2 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.

4.3 Calculation of Conflict Set and Dependency Set

In this phase, for each constraint in ***FCS***, we calculate the conflict set (Section 3.2.3) and dependency set (Section 3.3.3) and store the result into two matrixes: *ConflictMatrix* and *DependencyMatrix*. They will be used for later backtracking search algorithm.

4.4 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 any two elements in ***DS***, $|Si| \leq |Sj|$ if $i \leq j$. 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)

4.5 Backtracking Algorithm

The basic idea of the algorithm is for each subset S_i in ***DS***, selecting only one constraint, thus to construct a model, then checking the validity of this model. The correctness of this selection is guaranteed by Theorem 3.1.

For efficiency reasons, a backtracking tree search algorithm is adopted. The algorithm continuously adds constraints from different subsets in ***DS***, once a new constraint from a subset is added, the current partial model will be checked for validity by model checking algorithms. If this partial model fails to pass the check, we will abandon it, backtrack to previous node, and select the next node, in order to avoid searching hopeless nodes.

Notice that for each S_i which contains the defining constraints for a hidden variable, there is also an "empty" constraint ϕ in it. The current partial model will not change

if an "empty" constraint is added upon it. 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 can not be introduced to the system. For example, a system has two hidden variables, but the maximum number of possible hidden variables is 3, resulting in the generation of three subsets in ***DS*** for these three hidden variable. The target model in fact choose constraints from only two of these subsets; for another subset, the target model will select the empty constraint.

An auxiliary forward checking method (Russell & Norvig 2003) is also performed; that is, when a new constraint is added, all the constraints in ***FCS*** that are conflicting with this new constraint will be ignored in the later search process. In order to prune more sub-trees, the exploration order of the subsets in ***DS*** is determined by the number of *legal constraints* in these subsets: the subset which has the minimum number of legal constraints will be explored first. The legal constraints are all the constraints in ***FCS*** that do not conflict with any constraint in the current partial model.

4.6 Pseudo Code of The Tree Search Algorithm

Step1: **Preprocessing**
Step 1-1: Constraint Generation, get ***ICCS***
Step 1-2: Constraint Filtering, get ***FCS***
Step 1-3: Calculating Conflict Set and Dependency Set, get *ConflictMatrix* and *DependencyMatrix*.
Step 1-4: Partition ***FCS***, get ***DS***
specify model size *ModelSize*
Step2: **Backtracking Search**
Begin
PartialModel=null; // current partial model.
ExploredSubset=empty;
// record the subsets that have been explored;
LegalSet:= ***FCS***;
Backtracking_FC(PartialModel, ExploredSubset, LegalSet)
End
The Function Backtracking_FC is defined as follows:
Backtracking_FC (PartialModel, ExploredSubset, LegalSet)
a. if size of PartialModel $\geq$ *ModelSize*
then return; // the exit of the recursion
b. Select a subset $S_i \in$ *DS-ExploredSubset*,
St. min($|S_i \bigcap LegalSet|$)
c. if $S_i \bigcap LegalSet == \emptyset$ return;
// no legal constraint, exit.
d. ExploredSubset:=ExploredSubset+S_i;
e.For each constraint C1 $\in S_i \bigcap LegalSet$
Begin
CS_C1:=Conflict Set of C1;
LegalSet:= LegalSet- CS_C1;
PartialModel:=PartialModel+ C1;
if *CheckPartialModel*(PartialModel)==true then
Begin
if size of PartialModel $<$ *ModelSize*
Backtracking_FC(PartialModel, ExploredSubset,LegalSet)
Else
CheckCompleteModel(PartialModel);
End // if CheckPartialModel()==true
LegalSet:= LegalSet + CS_C1;
//Restore the legalSet
PartialModel:=PartialModel- C1;
End //for each constraint
f. ExploredSubset:=ExploredSubset- S_i;
//restore exploredSubset
End Function

4.7 Model Checking Algorithm

The model checking algorithm is divided into two parts, every time a new constraint is added into the current partial model, the partial model checking function (*CheckPartialModel()* in pseudo code) will be performed on the current partial model. This checking algorithm will quickly check whether this partial model is consistent. There are two submodules in this function:

1. Contradictory Check: Checking whether current partial model contains conflict constraints based on the *ConflictMatrix*.

2. Dimensional Consistency Check: Checking the situation in section 3.2.4, in which the conflict involves more than two constraints.

Another model checking module, *CheckCompleteModel()*, checks the other properties of well-posed models, as stated in (Coghill *et al.* 2004), including model language, model connection, model completeness, singularity, connection, causal ordering and model coverage, will be only performed on "full models" which attain the pre-specified model size.

The causal ordering check is based on the *DependencyMatrix*. The JMorven package is tailored and slightly modified as an embedded module to support the model coverage test. Because of the relatively expensive computational cost of JMorven simulation, the model coverage test is arranged in the final stage, only well-posed models which pass all the other model checking modules are allowed to be simulated by JMorven.

5. Experimental Results

Using the above efficient algorithm, the learning of CM1 and CM2 models are throughout tested. We focus on the influence of hidden variables and incomplete training data on the learning reliability.

5.1 Experimental Methodology

For each model, We will start from the easiest experiment, which is given maximum number of variables and complete data. If it succeeds, we will conduct more difficult experiments, categorized by the following conditions: losing nonderivative variables, partial or not specifying the state variables, and losing the derivative variables.

For each of the above conditions, first complete training data, obtained from JMorven's complete envisionment, will be provided. If our algorithm can find the target model, the

Experiment ID	Hidden Variable	Known State Variable	Success
CM1-E1	qx	c1, c2	Yes
CM1-E2	qx, f12	c1,c2	Yes
CM1-E3.a	qx,c1	c2	No
CM1-E3.b	qx,c1	c2	No
CM1-E4.a	qx,c1	c2,Hidden	No
CM1-E4.b	qx,c1	c2,Hidden	Yes *
CM1-E5	qx, f12,f21	c1,c2	Yes
CM1-E6	qx,f12, c1	c2,Hidden	No
CM1-E7	qx	None	Yes
CM1-E8	qx,f12	None	No

* under additional domain-specific knowledge

Table 5: Experimental Conditions for CM1

experiment will be tested by providing all the elements in the power set of the training data, and get the learning reliability from these result.

For CM1, there are 6 qualitative states, so there will be 2^6 =64 experiments in each different condition, the computation cost is tolerable. For CM2 there are 14 qualitative states in the complete envisionment, as shown in Appendix B. So there will be 2^{14} =16,384 experiments. This will be very computational expensive.

In order to accelerate the calculation, we take the following approach: instead of finding the well-posed models for different training data in each experiment separately, we will first search all the well-posed models only once, discarding the training data. Then in each experiment, the set of well-posed models will be narrowed by different training data. So the search for well-posed models are only executed once and the results are used 2^n times (n is the number of qualitative states). This approach can be done easily by modifying the original tree search algorithm:

a. First disabling the constraint filtering function (step 1-2) in the preprocessing phase and the coverage test in the *CheckCompleteModel()* function. After searching, all the well-posed models found in the algorithm will be stored for later use.

b. Then for each experiment with different training data, all the previously found well-posed models which contain the inconsistent constraints will be filtered out.

c. Finally the remaining well-posed models will be tested for coverage by JMorven one by one, and the final results are obtained.

5.2 Experiments for CM1

The target model and complete envisionment obtained from JMorven are shown in Appendix A. Table 5 shows the set of experiments under different conditions.

Table 6 shows the performance of the search algorithm discarding the training data. The learning reliability of successful ones is shown in Figure 2. The same learning curve is obtained in CM1-E1,E2,E5 and E7. CM1-E4.b has different curve which will be explained later.

CM1-E1 is the easiest one, which has only one hidden variable qx and fully specified state variables. There exists

Experiment ID	Search Space	Well-posed Models	Running Time (Milli-sec)
CM1-E1	614,061	2,520	13,496
CM1-E2	961,875	5,176	28,271
CM1-E3.a	2,975,625	1,672	64,732
CM1-E3.b	2,975,625	23,136	59,450
CM1-E4.a	1071,225	8,192	34,367
CM1-E4.b	1,071,225	5,848	31,421
CM1-E5	1,975,509	7,128	37,672
CM1-E6	1,358,127	7,336	57,295
CM1-E7	2,736,741	34,272	52,046
CM1-E8	6,281,875	80,616	99,322

Table 6: Performance for learning CM1

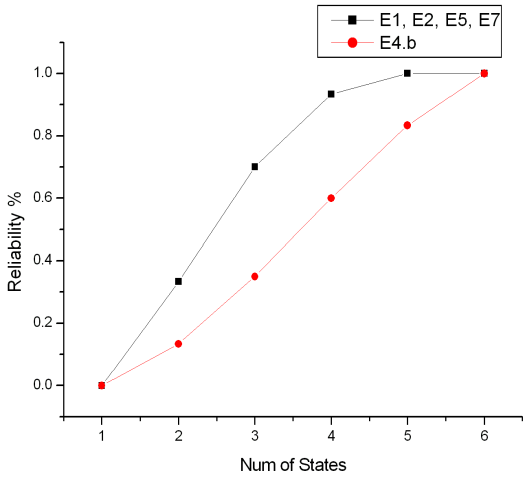

Figure 2: Learning Reliability of CM1 Experiment 1,2,5 and 7.

a kernel set which is defined in (Coghill *et al.* 2004). 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 including these pairs can successfully learn the right model. CM1-E2 removes variable f12 and CM1-E5 removes both f12 and f21. The same learning curve and kernel set as CM1-E1 are obtained. In CM1-E7, no state variables are specified, but only qx is hidden variable, we also can successfully learn the model and get the same learning curve and kernel set as CM1-E1. In addition, in CM1-E7 we found another kind of "correct" model:

Sub (dt0 Hid0) (dt0 c2) (dt0 c1)
Inc (dt0 c1) (f12)
Inc (dt0 c2) (f21)
Inc (dt1 f12) (Hid0)
Dec (dt1 f21) (Hid0)

This model can cover exactly the complete data, but it has different physical meaning: it is a flow-based system, in which it is the change of flow that causes the change of the concentration. On the contrary, the target model is a concentration-based system. The algorithm can not discriminate these two models because of missing state variable information.

In CM1-E3.a, the state variable c1 becomes hidden variable. Given complete data, 24 models which can cover exactly the complete data are found, of which only one is equivalent to target model. 112 over-generalized models which can cover not only the complete data but some other data are also found.

In CM1-E4.a, we make it easier than CM1-E3 by "telling" the system that there is a hidden variable and this hidden variable is a state variable. We found 8 over-generalized models and the target model. Then we go into the details of the non-target models in the result, and find out that all the over-generalized models have the following two constraints (or symmetric constraints in which the positions of Hid0 and Hid1 are swapped):

C5.1 Inc (dt1 Hid1) (dt0 Hid0)
C5.2 Dec (dt0 Hid0) (dt0 Hid1)

Hid0 and Hid1 are two hidden variables. We can add additional hypothesis upon the model to filter out the over-generalized models which contain the above two constraints. One possible hypothesis is that there is no hidden relation in the target model.

Another hypothesis can be no "redundant" hidden variables in the model. Hid0 is a redundant hidden variable because we can induce the following constraint from C5.1 and C5.2:

C5.3 Dec (dt1 Hid1) (dt0 Hid1)

Hid0 becomes logically "redundant", although Hid0 may have physical meaning in a real system. In CM1-E4.b we add this hypothesis and the experiment is successful. But we got different result from other successful experiments. The kernel set becomes smaller and is a subset of CM1-E1's:

(1,3) (2,3)

The learning reliability decreases as shown in Figure 2.

Experiment ID	Hidden Variable	Known State Variable	Success
CM2-E1	q1,q2	c1, c2	Yes
CM2-E2	Q1,q2	None	Yes
CM2-E3	q1,q2,f12	c1,c2	Yes
CM2-E4	q1,q2,f12	None	No
CM2-E5	q1,q2,f12,fo	c1,c2	Yes
CM2-E6	q1, q2,c1	c2,Hidden	Yes
CM2-E7	q1, q2,c1,f12	c2,Hidden	No

* No Model Connection Check

Table 7: Experimental Conditions for CM2

Experiment ID	Search Space	Well-posed Models	Running Time (Milli-sec)
CM2-E1	29,430,625	14,748	416,471
CM2-E2	216,825,625	657,785	1,660,930
CM2-E3.a	52,521,875	12,216	667,982
CM2-E3.b	52,521,875	41,784	517,114
CM2-E4	669,921,875	954,754	3,804,080
CM2-E5	113,358,609	10,080	1,172,365
CM2-E6	3,051,209	47,696	1,009,223
CM2-E7	95,918,823	38,748	1,709,191

Table 8: Performance for learning CM2 (No training data is provided)

Based on CM1-E4.b, in CM1-E6, f12 becomes hidden variable. In this experiment, the envisionment only includes 5 qualitative states. 808 models are found in this experiment. A detailed investigation in these models indicates that no simple hypothesis can be added upon the problem domain to discriminate the target model from the others.

Finally, the last CM1-E8 is based on and harder than CM1-E7 in the sense of hiding both qx and f12 and not specifying any state variables. In this experiment, our algorithm finds 380 models under complete data, of which only 12 models are equivalent to target models.

5.3 Experiments for CM2

The target model of CM2 has been given in the previous section, this model is equivalent to the cascaded tanks model in (Coghill *et al.* 2004) and the set of experiments under different conditions are listed in Table 7. Suppose the inflow u={pos, zer} , there are 14 qualitative states in the complete envisionment, shown in Appendix B.

Like the experiment for CM1, the performance of the search algorithm are listed in Table 8. The learning reliability is illustrated in Figure 3.

The kernel set of the easiest situation CM2-E1 is

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

The result is the same as that obtained in (Coghill *et al.* 2004).

In CM2-E2, all the conditions are the same as CM2-E1 except the state variables are not specified. Similarly as CM1-E7, we found two different kinds of models that can

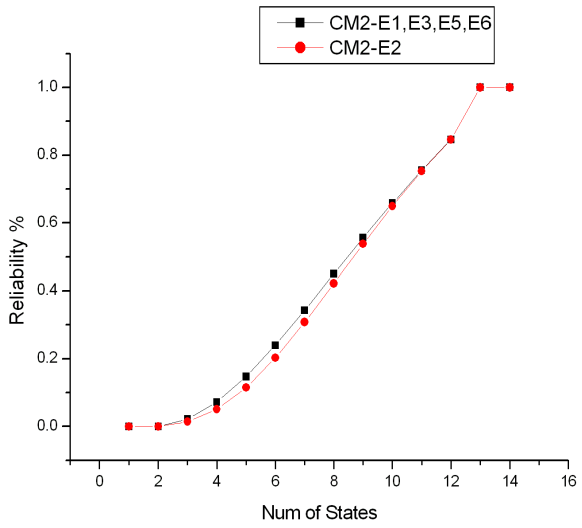

Figure 3: Learning Reliability of CM1 Experiment 1,2,5 and 7.

cover exactly the complete data: one is the target model and the other is shown as follows:

Inc (dt0 c2) (dt0 fo)
Inc (dt0 f12) (dt0 c1)
Sub (dt0 Hid1) (dt0 c1)(dt0 c2)
Sub (dt0 Hid0) (dt0 u) (dt0 f12)
Inc (dt1 c1) (dt0 hid0)
Inc (dt1 fo) (dt0 Hid1)

In this model, c1 is correctly identified as state variable, while fo is wrongly treated as state variable. This model can be seen as a "mixture" of concentration-based and flow-based system. The kernel set found in CM2-E2 is smaller than that in CM2-E1:

(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)

In this kernel set, to become an element of the kernel set, (0,4,5), (0,4,11) and (0,4,13) must be accompanied by an additional state 6. The learning reliability also slightly decreases, as shown in Figure 3.

In CM2-E3, f12 becomes hidden variable, we can still learn the target model in this experiment. Then based on CM2-E3, in CM2-E4 we do not specify state variables, resulting in an unsuccessful experiment. Again based on CM2-E3, in CM2-E5 fo becomes hidden variable, this experiment is similar to CM1-E5, only state variables are known, all the other variables become hidden ones. The target model is successfully learned in this experiment. In CM2-E6, the state variable c1 becomes hidden variable, but one hidden variable is specified as state variable. In this experiment, we succeed in descriminating the target model from other well-posed models. Based on CM2-E6,in CM2-E7, f12 is removed, and we find there exist non-target models in the learning result. In all the successful experiments except CM2-E2, the kernel set and learning reliability are the same.

Some initial conclusion can be drawn from the above experiments:

1. The state variables which have more than one derivative in the 0th differential plane are very important for learning, learning task will become difficult if some of them become hidden variables (CM1-E3, CM1-E4, CM1-E6, CM2-E6, CM2-E7).

2. If a learning task can not be successfully accomplished, we can add more domain specific information to guide the learning (CM1-E4.b). It is still possible to learn the target models. But the kernel set and learning reliability may change.

3. The specification of state variables is also a factor which can influence the learning. Partially or not specifying the state variables will result in a large search space and may lead to failed experiments (CM1-E3,E4,E8 and CM2 E4, E7).

4. Given the right model size, number of hidden variables, and fully specifying the state variables, the non-state variable has the least influence on learning (CM1-E1, E2, E5 and CM2-E1, E3, E5).

5. If state variables are not specified, too many hidden variables can lead to unsuccessful learning (CM1-E7, E8 and CM2-E2, E4).

6. Conclusions and Future Work

The work presented in this paper continues the previous work in (Coghill, Garrett, & King 2004) and (Coghill *et al.* 2004). The contributions of our work are:

First, a more flexible model representation JMorven 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.

Second, an problem-specific backtracking tree search algorithm is proposed, which can find out all the well-posed models efficiently.

Third, the learning of qualitative models under the conditions of different hidden variables and specified state variables are systematically tested and the influence of the hidden and state variables are analyzed.

Last, for speeding up the experiment and avoiding repeated calculation, the same routine work for searching the well-posed models is performed only once in each condition, and the result can be used by all the experiments.

Future work will involve the following aspects: First, the precision of the model can be improved by adding more quantities in the quantity space. The learning task will become more challenging. Second, more complex qualitative models, which have more training data and system variables, can be analyzed. Another direction is using the evolutionary computation to learn the qualitative models with huge size, in the field of which traditional methods do not perform well.

References

Bhaskhar, R., and Nigam, A. 1990. Qualitative physics using dimensional analysis. *Artificial Intelligence* 45:73–111.

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

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

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

Coghill, G. M. 1996. *Mycroft: A Framework for Constraint based Fuzzy Qualitative Reasoning*. Ph.D. Dissertation, Heriot-Watt University.

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

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

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

Morgan, A. 1988. *Qualitative behaviour of Dynamic Physical Systems*. Ph.D. Dissertation, University of Cambridge.

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

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

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

Shen, Q., and Leitch, R. 1993. Fuzzy qualitative simulation. *IEEE Transactions on Systems, Man, and Cybernetics* 23(4):1038–1061.

Wiegand, M. 1991. *Constructive Qualitative Simulation of Continuous Dynamic Systems*. Ph.D. Dissertation, Heriot-Watt university.

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}

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}