Computing Science

QML-Morven: A Novel Framework for Learning Qualitative Models

Wei Pang and George M. Coghill

Technical Report Series **ABDN–CS–12–03**

Department of Computing Science **June 2012**
University of Aberdeen
King's College
Aberdeen AB24 3UE

QML-Morven: A Novel Framework for Learning Qualitative Models

Wei Pang and George M. Coghill

Technical Report ABDN–CS–12–03
Department of Computing Science
University of Aberdeen

June 20, 2012

Abstract: In this report, a novel qualitative model learning (QML) framework named QML-Morven is presented. QML-Morven is an extensible framework and currently includes three QML subsystems, which employ either symbolic or evolutionary approaches as their learning strategies. QML-Morven uses the formalism of Morven, a fuzzy qualitative simulator, to represent and reason about qualitative models, and it also utilises Morven to verify candidate models. Based on this framework, a series of experiments were designed and carried out to: (1) verify the results obtained by the previous QML system ILP-QSI; (2) investigate factors that influence the learning precision and *minimum data requirement* for successful learning; (3) address the scalability issue of QML systems.

Keywords: Qualitative reasoning; Qualitative model learning; Artificial immune systems; Backtracking with forward checking

1 Introduction

Quantitative modelling has been widely used to study complex dynamic systems across various fields. Quantitative models, often in the form of differential equations, provide insights into the systems of interest, and numerical simulation based on such models offers precise descriptions and reliable predictions of the behaviour of dynamic systems.

However, in many real-world systems, it is not always possible to build a reliable quantitative model and thus perform numerical simulation. This is because of the lack of knowledge about the system and imperfect data obtained from experiments. For instance, when modelling some biological systems, on one hand little knowledge about these system is known from literature; on the other hand only sparse and noisy data are available due to the experimental limitations or the nature of the system. In these situations precise numerical parameters for the quantitative model might not be obtained confidently, or the interactions between components of the dynamic systems may not be completely understood. Quantitative models built upon these situations are not meaningful and do not offer much insight into the system.

The limitations of quantitative modelling motivate the development of *qualitative reasoning* (QR) [41], an area of *Artificial Intelligence* devoted to the study of dynamical systems at a qualitative level. The last three decades have seen significant developments in QR, and many QR systems have been proposed to tackle a variety of problems under different conditions. One branch of QR is qualitative simulation, an example of which is QSIM (Qualitative SIMulation) [29, 30]. Qualitative simulation starts from a qualitative model in the

form of qualitative differential equations [30] and can derive qualitative behaviours from this model. Qualitative simulation makes it possible to analyse system behaviours without the use of quantitative models, and it is also a complementary approach to quantitative modelling approaches.

A qualitative simulation system assumes that qualitative models are available or can be manually built from literature. However, in many cases because of the lack of knowledge and data, even a qualitative model is not easy to construct directly from available knowledge. This necessitates the use of qualitative model learning (QML), a branch of QR which involves inferring qualitative models automatically from either qualitative or quantitative data. QML is the inverse of qualitative simulation and has received an increasing amount attention within the qualitative reasoning community during the last two decades.

QML can be deemed a subfield of *system identification* [31], for which the term *Qualitative System Identification* is sometimes used [46]. QML can also be treated as a *Machine Learning* [33] problem, and consequently machine learning algorithms, such as *Inductive Logic Programming* (ILP) [1] can be employed to induce *hypotheses* (qualitative models) from given examples (qualitative data).

QML is a powerful tool for the study of dynamical systems across many disciplines, especially physics and biology [27]. It provides a new means of understanding the system and is particularly useful in situations where little is known about the system, the knowledge is incomplete, or the data available are sparse.

2 Related Work

Over the past two decades, several QML systems have been developed. For more information about these QML systems and their applications, the reader is directed to a comprehensive review by Pang and Coghill [38].

Among these QML systems the following are worth mentioning: GOLEM [35], GENMODEL [15, 22], MISQ [28, 42], QME [52], QSI [46], and the more recent systems ILP-QSI [13] and the incremental learning approach proposed by Srinivasan and King [50]. These systems are briefly described in the rest of this section.

GOLEM is a general ILP system and when applied to learning qualitative models in [4], it tries to construct qualitative models in the form of Horn Clauses [23] that can cover given *positive data*, those behaviours that the system can demonstrate, and exclude hand-generated *negative data*, those behaviours that a system cannot achieve. GOLEM employs a bottom-up search strategy by iteratively generalising a most specific clause. GOLEM is not a complete algorithm because a greedy hill-climbing heuristic search method is used. In addition, the use of negative data when learning qualitative models is not realistic in real-world applications. This is because it is only possible to observe the behaviours that a dynamic system can achieve and impossible to determine what behaviours that the system cannot achieve unless all behaviours of a system are known.

GENMODEL was probably the first special-purpose QML system able to learn from positive data only. It employs a straightforwardly generate-and-test learning strategy: it first generates all possible qualitative constraints and the constraints inconsistent with given data will be removed. Then the remaining constraints are examined by a redundancy check, and those redundant constraints are also removed. Finally the resulting constraints construct a most specific model that covers all positive data. A later version of GENMODEL [22] makes

use of dimensional information [2] to narrow down the generated search space. GENMODEL is a complete algorithm and able to deal with noisy data to some extent because the introduction of fault tolerance when checking the consistency of qualitative constraints against data. The limitations of GENMODEL are (1) it cannot deal with hidden variables (the unobserved variables in the experiments) and (2) it tends to generate overconstrained models.

MISQ employs a similar learning strategy to GENMODEL in its earlier version [28]. The later version of MISQ [42] goes further than GENMODEL because it is capable of processing incomplete knowledge and dealing with hidden variables by the use of the relational-path finding algorithm [43]. MISQ even allows partially specified variables in a qualitative state, which is useful when some of the measurements for some variables are missing. Given incomplete knowledge, MISQ may generate inconsistent constraint set, and it employs heuristics to guide the search over the implicit model space indicated by the generated constraint set. MISQ has the same limitation as GENMODEL: it tends to generate over-constrained models even when complete information is provided.

QME (Qualitative Model Evolution) uses a modified genetic algorithm (GA) as its search strategy. Candidate models are represented as binary trees, and the fitness of each candidate model is evaluated by both positive and negative examples, which is a similar way to GOLEM. The GA iteratively explores and exploits the search space in a beam search manner, until the target models are found or the maximum number of generations is exceeded. The inherent parallelism of GA makes the search more efficient. Up till now QME was the only existing QML system that employed an evolutionary algorithm as its search strategy. The limitations of QME are: (1) like GOLEM, QME uses negative examples; (2) for ease of implementation, qualitative models in QME are simplified by, for example, ignoring corresponding values [29]; (3) The premature convergence of population may happen if the parameters of the GA are chosen inappropriately.

QSI (Qualitative System Identification) is one of the most complicated QML systems and possesses most of the desired features of a QML system. It employs a unique iterative search strategy: it starts with building a very general model (termed a 'shallow' model in QSI) using given incomplete variables, then tries to find a more specialised model based on this 'shallow' model in an iterative manner: the current intermediate model will be iteratively extended by postulating new hidden variables and regenerating a deeper model containing these newly postulated hidden variables, until it passes the so-called model depth test, in which the model is simulated by the qualitative simulation engine QSIM [29, 30]. A major limitation of QSI is that it requires the input data must be complete and consistent, otherwise it may generate wrong models.

ILP-QSI is considered one of the state-of-the-art QML systems. It is an ILP-based QML system having all the characteristics of the earlier QML systems, such as MISQ and QSI. The learning algorithm of ILP-QSI is implemented within the ALEPH system [49], and is a variant of the branch-and-bound algorithm [40]. In the model searching process, a Bayesian posterior probability estimate [32] is used to evaluate candidate models, and the calculation of this Bayesian function does not need negative data. ILP-QSI proposed the concept of well-posed models to help the model searching. A well-posed model must satisfy pre-defined syntactic and semantic constraints, and in the search process only well-posed models will be evaluated, which significantly reduces the computational cost.

Srinivasan and King [50] proposed an incremental learning approach to QML, and the resulting QML system, termed QML-IL in this report, is considered as an extension of ILP-QSI. QML-IL makes use of a simple incremental decomposition strategy to decrease the

computational complexity of ILP-QSI. The learning task is divided into several stages and the decomposition is achieved by domain-specific knowledge. However, in each stage of the learning, the set of variables and the number of constraints must be provided, which is not always feasible. This becomes a major limitation of QML-IL. The automatic decomposition strategy has been preliminarily studied in QML-IL, and further research on the adaptability and scalability of the automatic decomposition algorithms needs to be carried out.

3 Motivations and Organisation of the Report

Although several QML systems have been developed over the last two decades, there still remain a number of unsolved problems in this field: the first issue is the search strategy utilised in order for a QML system to find the target model(s) from the model space. Most existing QML systems employ symbolic approaches, which perform the search in a systematic and deterministic manner. For instance, GENMODEL employed the simplest generate-and-test method. The search could also be performed by employing evolutionary approaches, which are non-deterministic. For example, QME employed a GA to fulfil the search task.

The scalability issue may arise when symbolic approaches are used to search large-sized model spaces, resulting from incomplete knowledge and the complexity of the real-world problems. Symbolic approaches may be very time consuming and even intractable when dealing with complicated models. For instance, when applying ILP-QSI to learn the qualitative model of a real-world biological pathway, the glycolysis pathway, it took several months of compute time on a computer cluster to finally find 35 possible models [13]. One can imagine situations in which QML systems using symbolic approaches are applied to learning more complicated biological pathways and other complex dynamic systems. There has been some effort to address the scalability problems of QML: for instance, the use of an incremental learning approach by QML-IL [50]. However, as mentioned in Section 2 the incremental learning approach highly depends on domain knowledge to divide the learning task into several stages, and the research on automatic decomposition of a learning task is still ongoing.

One promising approach to tackle the scalability issue is to make use of evolutionary algorithms. QML is essentially a search and optimisation problem, that is, to find the best model(s) from the model space to explain the data. Evolutionary algorithms have proven their effectiveness in various large-scale search and optimisation problems, and consequently one can come up with the idea of applying them as search strategies to QML. QME is the first attempt: it fulfils the learning task by employing a GA. But QME was not systematically tested on complicated large-scale problems. Furthermore, as mentioned in Section 2, like GOLEM, QME needs *negative data* to learn the models, which is not realistic in real-world applications. So one of the goals in this report is to develop an evolutionary QML system that is scalable to problems of various complexity and can also learn from only *positive data*. On the other hand, considering the accuracy of symbolic approaches for solving medium-sized problems, we have a more ambitious goal: to develop a QML system which can make use of both the symbolic and evolutionary approaches to meet different requirements.

The second issue is the *minimum data requirement* (termed the *kernel subsets* in ILP-QSI [13]) for successful learning. It is of interest to investigate what kind of data and how many data can lead to inferring the right model, because in real-world problems it is often impossible to obtain complete qualitative data. In ILP-QSI, through power set experiments, it was discovered that there exist *kernel subsets*, each of which is a subset of the complete

qualitative states (described later in Section 4.2). Each of the kernel subsets and any of its supersets can lead to successful learning. A further investigation through solution space analysis [11, 9], an analysis technique which relates qualitative states to the critical points via the isoclines of the system [1], reveals that the states in the kernel subsets tend to cover different branches of behaviours in the *envisionment* (described later in Section 4.2) graph. In this report we want to go further by exploring two key factors that influence the kernel subsets and learning precision: hidden variables (the unobserved variables) and specification of state variables (the variables that cause the changes of other variables).

The final issue is about the model formalism used in a QML system. Most existing QML systems use the QSIM [30] formalism to represent qualitative models. Although QSIM is a well-developed simulation engine, it lacks flexibility: qualitative variables in QSIM are in the form of magnitude-derivative pairs. It is not convenient to represent the behaviour of the higher derivatives of a dynamical system. In addition, we expect that fuzzy sets [54] will be integrated in qualitative model learning so that the developed QML system has the potential to deal with fuzzy data; a QML system using QSIM cannot easily achieve this. This motivates us to use a novel model formalism, Morven (detailed in Section 4), to represent qualitative models.

In this report we present solutions to the above three issues by means of a novel QML framework: QML-Morven. The rest of the report is organised as follows: In Section 4 the Morven formalism and a Java implementation of Morven is briefly introduced. The background and implementation details of QML-Morven are presented in Section 5. This is followed by the report of a series of experiments performed by QML-Morven in Section 6. Finally Section 7 concludes the report and explores possible future work.

4 The Morven Formalism and JMorven

4.1 The Morven Formalism

QML-Morven employs the *Morven* formalism to represent qualitative models. The *Morven* framework [10] is a constraint-based fuzzy qualitative simulator, and its development is largely based on FuSim [47], Predictive Algorithm (PA) [53], and Vector Envisionment(VE) [34].

As in PA, qualitative constraints in a Morven model are distributed over multiple *differential planes*. The *0th* differential plane contains the constraints, which can represent 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.

As in VE, qualitative variables in Morven are in the form of variable length vectors. The first element in the vector is the magnitude of the variable, the *ith* ($i > 1$) element is the *(i-1)th* derivative. The modeller can include as many derivatives as necessary.

As in FuSim, qualitative variables in Morven take their values from fuzzy quantity spaces, which are composed of fuzzy numbers in the form of fuzzy four-tuples [47]. Morven also employs the same fuzzy arithmetic operations as defined in FuSim to calculate the algebraic constraints using the fuzzy quantity spaces. However, in QML-Morven the fuzzy mechanism is not used, and consequently fuzzy numbers degenerate into interval numbers in the form of (a, b), where a, b are real numbers (including $-\infty$ and $+\infty$) and denote the lower bound and

[1] Critical points of a dynamic system are those where at least one derivative of the state variables is zero. The isoclines are contours of critical points.

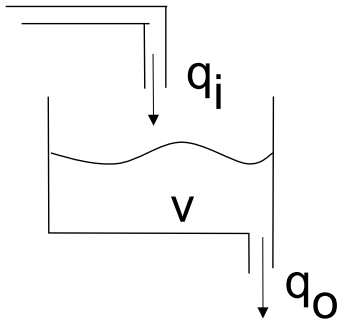

Figure 1: The Single Tank System

Table 1: The Morven Model for the Single Tank System

Differential Plane 0	
C1: Function (dt 0 q_o, dt 0 V)	$(q_o = k * V)$
C2: sub (dt 1 V, dt 0 q_i, dt 0 q_o)	$(V' = q_i - q_o)$
Differential Plane 1	
C3: Function (dt 1 q_o, dt1 V)	$(q_o' = k * V')$
C4: sub (dt 2 V, dt1 q_i, dt1 q_o)	$(V'' = q_i' - q_o')$

upper bound of an interval number, respectively. Accordingly the fuzzy arithmetic operations become interval arithmetic operations.

The single tank system shown in Figure 1 is used as an example to demonstrate how Morven is used to represent qualitative models. The quantitative model for a linear version of this system is as follows:

$$q_o = k * V,$$
$$dV/dt = q_i - q_o,$$

where V is the volume of the liquid in the tank, q_i is the inflow, q_o is the outflow, and k is a positive constant coefficient determined by the cross sectional area of the tank and the density of the liquid.

The corresponding Morven model is shown in Table 1. This model is composed of four constraints, *C1* to *C4*, and the corresponding quantitative relation for each constraint is shown on the right hand side in the brackets. Here the label *dt* means *derivative*, and the integer immediately following it indicates which derivative of the variable (0 means the magnitude). For variable V, the magnitude, the first and second derivatives are used; for variable q_o and q_i, only the magnitude and the first derivative are used.

The primitive *sub* in Constraints *C2* and *C4* stands for the subtraction relation. The primitive *Function* in Constraint *C1* and *C3* has the same definition as that in FuSim. It represents the function relation between two variables and can have arbitrary mappings, which is a generalisation of the M^+ and M^- constraints in QSIM. If all the qualitative variables (including their magnitudes and derivatives) use the signs quantity space, which is shown in Table 2, the mappings of the *Function* in constraint *C1* and *C3* are given in Table 3, in which "1" stands for the existence of a mapping between variables A and B.

Table 2: The Signs Quantity Space

Quantity	Range
negative(-)	$(-\infty, 0)$
zero(0)	0
positive(+)	$(0, \infty)$

Table 3: Function Mappings Under the Signs Quantity Space

Function(A,B)	negative	zero	positive
negative	1	0	0
zero	0	1	0
positive	0	0	1

4.2 JMorven

JMorven [6, 7] is a Java implementation of Morven, and it is tailored and embedded in QML-Morven as a model verification component. Candidate models generated by QML-Morven will be simulated by JMorven and the output will be compared against given data.

The output of JMorven for a qualitative model could be either an *envisionment* containing all possible qualitative states and their legal transitions, or a behaviour tree which is part of the envisionment. A qualitative state is a complete assignment of qualitative values to all qualitative variables of the system. One possible qualitative state of the single tank system described by Morven is shown in Figure 2. In this figure the assignment $V =< pos, zer, zer >$ means that the magnitude of V is *positive*, the first and second derivatives are *zero* (all values are taken from the signs quantity space). It is similar for the assignments of q_i and q_o.

5 QML-Morven

In this section, we introduce the QML-Morven framework. The name given to this framework indicates that Morven is adopted to represent qualitative models. Furthermore, Morven serves as a model verification component within the framework. In the rest of this section, first some background knowledge will be given, then three framework instances of QML-Morven, QML-GENMODEL, QML-BKFC, and QML-CSA, will be described.

V=<pos, zer, zer>
q_i=<pos, zer>
q_o=<pos,zer>

Figure 2: A Qualitative State of the Single Tank by Morven

5.1 Background Knowledge

5.1.1 First-Order Canonical Forms

In this report, the qualitative models to be learnt are specified to be in first-order canonical form. This means that for all constraints in the *0th* differential plane, only the magnitude and first derivative of a variable can appear as arguments. As any higher order system can be represented as a set of first-order systems by introducing more variables, this specification is reasonable. In addition, the first-order form is also required by the causal ordering constraint, which will be described in Section 5.1.3.

5.1.2 Exogenous Variables, System Variables, and State Variables

Exogenous variables in a dynamical system are those variables determined from outside the model, and all the non-exogenous variables are referred to as *system variables*. For instance, in the single tank system shown in Figure 1, variable q_i is an exogenous variable, and variable V and q_o are system variables. *State variables* are those directly affected by the action of integration in a dynamical system [53], and consequently, they are the only variables whose first derivatives can appear in the *0th* differential plane of a model in the first-order canonical form.

5.1.3 Causally Ordered Models

The theory of causal ordering has been publicly debated in the qualitative reasoning community (See [24, 18, 25] for details). In this report, the concept of causally ordered models basically follows the description given by Wiegand [53] and Iwasaki *et al.* [26].

In a causally ordered model, system equations are considered to be directional in the sense that on the left-hand side of an equation there is only one variable and its value is determined by the variables on the right-hand side. So variables on the right-hand side are deemed as the *causes* of the left-hand variable.

Since exogenous variables are not determined by any system variables, they cannot appear on the left-hand side of any system equations. State variables *cause* the changes of other system variables, consequently the magnitudes of state variables cannot appear on the left-hand side of any equation. In addition, according to the definition of state variables, the first derivative of a state variable must appear in one and only one system equation, and its position is on the left-hand side of this equation.

A causally ordered model also requires:

All the system equations are rearranged into an ordered sequence e_1, e_2, ..., e_n, *such that for any equations* e_i *and* e_j, *if* ($i<j$), *the left-hand variable of* e_j *does not appear in the right-hand side of* e_i. [53]

In the case of QML-Morven models, qualitative constraints are deemed to be system equations, and if the qualitative constraints in the *0th* differential plane are arranged to be causally ordered, constraints in higher differential planes are also causally ordered.

The model shown in Table 1 is an example of a causally ordered model. The causal relations of this model are illustrated in Figure 3. In this figure, the arrow links two variables, and the variable at the end point of the arrow "depends" on the one at the start point. The dashed line denotes the integration relation.

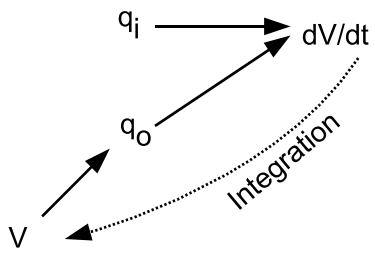

Figure 3: Causal Relations in the Single Tank System

Table 4: M_{dec} Mappings

Function(A,B)	negative	zero	positive
negative	0	0	1
zero	0	1	0
positive	1	0	0

5.1.4 M_{inc} and M_{dec} Constraints in QML-Morven

Because most of the existing QML systems employ the QSIM formalism, in order to compare the learning results of QML-Morven with those obtained by these QML systems, and also for simplifying the learning tasks, the function constraints in QML-Morven have to be specialised. Two functional constraints were introduced to QML-Morven: M_{inc} and M_{dec}. M_{inc} is defined as the function constraint with the mappings described in Table 3, and M_{dec} is defined as the function constraint with the mappings shown in Table 4.

$M^+(A, B)$ in QSIM is equivalent to the conjunction of the following two function relations in QML-Morven:

$$M_{inc}\,(dt\,0\,A,\ dt\,0\,B),$$
$$M_{inc}\,(dt\,1\,A,\ dt\,1B).$$

Note the first M_{inc} constraint specifies the corresponding values and the second one describes that the signs of the derivatives must be the same. These two constraints should appear in the *0th* and *1st* differential plane respectively. Similarly, M^-(A,B) in QSIM is equivalent to the following two constraints:

$$M_{dec}\,(dt\,0\,A,\ dt\,0\,B),$$
$$M_{dec}\,(dt\,1\,A,\ dt\,1\,B).$$

5.1.5 Inconsistent Constraints and Consistency Checking

Inconsistent constraints are any qualitative constraints in the *0th* differential plane that are inconsistent with the qualitative data provided and consequently fail to pass the *consistency checking* (described in the next paragraph). In QML-Morven, as each constraint C_i in the *0th* differential plane may have corresponding constraints in the higher differential planes, which are obtained by successively differentiating C_i, we say C_i is a consistent constraint if and only

if both C_i and its differentiated constraints in the higher differential planes are consistent with the data.

The *consistency checking* examines whether a given *Set of Qualitative States*, *SQS*, is consistent with a given *Qualitative Constraint*, *QC*, in the *0th* differential plane and its differentiated constraints in the higher differential planes. For each qualitative state *QS* in *SQS*, the *consistency checking* works as follows: (1) Extract the qualitative values of all the variables in *QC* and its differentiated constraints in the higher differential planes from *QS*. (2) If *QC* and its differentiated constraints are M_{inc} or M_{dec} constraints, check whether the extracted values are consistent with the value mappings in all these function constraints. If *QC* and its differentiated constraints are arithmetic constraints, check whether the extracted values are consistent with the arithmetic operation in these constraints.

After checking all the states in *SQS*, if the proportion of the states that are consistent with *QC* to all the qualitative states in *SQS* is smaller than a given threshold η ($0 < \eta \leq 1$), *QC* is considered to be inconsistent with *SQS*. The threshold η reflects the noise tolerance: the bigger the value, the less tolerant of noisy qualitative data the consistency checking is.

5.1.6 Conflict Constraints

Two qualitative constraints conflict if they fall into any of the following three categories:

a. Logical Conflict: Two qualitative constraints contradict each other. For example, the following two constraints are logically conflicting:

$$M_{inc}\,(dt\,1\,X,\ dt\,0\,Y),$$
$$M_{dec}\,(dt\,1\,X,\ dt\,0\,Y).$$

b. Redundancy: Two qualitative constraints in the same model describe the same relation. For example, the following two constraints are redundant if they appear in the same model:

$$M_{inc}\,(dt\,0\,X,\ dt\,0\,Y),$$
$$M_{inc}\,(dt\,0\,Y,\ dt\,0\,X).$$

c. Dimensional Inconsistency: The same variable has two different dimensions in two qualitative constraints. For example, if *Hid0* is a hidden variable and the dimension of variables *a* and *b* is different from that of variables *c* and *d*, the following two constraints are dimensionally inconsistent:

Sub (dt 0 Hid0, dt 0 a, dt 0 b),
Sub (dt 0 c, dt 0 Hid0, dt 0 d).

5.1.7 Conflict Set of a Constraint

Given a set of qualitative constraints $\boldsymbol{SC}$, if $C1 \bowtie C2$ is used to represent the fact that *C1* and *C2* are conflicting, the conflict set for a constraint *C1* in $\boldsymbol{SC}$ is defined as:

$$ConflictSet_{\boldsymbol{SC}}(C1) = \{C_i | C_i \in \boldsymbol{SC}, C1 \bowtie C_i\} \quad (1)$$

5.1.8 Multiple Conflict Relations

A conflict may involve more than two constraints. Consider the following three qualitative constraints:

M_{inc} (dt 0 $Hid0$, dt 0 $Hid1$),
sub(dt 0 Hid0, dt 0 a, dt 0 b),
sub(dt 0 a, dt 0 Hid1, dt 0 d).

Here the hidden variables *Hid0* and *Hid1* have the same dimension derived from the second and third constraints. But if there is no gain or amplifier in the system, the first constraint requires *Hid0* and *Hid1* must have different dimensions. In this situation the above three constraints conflict if they appear in the same model.

5.1.9 Defining Constraints

The defining constraint for a variable with a specified derivative (or variable/derivative for short) is the qualitative constraint in which this variable/derivative appears on the left-hand side.

For instance, constraint *sub (dt 1 X, dt 0 Y, dt 0 Z)* is a defining constraint for the first derivative of variable *X*. All derivatives of an exogenous variable and the *0th* derivative (magnitude) of a state variable do not have defining constraints.

5.1.10 Referring Constraints

A referring constraint of a variable/derivative is the constraint in which this variable/derivative appears on the right-hand side. For example, *Sub(dt 0 Y, dt 0 X, dt 0 Z)* is a referring constraint of both the *0th* derivative of variable X and the *0th* derivative of variable Z.

5.1.11 Dependency Set of a Constraint

For a given variable/derivative, all its referring constraints depend on its defining constraints in a causally ordered model. If constraint *C1* depends on *C2*, then this relation is denoted by $C1 \rightarrow C2$.

Given a set of qualitative constraints $\boldsymbol{CS}$, the dependency set for a constraint *C1* is defined as:

$$DependencySet_{CS}(C1) = \{C_i | C_i \in \boldsymbol{CS}, C1 \rightarrow C_i\} \quad (2)$$

For example, the dependency set of constraint *sub (dt 0 X, dt 0 Y, dt 0 Z)* may contain the following two constraints: M_{inc} (dt 0 Y, dt 0 A) and M_{inc} (dt 0 Z, dt 0 B).

In a causally ordered model, a constraint is not allowed to 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.

5.1.12 Well-posed Models

The well-posed model constraints in QML-Morven are basically the same as those defined in ILP-QSI [13]. A well-posed model must satisfy the following constraints: (a) *Size*: The model size, defined as the number of constraints in the *0th* differential plane, must be within the given range. (b) *Completeness*: The model must include all given variables. (c) *Logical consistency*: No conflicting or redundant constraints. (d) *Dimensional Consistency*: Each variable has the same dimension in all constraints in which it appears. (e) *Language*: A model has to satisfy the given language constraints, such as he number of instances of any qualitative relation in the model must be below some pre-specified limit. Detailed discussion

is given by Camacho [8]. (f) *Connection*: All system variables should appear in at least two constraints. (g) *Singularity*: No disjoint sub-models. (h) *Causal Ordering*: The model can be causally ordered. (i) *Coverage*: The model can cover all the given data.

As QML-Morven uses multiple differential planes, only qualitative constraints in the *0th* differential plane need to be checked by the well-posed model constraints.

5.1.13 The Defining Constraint Theorem and Corollary

After the introduction of the concept of well-posed models and the definition of defining constraints, in this section we present the defining constraint theorem and its corollary for well-posed models.

Theorem 1. *In the 0th differential plane, a well-posed model must include one, and only one, defining constraint for each of the system variables with either the 0th or first derivative.*

Proof. Suppose X is a system variable in the model. If X is a state variable, according to the definition of state variables, there must be a defining constraint for the first derivative of X. Otherwise, because the first derivative of X cannot appear on the right-hand side of any constraints, and non-state variables do not have first derivatives in 0th differential plane, there will not be any derivatives in the whole model. This is contradictory to the dynamical system assumption.

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 (either its 0th or first derivative). On the other hand, if a model includes more than one defining constraint for the same variable, it cannot be causally ordered. Consequently Theorem 1 is sound. □

□

Based on *Theorem 1*, together with the definition of *model size*, we have the following corollary:

Corollary 1: *The size of a target model equals to the number of system variables (including hidden variables) in the model.*

The above theorem and corollary will be used later by QML-Morven to reduce the size of the search space.

5.2 QML-GENMODEL

Having described the background utilised by all instantiations of QML-Morven, we now describe the three framework instances of QML-Morven. The simplest framework instance is QML-GENMODEL. QML-GENMODEL employs a similar search strategy to GENMODEL [14, 22] and also requires the same assumptions as GENMODEL to obtain a unique most specific model: It is assumed that (1) the complete qualitative states are available; (2) there are no hidden variables; (3) the dimensional information for all variables is known.

In practice GENMODEL can be used to infer a simple dynamic system when all its variables are identified and the system behaviours are relatively simple and easy to measure

qualitatively. There are three phases in QML-GENMODEL, *data pre-processing*, *model space generation*, and *constraint filtering*. They will be described in detail below.

5.2.1 Data Pre-processing

QML-Morven can directly take as input qualitative data. QML-Morven can also deal with raw quantitative data by reusing the Q2Q (Quantitative to Qualitative data transformation) component in ILP-QSI. In this Q2Q component, the central difference approach [48] is used to estimate the first and second derivatives of a quantitative variable. Then a Blackman filter [3] is used to smooth the estimated first and second derivatives. Finally the smoothed first and second derivatives are converted to qualitative values according to the given quantity space.

5.2.2 Model Space Generation

In this phase, given all possible types of qualitative constraints (including M_{inc} and M_{dec} constraints and arithmetic constraints), all known variables and their derivatives, and the dimensions of all these variables and their derivatives, first all possible qualitative constraints in the *0th* differential plane are generated, which are all the combinations of the constraint types and variables. Then all the generated constraints are checked for dimensional consistency so that all the dimensionally inconsistent constraints are filtered out.

After generating constraints in the *0th* differential plane, constraints in the *ith* differential plane are obtained by differentiating the corresponding constraints in the *(i-1)th* differential plane (i>0). Finally all the generated constraints in all differential planes constitute the *initial model space*, denoted ***IMS***.

5.2.3 Learning Strategy

Similar to GENMODEL, each constraint in the ***IMS*** will be checked for consistency with all the given qualitative states by the *consistency checking* described in Section 5.1.5. All the inconsistent constraints will be filtered out from ***IMS***, resulting in a new constraint set, denoted ***FMS*** (Filtered Model Space). After all the inconsistent constraints have been filtered out, a further *redundancy check* will remove all the redundant constraints in the ***FMS***. Finally a most specific model is obtained.

Based on the description of this section and Section 5.2.2, the pseudo code of QML-GENMODEL is given in Figure 4.

5.2.4 An Example: Learning the Single Tank System

Learning the single tank system described in Figure 1 and Table 1 is used to demonstrate the learning process of QML-GENMODEL. Suppose the following information is given (1) all variables and their dimensions; (2) the inflow was steady and positive, and four states were provided, as shown in Table 5; (3) four constraint types were given: M_{inc}, M_{dec}, *sub*, and *add*. The first three constraint types have been described in previous sections, and the *add* constraint stands for the qualitative addition, for instance, *add (dt 1 A, dt 0 B, dt 0 C)* means $A' = B + C$.

Based on the given information, QML-GENMODEL first generates all 48 possible constraints in the *0th* differential plane, and 20 of them are dimensionally consistent. These 20 constraints and their corresponding constraints in the *1st* differential plane constitute the

```
FUNCTION QML-GENMODEL()
    INPUT:
        1. QD: Qualitative data;
        2. Constraint Repotoire;
            // All possible types of qualitative constraints
        3. All known variables, and their dimensional information;
    BEGIN
        Step 1: Initial Model Space Generation, obtain IMS
              // as described in Section 5.2.2
        Step 2: Constraint Filtering
                  FMS={};
                  FOR each constraint C1 in IMS
                  BEGIN
                       IF C1 is consistent with QD
                           FMS= FMS+ C1;
                        END IF
                  END FOR
        Step 3: Redundancy check on FMS;
        RETURN FMS; // a most specific model
    END
```

Figure 4: The Pseudo Code of QML-GENMODEL

Table 5: States Used for Learning the Single Tank System

State ID	V	q_i	q_o
0	<pos , neg , pos>	<pos , zer>	<pos , neg>
1	<pos , zer , zer>	<pos , zer>	<pos , zer>
2	<zer , pos , neg>	<pos , zer>	<zer , pos>
3	<pos , pos , neg>	<pos , zer>	<pos , pos>

Table 6: One Possible Model Obtained by QML-GENMODEL

0th Differential Plane	*1st Differential Plane*
M_{inc} (dt 0 V, dt 0 q_o)	M_{inc} (dt 1 V, dt 1 q_o)
sub (dt 1 V, dt 0 q_i, dt 0 q_o)	sub (dt 2 V, dt 1 q_i, dt 1 q_o)

IMS. Then the ***IMS*** is filtered by the *constraint filter*, and only 3 constraints (in the *0th* differential plane) are consistent with all four qualitative states: (a) $M_{inc}\,(dt\,0\,V,\ dt\,0\,q_o)$, (b) $M_{inc}\,(dt\,0\,q_o,\ dt\,0\,V)$, and (c) $sub\,(dt\,1\,V,\ dt\,0\,q_i,\ dt\,0\,q_o)$. A redundancy check on these three constraints will remove Constraint (a) or Constraint (b), and one possible model is listed in Table 6. Finally the qualified constraints in different stages are listed in Table 7.

5.2.5 Summary

QML-GENMODEL is a re-implementation of GENMODEL within the QML-Morven framework. The introduction of QML-GENMODEL is illustrative because many infrastructure modules it utilises are described in detail, and these infrastructure modules are also utilised by the other two framework instances, namely QML-BKFC and QML-CSA.

QML-GENMODEL can generate a unique most specific qualitative model, provided that all the variables are identified and complete qualitative data are available. However, in most cases, the existence of hidden variables and incomplete data is unavoidable. QML-GENMODEL may produce an over-constrained model if incomplete data are provided. The existence of hidden variables will increase the size of the model space and result in many possible models. In particular, when both hidden variables and incomplete data exist, the learning task will be more difficult. In the next two sections, two framework instances aiming to deal with incomplete data and hidden variables, QML-BKFC and QML-CSA, are introduced.

5.3 QML-BKFC

QML-BKFC allows the existence of hidden variables and incomplete data. On the other hand, QML-BKFC makes use of the well-posed model constraints as described in Section 5.1.12 to narrow the range of candidate models.

QML-BKFC performs the same data preprocessing as QML-GENMODEL and has a similar model space generation process. For the learning strategy, rather than the simple redundancy check utilised in QML-GENMODEL, QML-BKFC first calculates the conflicting and dependent relations of the constraints in the model space and partitions the model space accordingly. Then it employs an efficient backtracking algorithm with forward checking (BKFC) to search in this partitioned model space and extract all possible models. The details of QML-BKFC are as follows:

5.3.1 Model Space Generation

The model space generation is similar to that in QML-GENMODEL, but there are three differences:

(a) In QML-BKFC, users can specify the maximum number of hidden variables. For instance, if this number is 3, QML-BKFC will generate three possible hidden variables, denoted

Table 7: Constraints in Different Stages of QML-GENMODEL

Constraint ID	Constraints	Dimensional Consistency	Dimensional & Data Consistency
1	M_{inc} (dt 0 V, dt 0 q_o)	√	√
2	M_{inc} (dt 1 V, dt 0 q_o)	√	
3	M_{dec} (dt 0 V, dt 0 q_o)	√	
4	M_{dec} (dt 1 V, dt 0 q_o)	√	
5	M_{inc} (dt 0 V, dt 0 q_i)	√	
6	M_{inc} (dt 1 V, dt 0 q_i)	√	
7	M_{dec} (dt 0 V, dt 0 q_i)	√	
8	M_{dec} (dt 1 V, dt 0 q_i)	√	
9	M_{inc} (dt 0 q_o, dt 0 V)	√	√
10	M_{inc} (dt 1 q_o, dt 0 V)		
11	M_{dec} (dt 0 q_o, dt 0 V)	√	
12	M_{dec} (dt 1 q_o, dt 0 V)		
13	M_{inc} (dt 0 q_o, dt 0 q_i)	√	
14	M_{inc} (dt 1 q_o, dt 0 q_i)		
15	M_{dec} (dt 0 q_o, dt 0 q_i)	√	
16	M_{dec} (dt 1 q_o, dt 0 q_i)		
17	M_{inc} (dt 0 q_i, dt 0 V)	√	
18	M_{inc} (dt 1 q_i, dt 0 V)		
19	M_{dec} (dt 0 q_i, dt 0 V)	√	
20	M_{dec} (dt 1 q_i, dt 0 V)		
21	M_{inc} (dt 0 q_i, dt 0 q_o)	√	
22	M_{inc} (dt 1 q_i, dt 0 q_o)		
23	M_{dec} (dt 0 q_i, dt 0 q_o)	√	
24	M_{dec} (dt 1 q_i, dt 0 q_o)		
25	sub (dt 0 V, dt 0 q_o, dt 0 q_i)		
26	sub (dt 1 V, dt 0 q_o, dt 0 q_i)	√	
27	add(dt 0 V, dt 0 q_o, dt 0 q_i)		
28	add(dt 1 V, dt 0 q_o, dt 0 q_i)	√	
29	sub (dt 0 V, dt 0 q_i, dt 0 q_o)		
30	sub (dt 1 V, dt 0 q_i, dt 0 q_o)	√	√
31	add (dt 0 V, dt 0 q_i, dt 0 q_o)		
32	add (dt 1 V, dt 0 q_i, dt 0 q_o)	√	
33	sub (dt 0 q_o, dt 0 V, dt 0 q_i)		
34	sub (dt 1 q_o, dt 0 V, dt 0 q_i)		
35	add (dt 0 q_o, dt 0 V, dt 0 q_i)		
36	add (dt 1 q_o, dt 0 V, dt 0 q_i)		
37	sub (dt 0 q_o, dt 0 q_i, dt 0 V)		
38	sub (dt 1 q_o, dt 0 q_i, dt 0 V)		
39	add (dt 0 q_o, dt 0 q_i, dt 0 V)		
40	add (dt 1 q_o, dt 0 q_i, dt 0 V)		
41	sub (dt 0 q_i, dt 0 V, dt 0 q_o)		
42	sub (dt 1 q_i, dt 0 V, dt 0 q_o)		
43	add (dt 0 q_i, dt 0 V, dt 0 q_o)		
44	add (dt 1 q_i, dt 0 V, dt 0 q_o)		
45	sub (dt 0 q_i, dt 0 q_o, dt 0 V)		
46	sub (dt 1 q_i, dt 0 q_o, dt 0 V)		
47	add (dt 0 q_i, dt 0 q_o, dt 0 V)		
48	add (dt 1 q_i, dt 0 q_o, dt 0 V)		

"Hid0", "Hid1" and "Hid2", where the prefix "Hid" indicates they are hidden variables. (Note the resulting models do not necessarily include all the hidden variables, because it is possible that a model containing a subset of given hidden variables is good enough to explain the data.) When generating the model space, these hidden variables will be taken into account.

(b) As QML-BKFC aims to find models that can be causally ordered (one of the well-posed model constraints), the generation of the following two kinds of qualitative constraints should be prevented: (b.1) constraints in which the magnitude or first derivative of an exogenous variable appears on the left-hand side; (b.2) constraints in which the magnitude of a state variable appears on the left-hand side.

(c) In the succeeding dimensional consistency checking, hidden variables in the constraints will not be considered as their dimensions are unknown.

5.3.2 Constraint Filtering

As in QML-GENMODEL, the generated ***IMS*** will be further filtered by the *consistency checking*, and the ***FMS*** is obtained. The size of the ***FMS*** may be very large and often contain conflicting constraints. The reasons are twofold: (1) the constraint filter cannot filter out any constraints containing hidden variables. (2) Fewer constraints are filtered because of incomplete data. So it is not always possible to extract a unique model from ***FMS*** by simply removing the redundant constraints, as in QML-GENMODEL.

5.3.3 Calculation of the Conflict Set and Dependency Set

First we define ***FMS0*** as a set that includes all constraints in the *0th* differential plane in ***FMS***. In this phase, for each constraint C_i in ***FMS0***, its conflict set (defined by Formula (1) in Section 5.1.7) is as follows:

$$ConflictSet_{\boldsymbol{FMS0}}(C_i),$$

and the dependency set (defined by Formula (2) in Section 5.1.11) is as follows:

$$DependencySet_{\boldsymbol{FMS0}}(C_i).$$

These two sets will be calculated and the results are recorded for later use.

5.3.4 Constraint Set Partition

FMS0 is divided into several subsets, each of which contains all the defining constraints (defined in Section 5.1.9) of either the magnitude or first derivative of the same system variable. ***DS*** is a set that takes each of these subsets as an element, denoted as ***DS***= $\{S_i\}$ (i=1 to N), and N is the number of system variables (including hidden variables). For any two elements in ***DS***, $|S_i| \leq |S_j|$ if $i < j$. For example, in the single tank system, when q_o is a hidden variable, denoted *Hid0*, the subset for variable *Hid0* will include the following three constraints:

M_{inc}(dt 0 Hid0, dt 0 V),
M_{dec}(dt 1 Hid0, dt 0 V),
M_{dec}(dt 1 Hid0, dt 0 q_i).

In the above constraints, the first is a defining constraint of the magnitude of *Hid0*, and the other two are defining constraints of the first derivative of *Hid0*.

5.3.5 Backtracking with Forward Checking

The basic idea of QML-BKFC is: for each subset S_i in the partitioned ***DS***, select only one constraint, thus construct a model, then check the validity of this model. The correctness of this selection is guaranteed by *Theorem 1* described in Section 5.1.13. To achieve this, a backtracking algorithm with forward checking (details of which can be found in [45]) is employed to perform a systematic search in ***DS*** in a depth-first manner.

5.3.6 Search Modes

To deal with hidden variables, users are provided with two search modes: *full* and *half* search modes. In the *half* search mode, QML-BKFC will find the models in which the number of hidden variables is equal to the given maximum number of hidden variables. While in the *full* search mode, given the maximum number of hidden variables n, the resulting model will include *at most* n hidden variables. The *full* search mode can be implemented easily by modifying ***DS*** as follows: for each S_i in ***DS*** which contains the defining constraints for a hidden variable, an "empty" constraint ϕ is appended to it: $S_i = S_i \bigcup \{\phi\}$. The current partial model will not change if an "empty" constraint is added to it.

The pseudo code for QML-BKFC is given in Figure 5. In this algorithm, for a constraint C_j in the partial model PM, *ConflictsCount[j]* is used to record the number of constraints in PM that conflict with C_j. The function *CheckPartialModel* (C_j, *PM)* checks the consistency when constraint C_j is added into PM. This function only checks the multiple conflicting relations, because the forward checking has already ensured that there are no binary conflict relations in *PM*. In the next paragraph, the implementation of *CheckCompleteModel (PM)* will be presented.

5.3.7 Well-posed Model Checking

CheckCompleteModel (PM) will check all the well-posed model constraints which are not considered in *CheckPartialModel* (C_j,*PM)*. In order to identify a non-well-posed model with the minimum computational effort, the more easily a constraint can be checked the earlier it should be checked. For instance, the *model size* and *model completeness* should be checked first, the *causal ordering* and *coverage test* should be checked last. The checking for three of the well-posed constraints is detailed as follows (the other constraints are the same as those in ILP-QSI):

(1) Model Size: in ILP-QSI, the model size has to be pre-specified. *Corollary 1* tells us that this is equal to the *half* search mode in QML-BKFC, in which the number of hidden variables is determined, and hence the model size. If the *full* search mode is selected, only the range of the model size is provided. In this situation, the model size constraint in QML-Morven is more relaxed than that in ILP-QSI.

(2) Causal Ordering: A causal ordering algorithm based on [53, 26] has been implemented to automatically examine whether a candidate model can be causally ordered. Note that for efficiency reasons this algorithm does not try to rearrange the constraints.

(3) Coverage Test: The coverage test is fulfilled by a tailored JMorven. In this tailored JMorven, the module which can generate the *envisionment* [6] is modified to make it capable of being intensively and continually called. When a candidate model is submitted to this tailored JMorven, all possible qualitative states for this model will be generated. It is not

```
FUNCTION QML-BKFC()
 INPUT:
     FMS0={C1,C2,. . ., Cn}     // n is the number of constraints
     DS={S1, S2,. . . SN}     // DS is the partitioned search space
 BEGIN
     PM={}; // The partial model, initially set to be empty
      Integer[] ConflictsCount= new Integer[n];
     ConflictsCount[1]=ConflictsCount[2].....  =ConflictsCount[n]=0 ;
      BackTrackingFC(1, ConflictsCount);
 END FUNCTION     // function QML-BKFC
 FUNCTION BacktrackingFC(Integer i,  Integer[] ConflictsCount)
  BEGIN
         IF (i==N+1)  exit;   // the exit of the recursion
         FOR EACH Constraint Cj in Si  DO
         BEGIN
              IF ( ConflictsCount[j]>0)     // Cj conflicts with PM
                   Continue;
              END IF
              DirtyMark={};  // record the conflict set of Cj
              IF (CheckPartialModel (Cj, PartialModel))
              BEGIN
                   FOR EACH Constraint Cx in ConflictSetFMS0(Cj)
                   BEGIN       //Set the conflict counts
                     ConflictsCount[x]++;    DirtyMark+= {x};
                   END FOR
                   PM= PM+Cj;
                   IF(i <N+1)
                       BacktrackingFC(i+1,LeagalValue);
                   END IF
                   IF (i==N)
                     CheckCompleteModel (PartialModel);
                   END IF
                    FOR EACH integer x in DirtyMark
                    BEGIN   //restore the conflict counts
                      ConflictsCount[x]--;
                    END  FOR
                      PM= PM-Cj;
              END IF     // IF CheckPartialModel ()
         END FOR      // FOR EACH Constraint Cj in Si
  END FUNCTION
```

Figure 5: The Pseudo Code of QML-BKFC

necessary to calculate the legal transitions among the generated states, because only the states themselves are used for learning.

If the given maximum number of hidden variables are very big, there may exist some large-sized candidate models. The qualitative simulation performed by JMorven might be computationally expensive for these large-sized models compared with other well-posed model checking modules, so the coverage test is arranged as the last module performed in the well-posed model checking. Only the qualitative model that satisfies all the other well-posed model constraints can be tested for coverage. If a model satisfies all well-posed model constraints, it will be recorded. More than one well-posed model may be found in an experiment.

5.3.8 Summary

QML-BKFC employs a symbolic and deterministic approach to systematically search in the model space, and it is a *complete* algorithm in the sense that it can find a well-posed model if one exists (or report correctly if there is no well-posed model). The efficiency and completeness of QML-BKFC, together with the more relaxed assumption it allows (hidden variables and incomplete data), make it capable of performing systematic experiments under different conditions, that is, different hidden variables and different incomplete data given. These experiments will be described in detail in Section 6.

5.4 QML-CSA

As mentioned in Section 3, QML is essentially an optimisation problem, that is, to find the best qualitative model that can explain the data. QML-BKFC can achieve this when dealing with small and medium-sized model spaces. However, scalability issues will arise when QML-BKFC is applied to solve large-sized complicated problems. This makes it necessary to employ more effective algorithms to learn complicated models.

QML-CSA is proposed to address the scalability problem. It utilises the *clonal selection algorithm* (CSA) [17] to search in the model space. In addition, under the conditions of incomplete data and hidden variables, the model search space may be highly multimodal because there may exist multiple models that satisfy all the well-posed model constraints, and the optimisation version of CSA [17] is particularly suitable for searching in this kind of search space.

Except for the core search algorithm, the processes in QML-CSA, including the *Model Space Generation*, *Constraint Filtering*, *Calculation of Conflict Set and Dependent Set* and *Constraint Set Partition* are the same as those in QML-BKFC. In the rest of this section, we will describe the modified CSA search strategy for searching the qualitative model space. It is pointed out that an earlier version of QML-CSA and some preliminary experimental results were presented by Pang and Coghill [36].

In order to make CSA suitable for learning qualitative models, three key issues must be considered: the antibody encoding strategy, hyper-mutation and affinity evaluation. These issues are described below:

5.4.1 Antibody Encoding Strategy

When applying CSA to qualitative model learning, the target models to be found are treated as the antigens, and the antibodies stand for candidate models. Considering the structure

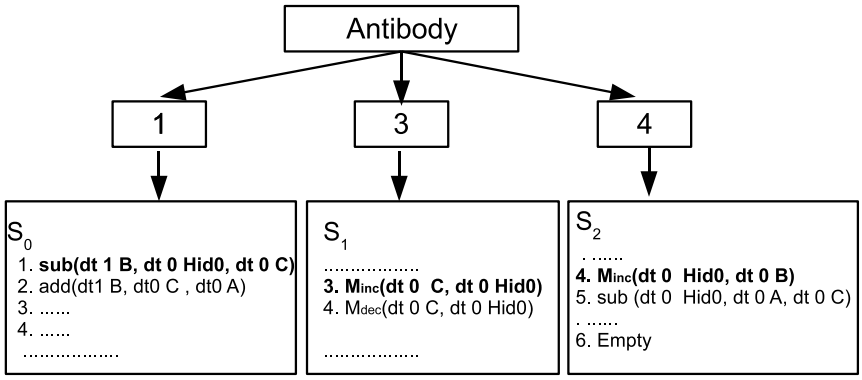

Figure 6: Antibody Encoding

of the partitioned search space ***DS*** described in Section 5.3.4, to better make use of the information indicated in ***DS***, an integer encoding strategy is adopted.

The antibody is composed of several slots, each of which corresponds to a different subset S_i in ***DS***. Each slot is assigned to an integer, which indicates a qualitative constraint selected from the corresponding subset. The correctness of this encoding strategy is guaranteed by *Theorem 1* and *Corollary 1*. In addition, to avoid the antibody getting trapped in a local optimum for an intolerable time, a survival time counted by generations is associated with each antibody. If an antibody exceeds its survival time, it will be replaced by a randomly generated antibody.

Figure 6 shows an example of the antibody encoding for a dynamical system. Suppose there are three known variables in this system: A, B and C. A is the exogenous variable, B is a state variable. There is also one hidden variable Hid0. In this figure, the constraints in bold will be selected to compose a candidate model.

5.4.2 Hyper-Mutation

The hyper-mutation is also modified due to the modification of the antibody encoding. For each slot in 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, and 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 belonging to a neighbourhood of the original antibody.

5.4.3 Affinity Evaluation

A scoring system based on the well-posed model constraints was set up to evaluate the affinity of an antibody. The more and better a model satisfies the well-posed model constraints, the higher score it will get.

For example, for the *Model Conflicts* criterion, if there is no conflicting constraint in the model, a full weighted score will be added to the affinity value. Otherwise, the score is calculated as follows:

$$ConflictScore = W1 * (1 - \frac{N_C}{N_M}). \quad (3)$$

In the above formula, *W1* is the weight, and the weights for all the criteria are currently the same. N_C and N_M are the number of conflicting constraints and the total number of constraints in the model, respectively. The scores for other constraints can be calculated in a similar way. Finally the affinity score equals to the sum of all these calculated scores.

Because of the relatively expensive computational cost of the causal ordering check and coverage test, the coverage test will be performed only if all other constraints are satisfied, and the causal ordering check will be performed only if all other constraints except the coverage constraint are satisfied.

5.4.4 Algorithm Description

The steps of QML-CSA for learning qualitative models are basically the same as for the original optimisation version of CSA. Note the termination condition of the algorithm differs for different experiments, which will be described in Section 6.2 and Section 6.4 respectively.

Step 1: Initialization **Parameter Setting**: Set the values of the following parameters: 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 generate *PopSize* antibodies to construct the initial antibody repertoire (population). Initialise the *Memory Pool*, which records the newly found well-posed models.

Step 2: Evolutionary Iteration

While (*termination conditions are 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 is 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. Note one copy is kept unchanged for each of the original antibodies.

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

Step2-5 Update Memory Pool: Record the newly found well-posed models.

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

5.4.5 Summary

QML-CSA is the only QML system that utilises an evolutionary algorithm to learn qualitative models from positive only data. (QME [52] cannot fulfil the learning tasks without *negative data*.) As CSA is not a *complete* algorithm, we need to investigate its reliability under different circumstances. The experiments performed to evaluate the reliability and scalability of QML-CSA will be detailed in Section 6.2 and Section 6.4.

6 Experiments to Evaluate QML-Morven

In this section, QML-Morven will be assessed by analysing the results obtained from performing a series of experiments. First, the experimental testbed is described in Section 6.1. Second, in Section 6.2 QML-Morven is compared directly with ILP-QSI by means of the same experiments. Third, the scalability of QML-CSA is explored in Section 6.4.

6.1 Experimental Testbed: Compartmental Models

Compartmental models are chosen as the experimental testbed for two reasons: First, compartmental models are abstractions of many dynamical systems, and their applications have been found in many disciplines, including biology [19], pharmacokinetics [44], epidemiology [5], physiology [20], and ecology [21]. Second, because compartmental modelling is a more general methodology, many *de facto* benchmarks in the qualitative reasoning community such as the single-tank, U-tube, cascaded tanks and couple tanks systems [13], have analogous compartmental models. By carrying out the experiments on these models, first we aim to reproduce the same or consistent experimental results as obtained from previous QML systems, especially ILP-QSI. More importantly, analysing the results of the experiments performed on these general models will also benefit QML research in general.

A compartmental model is composed of several units and the flows of material (or the transmission of energy) between these units. The features of the compartmental models are listed as follows:

- Each unit is called a *compartment*, and the material in all compartments is homogeneous (well-mixed).
- The volumes of all compartments are assumed to be constant.
- The material flowing into a compartment is assumed to be instantaneously mixed with the material of the compartment.
- Compartmental models can also have input flows from outside and outflows to the environment.

Figure 7 shows all the compartmental models to be investigated in this section. In this figure, *c1*, *c2*, *c3*, and *c4* stand for the concentrations in the compartments, and they are also used to "label" the compartments; *f12*, *f21*, *f23*, and *f34* denote the flows from one compartment to another ("*fij*" stands for the flow from compartment i to compartment j); u is the input flow; *f20*, *f30*, and *f40* are the output flows ("*fi0*" stands for the outflow of compartment i) to the environment. Each flow has a monotonically increasing relation with the concentration of its source compartment (namely the compartment it starts from). For instance, *f12* will increase with the increase of *c1*. One exception is the input flow u, which is an exogenous variable and determined from outside.

CM1 is a closed coupled system with bidirectional flows between the two compartments. The U-Tube system presented in ILP-QSI [13] is a special case of model CM1. The U-Tube system consists of a hollow tube in the shape of a U and liquid in the tube, as shown in Figure 8. In this figure, the left and right tank of the U-Tube are labeled A and B, and the levels of the liquid in these two tanks are denoted as $level_A$ and $level_B$, respectively. Liquid can flow freely from one tank to another, and the flow from Tank A to Tank B is denoted as

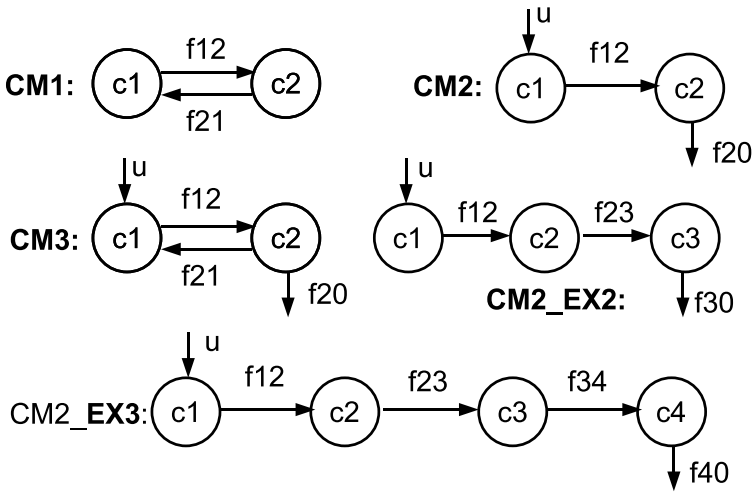

Figure 7: Compartmental Models

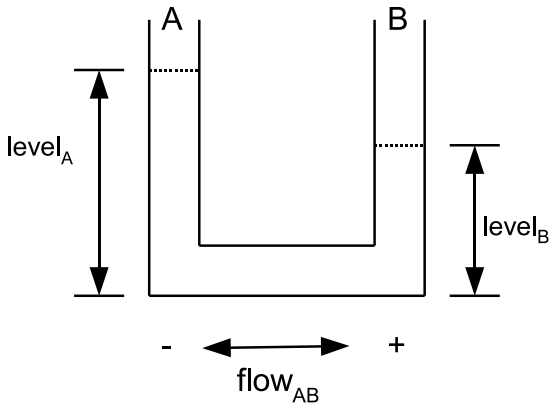

Figure 8: The U-Tube System

$flow_{AB}$. The following two reasons make the U-Tube a special case of model CM1: first, in the U-Tube if the levels of water ($level_A$ and $level_B$) correspond to the concentrations in CM1 ($c1$ and $c2$), and the flows are ignored, the possible qualitative states and the legal transitions among these states in the U-Tube are the same as those in CM1; second, The U-Tube system can be seen as a special CM1 model in which f12 is always equal to -f21.

In CM2 there is only one uni-directional flow from the left compartment to the right, but with an input flow u and output flow *f20*. Similarly, the cascaded tanks system used in ILP-QSI is an example of model CM2.

Based on model CM1, model CM3 has additional input and output. Model CM3 is often used to represent metabolic systems, and its simplified version (the input flow u is assumed to be zero and steady) has been qualitatively studied by Coghill *et al.* [12].

Models CM2_Ex2 and CM2_Ex3 are the extensions of model CM2 with three and four compartments respectively. These two models are catenary compartmental models, in which all compartments are arranged as a chain, and each compartment has connections to its two adjacent neighbours only.

Table 8: The JMorven Model for CM1

Differential Plane 0
C1: M_{inc} (dt 0 f12, dt 0 c1)
C2: M_{inc} (dt 0 f21, dt 0 c2)
C3: sub (dt 0 f_x, dt 0 f12, dt 0 f21)
C4: M_{dec} (dt 1 c1, dt 0 f_x)
C5: M_{inc} (dt 1 c2, dt 0 f_x)
Differential Plane 1
C6: M_{inc} (dt 1 f12, dt 1 c1)
C7: M_{inc} (dt 1 f21, dt 1 c2)
C8: sub (dt 1 f_x, dt 1 f12, dt 1 f21)
C9: M_{dec} (dt 2 c1, dt 1 f_x)
C10: M_{inc} (dt 2 c2, dt 1 f_x)

Finally, as an example, we give the JMorven model of CM1 with two differential planes in Table 8, where f_x stands for the $netflow$ from compartment $c1$ to $c2$.

6.2 Performing Similar Kernel Subset Experiments to ILP-QSI

Model CM1, the U-Tube, and model CM2 are selected to perform the similar kernel subset experiments as described by Coghill *et al.* [13]. As ILP-QSI used the QSIM representation, the data provided to ILP-QSI for the kernel subset experiments are in the form of magnitude-derivative pairs, for instance, the data shown in Table 10. In QML-Morven, this kind of data is called the data of "one and a half" differential planes because the second derivatives of state variables are not given (or explicitly given). As QML-Morven intends to repeat experiments in ILP-QSI and compare the results, the training data used in this section are also in the format of "one and a half" differential planes. Consequently in the model coverage test constraints containing the second derivatives will not be considered. For instance, after constraints C9 and C10 in model CM1 shown in Table 8 are removed, the new model will generate exactly the same data as the given ones, as listed in the first five columns of Table 9.

6.2.1 Experimental Method

(1) We use JMorven to simulate models CM1 and CM2 and obtain the envisionment.

(2) All non-empty subsets of the envisionment are presented to QML-Morven, and each of these subsets is treated as the training data for an individual experiment. This results in 2^N-1 different experiments, where N is the number of states in the envisionment.

(3) QML-BKFC is employed to perform all experiments, it works in *full* search mode, and the maximum number of hidden variables is 1 for CM1 and 2 for CM2.

(4) For all the experiments in which the target model is successfully identified, the corresponding sets of training data will be recorded as *kernel subsets*.

(5) Plot the curve of learning precision versus the number of states used. The learning precision is defined as the proportion of the models in the result that are equivalent to the correct model.

(6) Repeat Step 3 ∼5, but using QML-CSA instead of QML-BKFC. As the search spaces for learning CM1 and CM2 are relatively small (given empty data, the size of the search space is $\sim 10^5$ for CM1 and $\sim 10^8$ for CM2), the parameters of QML-CSA are set as follows: the

Table 9: The Data Used for Learning the U-Tube and CM1

State ID	c1 ($level_A$)	c2 ($level_B$)	f12	f21	$f_x(flow_{AB})$
0	<zer , zer>	<zer , zer>	<zer , zer>	<zer , zer>	<zer , zer>
1	<zer , pos>	<pos , neg>	<zer , pos> *	<pos , neg> *	<neg , pos> †
2	<pos , neg>	<zer , pos>	<pos , neg>	<zer , pos> *	<pos , neg>
3	<pos , zer>	<pos , zer>	<pos , zer> *	<pos , zer> *	<zer , zer> †
4	<pos , pos>	<pos , neg>	<pos , pos> *	<pos , neg> *	<neg , pos> †
5	<pos , neg>	<pos , pos>	<pos , neg>	<pos , pos> *	<pos , neg>

* Data used to learn CM1 but not U-Tube
† Data used to learn U-Tube but not CM1

population size is 10 for CM1 and 100 for CM2. For both models CM1 and CM2, the clonal size is 10, the surviving time of all antibodies is 100 generations, the maximum running time is 60 seconds, and the hyper-mutation probability is 0.5. The termination conditions of the algorithm are either the experiment exceeds the maximum running time or a false positive model is found.

6.2.2 Experiments on Model CM1 and the U-Tube

For model CM1, suppose f_x cannot be measured. The qualitative states obtained from the envisionment are listed in the first five columns of Table 9. For the U-Tube, suppose $level_A$, $level_B$, and $flow_{AB}$ can be measured, as shown in Table 9.

For learning these two models, both QML-BKFC and QML-CSA obtained the same learning precision and kernel subsets. The learning precision is shown in Figure 9, in which "CM1-E1" indicates the learning precision of CM1.

For learning the U-Tube, the learning precision and kernel subsets obtained are exactly the same as those obtained by ILP-QSI. The kernel subsets of learning the U-Tube are eight pairs: (1,2) (1,3) (1,5) (2,3) (2,4) (3,4) (3,5) (4,5). The number in the pairs stands for the State ID in Table 9. For learning CM1, the kernel subsets include only the first five pairs of the above eight. This means for learning CM1, QML-Morven can successfully learn the correct model from any of these five pairs and all their supersets.

Compared to the results of learning the U-Tube, one can see that the learning precision of model CM1 is lower, and the kernel subsets obtained are fewer. This is because different data are used for learning the U-Tube and CM1, as shown in Table 9, which results in the observed behaviours of the U-Tube being more *symmetric* than those of CM1:

- For the U-Tube system, (a) when the level of fluid in one side is increasing, the level in the other side must be decreasing, or both are steady; (b) when $flow_{AB}$ is positive and decreasing, the $flow_{BA}$ (or -$flow_{AB}$) must be negative and increasing, or both of them are zero and steady.

- While in the case of CM1, although the first derivatives of $c1$ and $c2$ are always opposite or both steady, and so are the first derivatives of $f12$ and $f21$, the magnitudes of the two flows are not always opposite any more. (This means the magnitude of f_x sometimes cannot be determined.)

The more symmetric behaviours demonstrated by the U-Tube make it more likely to be identified given incomplete data.

Table 10: The Envisionment States Used for CM2 Experiments

State ID	c1	c2	f12	f20
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>

6.2.3 Experiments on Model CM2

For model CM2, suppose the inflow u=<*pos*, *zer*>, which is the same assignment as used in previous experiments on the cascaded tanks system performed by ILP-QSI, the qualitative states obtained from the complete envisionment are listed in Table 10.

The kernel subsets and learning curve are exactly the same as those obtained by ILP-QSI: The kernel subsets obtained are eight triples: (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 learning precision is shown by the curve "CM2-E1" in Figure 10. As the learning results have been analysed by Coghill *et al.*[13], we will not describe the details of the analysis.

The experiments presented in this subsection demonstrated that QML-Morven could perform similar kernel subset and learning precision analysis to ILP-QSI. Furthermore, results obtained are consistent with those from ILP-QSI.

6.3 Extension Experiments on Models CM1 and CM2

In the previous section we have performed similar experiments as in ILP-QSI. In this section, we present a set of experiments for CM1 and CM2 designed to investigate the influence of different hidden variables and different specifications of state variables. These experiments are categorized as three conditions: omitting non-state variables, partially or not specifying the state variables, and omitting state variables.

Table 11 shows the sets of experiments for CM1 and CM2. In this table, f_{x1} and f_{x2} stand for the *netflow* of compartment *c1* and *c2*, respectively. Experiments CM1-E1 and CM2-E1 have been reported in Section 6.2, and they are listed here for comparison. CM1-E6 is taken as an example to illustrate the experimental conditions: in this experiment, f_x, *f12*, and *c1* are hidden variables, *c2* is known as a state variable and we also hypothesise that another hidden variable is a state variable. Some experiments are explained as follows: The difference between CM1-E4.a and CM-E4.b is that in CM1-E4.b an additional hypothesis is added to the background knowledge: we assume that there is no hidden relation (a constraint in which all variables are hidden ones).

For each of the combinations, first complete states from the envisionment are presented to QML-BKFC (the *full* search mode, but the maximum number of hidden variables is equal

Table 11: Experimental Conditions for CM1 and CM2[1]

Experiment ID	Hidden Variables	Known State Variables	Success
CM1-E1	f_x	c1, c2	Yes
CM1-E2	f_x, f12	c1,c2	Yes
CM1-E3	f_x,c1	c2	No
CM1-E4.a	f_x,c1	c2,Hidden	No
CM1-E4.b	f_x,c1	c2,Hidden	Yes [2]
CM1-E5	f_x, f12,f21	c1,c2	Yes
CM1-E6	f_x,f12, c1	c2,Hidden	No
CM1-E7	f_x	None	Yes
CM1-E8	f_x,f12	None	No
CM2-E1	f_{x1},f_{x2}	c1, c2	Yes
CM2-E2	f_{x1},f_{x2}	None	Yes
CM2-E3	f_{x1},f_{x2},f12	c1,c2	Yes
CM2-E4	f_{x1},f_{x2},f12	None	No
CM2-E5	f_{x1},f_{x2},f12,f20	c1,c2	Yes
CM2-E6	f_{x1}, f_{x2},c1	c2,Hidden	Yes
CM2-E7	f_{x1}, f_{x2},c1,f12	c2,Hidden	No

[1] Suppose in CM2 the inflow u=<pos,zer>.
[2] with additional domain-specific knowledge.

to the actual number of hidden variables). If QML-BKFC can successfully identify the target model, the succeeding kernel subset experiments will be carried out.

For experiments on models CM1 and CM2, the learning precision of the successful experiments is shown in Figures 9 and 10, respectively.

For experiments on model CM1, the same learning precision and kernel subsets were obtained in experiments CM1-E1, E2, E5, and E7. Experiment CM1-E4.b has different learning precision because an additional assumption has been made, and its kernel subsets only include two pairs: (1,3), (2,3).

For experiments on model CM2, CM2-E1, E3, E5, and E6 have the same learning precision and kernel subsets. CM2-E2 has a lower learning precision and its kernel subsets are 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). The first five triples are the same as those in CM2-E1. The last three sets are constructed by adding state 6 into the remaining three triples in CM2-E1.

Some conclusions can be drawn based on all experiments reported in this section: (1) The state variables are very important for learning. The learning task will become difficult if some of them are hidden variables. (CM1-E3, CM1-E4, CM1-E6, CM2-E6, and CM2-E7) (2) If a learning task cannot be successfully accomplished, more domain specific knowledge can be added to facilitate the learning. Given enough information, it is still possible to learn the target models, but the kernel subsets and learning precision may change. (CM1-E4.a, and CM1-E4.b) (3) The specification of state variables is also a factor which influences the learning. Partially or not specifying the state variables will result in a large search space and may lead to unsuccessful experiments. (CM1-E3,E4,E8; CM2-E3, E4, E6) (4) Given the correct number of hidden variables, and fully specifying the state variables, the non-state variables have the least influence on learning. This can be seen from experiments CM1-E1, E2, E5, and also from experiments CM2-E1, E3, E5. (5) If state variables are not specified, too

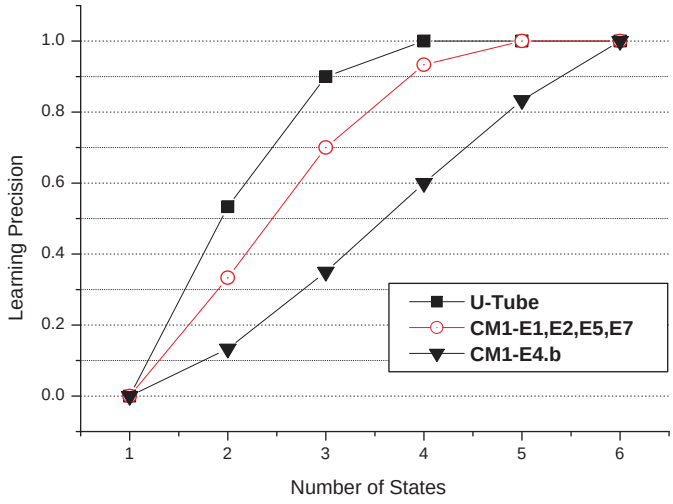

Figure 9: The Learning Precision of Experiments U-Tube, CM1-E1, E2, E5, E7, and E4.b

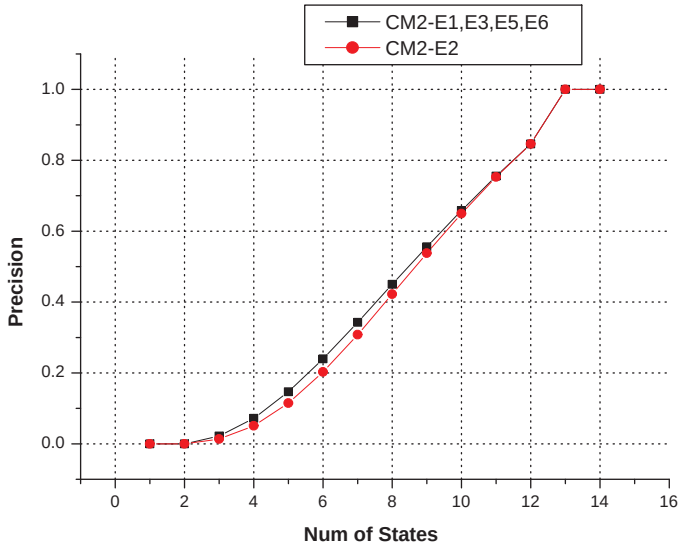

Figure 10: The Learning Precision of Experiments CM2-E1, E2, E3, E5, and E6

Table 12: The Description of the Scalability Experiments

Experiment ID	Hidden Variables	Num. Of States	Search Space
CM2-Ex3-E1	f_{x3}	68	$6.95*10^8$
CM2-Ex3-E2	f_{x2}, f_{x3}	48	$4.81*10^{10}$
CM2-Ex3-E3	f_{x1}, f_{x2}, f_{x3}	48	$6.31*10^{11}$
CM2-Ex4-E1	None	500	$6.55*10^4$
CM2-Ex4-E2	f_{x4}	340	$4.22*10^{12}$

Table 13: Experimental Results of the Scalability Test

Experiment ID	QML-BKFC Run Time (millisecond)	QML-CSA Best Run Time (millisecond)	QML-CSA Average Run Time (millisecond)
CM2-Ex3-E1	2,504,080	9,990	928,616
CM2-Ex3-E2	193,327,522	500,826	12,135,336
CM2-Ex3-E3	>604,800,000	42,060,168	205,663,188
CM2-Ex4-E1	34,858	6,281	6,554
CM2-Ex4-E2	>1,728,000,000	126,592,860	687,477,140

many hidden variables can lead to unsuccessful learning. This can be seen from experiments CM1-E7, E8, and also from experiments CM2-E2, E4.

6.4 Scalability Test

In this section the scalability of QML-Morven will be investigated when learning complicated models. QML-BKFC and QML-CSA were performed to learn models CM2_Ex2 and CM2_Ex3 shown in Figure 7, and their learning results were compared.

A series of experiments was designed, as shown in Table 12. In all these experiments: (1) The data of one and a half differential planes were used for comparison with the experimental results obtained from ILP-QSI; (2) All state variables were specified, and the complete data were used. (3) f_{x1}, f_{x2}, f_{x3}, and f_{x4} stand for the net flow of *c1*, *c2*, *c3* and *c4* respectively. (4) It is also assumed that the correct number of hidden variables is given.

For QML-BKFC, the *full* search mode is adopted. For QML-CSA, the population size is 100 for CM2-EX4-E1 and 1000 for all other experiments, the clonal size is 10, the surviving time for all antibodies is 1000 generations, and the hyper-mutation probability is 0.5. The value of the hyper-mutation probability is a classical one used in the original CSA. The values of other parameters in the experiments are determined according to the complexity of the search space. For both framework instances, the termination condition is that the target model is found.

The experimental results are shown in Table 13. In this table, experiments CM2-Ex3-E1 and CM2-Ex4-E1 were performed on a Dell PC with an Intel Pentium IV 3.0GHz CPU and 2GB RAM. Other experiments were tested on a computer cluster with 8 compute nodes (each node has two Opteron 850 (2.4GHz) CPUs and 4GB RAM). For QML-CSA, each experiment was executed ten times and the best and average performances were recorded. The details of

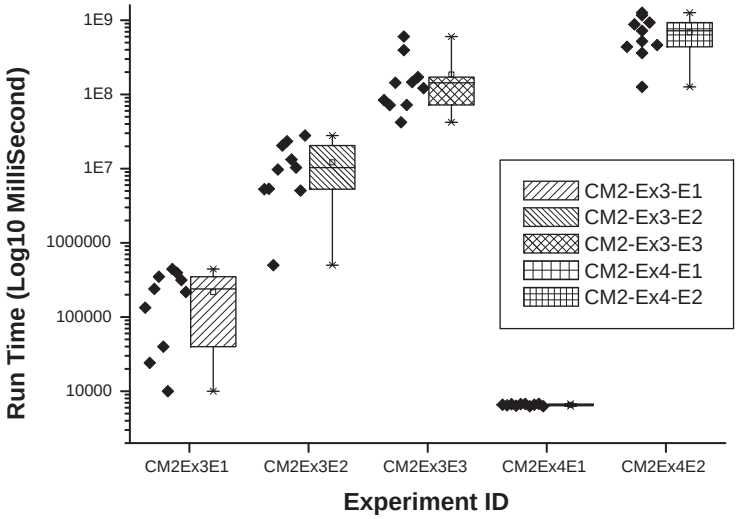

Figure 11: The Ten Trials of QML-CSA for Each Experiment

the ten trials of QML-CSA for each experiment are shown in Figure 11.

From the results one can see that when learning models with large search spaces, compared to QML-BKFC, QML-CSA demonstrates its scalability, especially in experiments CM2-Ex3-E3 and CM2-Ex4-E2. In these two experiments QML-BKFC cannot find any solution within the specified time, while QML-CSA can find the target model in a relative short time. The high efficiency of QML-CSA is because it combines the random exploration and fine exploitation when searching the model space, and these two mechanisms are provided by the clonal expansion and hyper-mutation operators, as well as the problem-dependent affinity evaluation function and the surviving time constraints for each antibody.

7 Conclusion and future work

In this report, we presented a novel qualitative model learning framework, QML-Morven. QML-Morven can perform the same learning tasks as its predecessors such as ILP-QSI: learning from positive only data, utilising the well-posed model constraints to narrow the search space, analysing the kernel subsets and learning precision, and dealing with hidden variables. Furthermore, QML-Morven is an extensible framework that consists of three QML sub-systems, and it allows both deterministic and evolutionary approaches applied as learning strategies to learning under various situations.

More importantly, in this research a series of experiments have been performed with QML-Morven to reveal the influence of two important factors on the learning: the specification of state variables and hidden variables. The results obtained and the conclusions drawn from these results offer an insight into the basic principles of qualitative model learning and may make contributions to the future development of QML systems.

Finally the scalability of the QML system was addressed by employing an evolutionary algorithm as the search strategy in QML-CSA and promising experimental results were ob-

tained.

The three QML subsystems implemented within the QML-Morven framework are adapted to a wide range of problems: QML-GENMODEL is employed to learn simple systems of which all variables are identified and complete qualitative behaviours are obtained. QML-BKFC and QML-CSA are utilised when there exist hidden variables and (or) only incomplete behaviours are available. In addition, QML-BKFC is suitable for small and medium-sized problems, and as a complete algorithm, it offers more precise search on the model space compared to QML-CSA: QML-BKFC can guarantee to find a solution if one exists. On the other hand, QML-CSA performs better on large-sized problems and addresses the scalability of QML. These three QML subsystems make QML-Morven very flexible to various problems.

The future development of QML-Morven will involve the following: First, learn fuzzy qualitative models. QML-Morven has the potential to learn fuzzy qualitative models because the Morven framework employed by QML-Morven can represent such models and perform fuzzy qualitative simulation. Second, learn models of more complex real-world problems. To begin with, QML-Morven has been used to reconstruct qualitatively a biological pathway [37]. Third, investigate the feasibility of other evolutionary and immune-inspired algorithms for learning large-scale problems. Some pilot work [39] has been done to investigate how to adapt the opt-AiNet [16, 51], an immune network algorithm, to qualitative model learning. It has shown that at the current research stage the opt-AiNet approach to QML is very promising and may outperform QML-CSA if more modifications are made to its operators in the future. We expect that in the future more evolutionary and immune-inspired algorithms will be adapted to qualitative model learning and the learning reliability and performance of these algorithms will be further analysed and compared.

References

[1] Francesco Bergadano and Daniele Gunetti. *Inductive Logic Programming From Machine Learning to Software Engineering*. MIT Press, 1996.

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

[3] R.B. Blackman and J.W. Tukey. *The measurement of Power Spectra*. Dover Publications Inc, 1958.

[4] Ivan. Bratko, Stephen. Muggleton, and Alen. Varšek. Learning qualitative models of dynamic systems. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 385–388. Morgan Kaufmann, 1991.

[5] Fred Brauer. *Mathematical epidemiology*, chapter 2, pages 19–80. Springer, 2008.

[6] 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.

[7] Allan M. Bruce. *JMorven: A Framework for parallel non-constructive qualitative reasoning and fuzzy interval simulation*. PhD thesis, Department of Computing Science, Univeristy of Aberdeen, October 2007.

[8] R. Camacho. *Inducing Models of Human Control Skills using Machine Learning Algorithms*. PhD thesis, University of Porto, 2000.

[9] George M. Coghill. Fuzzy envisionment. In *Proceedings of the Third International Workshop on Hybrid Methods for Adaptive Systems*, Oulu, Finland, 1992.

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

[11] George M. Coghill, A.J. Asbury, C.J. van Rijsbergen, and W.M. Gray. The application of vector envisionment to compartmental systems. In *Proceedings of the first international conference on Intelligent Systems Engineering*, pages 123– 128, Edinburgh, Scotland, 1992.

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

[13] George M. Coghill, Ashwin Srinivasan, and Ross D. King. Qualitative system identification from imperfect data. *Journal of Artificial Intelligence Research*, 32:825–877, 2008.

[14] E.W. Coiera. Generating qualitative models from example behaviours. Technical Report DCS Report 8901, Department of Computer Science, University of New South Wales, Sydney, Australia, 1989.

[15] E.W. Coiera. Learning qualitative models from example behaviours. In *Proc. Third Workshop on Qualitative Physics*, pages 45–51, Stanford, August 1989.

[16] de Castro and Jon Timmis. An artificial immune network for multimodal function optimization. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC'02)*, pages 674–699. IEEE Press, 2002.

[17] de Castro and Von 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.

[18] Johan de Kleer and John Seely Brown. Theories of causal ordering. *Artificial Intelligence*, 29:33–61, 1986.

[19] Stephen P. Ellner and John Guckenheimer. *Dynamic Models in Biology*, chapter 1, pages 1–28. Princeton University Press, 2006.

[20] David A. Fields and Michael I. Goran. Body composition techniques and the four-compartment model in children. *Journal of Applied Physiology*, 89(2):613–620, 2000.

[21] Franois Gillet, Olivier Besson, and Jean-Michel Gobat. Patumod: a compartment model of vegetation dynamics in wooded pastures. *Ecological Modelling*, 187(3):267–290, 2002.

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

[23] Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

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

[25] Yumi Iwasaki and Herbert A. Simon. Theories of causal ordering: Reply to de kleer and brown. *Artificial Intelligence*, 29:63–72, 1986.

[26] Yumi Iwasaki and Herbert A. Simon. Causality and model abstraction. *Artificial Intelligence*, 67(1):143–194, 1994.

[27] R. D. King, S. M. Garrett, and G. M. Coghill. On the use of qualitative reasoning to simulate and identify metabolic pathways. *Bioinformatics*, 21(9):2017–2026, August 2005.

[28] Ina Kraan, Bradley L. Richards, and Benjamin Kuipers. Automatic abduction of qualitative models. In *Proceedings of the Fifth International Workshop on Qualitative Reasoning about Physical Systems*, pages 295–301, 1991.

[29] Benjamin Kuipers. Modeling and simulation with incomplete knowledge. *Automatica*, 25(4):571–585, 1989.

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

[31] Lennart Ljung. *System Identification – Theory For the User*. Prentice Hall, Upper Saddle River, N.J., 2nd edition, 1999.

[32] E. McCreath. *Induction in first order logic from noisy training samples and fixed sample sizes*. PhD thesis, University of New South Wales, 1999.

[33] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

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

[35] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan, 1990.

[36] Wei Pang and George M. Coghill. Modified clonal selection algorithm for learning qualitative compartmental models of metabolic systems. In Dirk Thierens, editor, *Genetic and Evolutionary Computation Conference (GECCO07)*, pages 2887–2894, New York, NY, USA, 2007. ACM Press.

[37] Wei Pang and George M. Coghill. An immune-inspired approach to qualitative system identification of the detoxification pathway of methylglyoxal. In Paul Andrews and Jon Timmis, editors, *LNCS 5666, Proceedings of 8th International Conference on Artificial Immune Systems (ICARIS 2009)*, pages 151–164, York, UK, 2009. Springer.

[38] Wei Pang and George M. Coghill. Learning qualitative differential equation models: a survey of algorithms and applications. *The Knowledge Engineering Review*, 25(1):69–107, 2010.

[39] Wei Pang and George M. Coghill. QML-AiNet: An immune-inspired network approach to qualitative model learning. In Emma Hart et al., editor, *LNCS 6209, Proceedings of 9th International Conference on Artificial Immune Systems (ICARIS 2010)*, pages 223–236, Edinburgh, UK, 2010. Springer.

[40] Christos H. Papadimitriou and Ken Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, 1982.

[41] Chris Price, Louise Trave-Massuyes, Rob Milne, Liliana Ironi, Kenneth Forbus, Bert Bredeweg, Mark Lee, Peter Struss, Neal Snooke, Peter Lucas, Marc Cavazza, and George Coghill. Qualitative futures. *The Knowledge Engineering Review*, 21(4):317–334, 2006.

[42] Bradley L. Richards, Ina Kraan, and Benjamin Kuipers. Automatic abduction of qualitative models. In *National Conference on Artificial Intelligence*, pages 723–728, 1992.

[43] Bradley L. Richards and Raymond J. Mooney. Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 50–55, San Jose, CA, July 1992.

[44] Wolfgang A. Ritschel and Gregory L. Kearns. *Handbook of Basic Pharmacokinetics Including Clinical Applications*. American Pharmacists Association, Washington, DC, seventh edition, 2009.

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

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

[47] Qiang Shen and Roy Leitch. Fuzzy qualitative simulation. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(4):1038–1061, 1993.

[48] Terry E Shoup. *A practical guide to computer methods for engineers*. Englewood Cliffs, N.J. : Prentice-Hall, 1979.

[49] Ashwin Srinivasan. The aleph manual. `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html`, 1999.

[50] Ashwin Srinivasan and Ross D. King. Incremental identification of qualitative models of biological systems using inductive logic programming. *Journal of Machine Learning Research*, 9:1475–1533, 2008.

[51] Jon Timmis and C. Edmonds. A comment on opt-AINet: An immune network algorithm for optimisation. In D. Kalyanmoy, editor, *Genetic and Evolutionary Computation (GECCO 2004),Lecture Notes in Computer Science*, volume 3102, pages 308–317, 2004.

[52] A. Varšek. Qualitative model evolution. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia, August 1991.

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

[54] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.