Learning qualitative models

Ivan Bratko, Dorian Šuc
Faculty of Computer and Information Sc.,
University of Ljubljana, Slovenia
Published in *AI Magazine*, 2003, vol. 24, no. 4, pp. 107-119, © AAAI Press

In general, modelling is a complex and creative task, and building qualitative models is no exception. One way of automating this task is by means of machine learning. Observed behaviours of a modelled system are used as examples for a learning algorithm which constructs a model that is consistent with the data. In this paper we review approaches to the learning of qualitative models, either from numerical data or qualitative observations. We describe the QUIN program that looks for qualitative patterns in numerical data, and outputs the results of learning as "qualitative trees". We illustrate this by applications associated with systems control, in particular the identification and optimisation of controllers and of human operator's control skill. We also review approaches that learn models in terms of qualitative differential equations.

Introduction

Much research in the field of qualitative reasoning has been devoted to the questions of representation of qualitative models, and to qualitative simulation algorithms that derive qualitative behaviours from given qualitative models. However, an important practical question is how to construct qualitative models in the first place. In general, model construction is usually the most demanding aspect of the modelling task. In this paper we look at research that aims at automating this task.

One idea is to use observations of the modelled system, obtained through measurements, and try to find a model that, when simulated, would reproduce the same, observed behaviours.

This task is known as *system identification*, and is just the opposite of system simulation. Of course, to be of interest, the model induced from observations should be more general than the observations themselves. The induced model should be capable of making predictions also in situations other than those literally included among the observations. This task can also be viewed as machine learning from examples. The observed system behaviours are taken as examples for a learning algorithm, and the result of learning, usually called a *theory*, or a *hypothesis* induced from the examples, represents a model of the system.

In this paper we consider the particular problem of inducing a *qualitative* model from examples of system behaviours. We first look at a recently developed approach based on the induction of *qualitative trees*. Then we present an application of this technique to problems associated with the control of dynamic systems. One such application is the qualitative identification of industrial controllers, which can also be viewed as qualitative reverse engineering. Another application is in the identification of tacit control skills of human operators. In the last part of the paper we review some other representative approaches to the learning of qualitative models.

It should be noted that, in comparison with traditional (quantitative) system identification, in "qualitative system identification" there is much more emphasis on obtaining comprehensible models, models that intuitively explain how the system works.

Qualitative data mining with QUIN

QUIN (Qualitative Induction) is a learning program that looks for qualitative patterns in numerical data (Šuc 2001; Šuc and Bratko 2001). Induction of the so-called qualitative trees is similar to the well-known induction of decision trees (e.g. CART, Breiman et al. 1984; C4.5, Quinlan 1993). The difference is that in decision trees the leaves are labelled with class values, whereas in qualitative trees the leaves are labelled with what we call *qualitatively constrained functions*.

Qualitatively constrained functions (QCFs for short) are a kind of monotonicity constraints that are widely used in the field of qualitative reasoning. A simple example of QCF is: $Y = M^{+}(X)$. This says that Y is a monotonically increasing function of X. In general, QCFs can

have more than one argument. For example, $Z = M^{+,-}(X,Y)$ says that Z monotonically increases in X and decreases in Y. If both X and Y increase, then according to this constraint, Z may increase, decrease or stay unchanged. In such a case, a QCF cannot make an unambiguous prediction of the qualitative change in Z. In the literature, QCFs appear also under the term "multivariate monotonic function constraints" (Wellman 1991).

QUIN takes as input a set of numerical examples and looks for qualitative patterns among the data. More precisely, QUIN looks for regions in the data space where monotonicity constraints hold. Such a set of qualitative patterns are represented in terms of a qualitative tree. As in decision trees, the internal nodes in a qualitative tree specify conditions that split the attribute space into subspaces. In a qualitative tree, however, each leaf specifies a QCF that holds among the input data that fall into that leaf. Figure 1a shows an example data set with three variables X, Y and Z. The data points correspond to the function $Z = X^2 - Y^2$ with some Gaussian noise added. When QUIN is asked to find in these data qualitative constraints on Z as a function of X and Y, QUIN generates the qualitative tree shown in Figure 1b. X and Y are independent variables, also called attributes, and Z is the dependent variable, also called class. This tree partitions the data space into four regions that correspond to the four leaves of the tree. A different QCF applies in each of the leaves. The tree describes how Z qualitatively depends on X and Y. Notice that noise in the data in this example did not present problems to QUIN.

QUIN constructs a tree in a top-down greedy fashion, similarly to decision tree induction algorithms. At each internal node of the tree, QUIN considers all possible splits, that is conditions of the form $X < T$ for all the attribute variables X and effectively all possible thresholds T with respect to X. Each such condition partitions the training data into two subsets. QUIN finds the "best" QCF for each subset according to an error-cost measure for QCFs. Then the best split is selected according to the MDL (minimum description length) principle, which minimizes the error-cost and the encoding complexity of QCFs. The error-cost of a QCF with respect to an example set S is defined so that it takes into account the consistency of the QCF with S, and the "ambiguity" of the QCF with respect to the data in S (the more unambiguous qualitative predictions the QCF can make in S the better). Technical details of all this can be found in (Šuc 2001) or (Šuc and Bratko 2001) where QUIN's performance on noisy data is also studied.

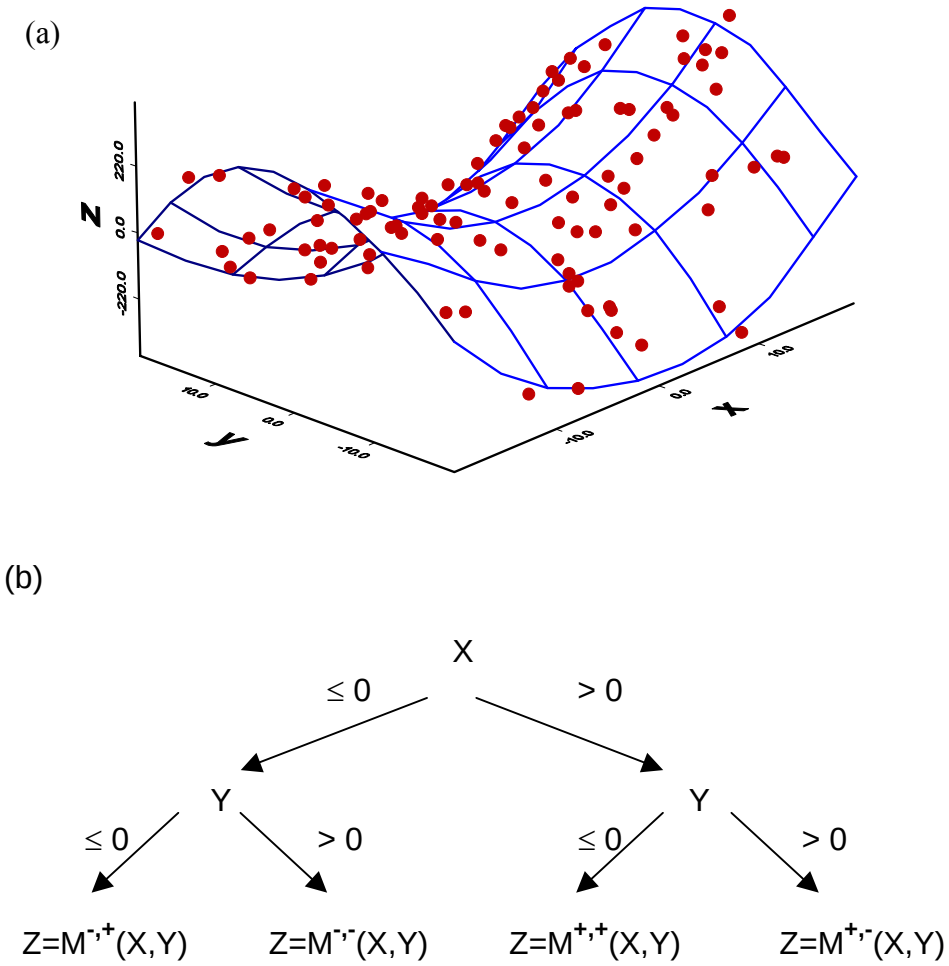

Figure 1: (a) A data set of points where $Z = X^2 - Y^2$ + some noise; (b) A qualitative tree induced by QUIN from this data set.

A qualitative reverse engineering application of QUIN

In this section we present an application of QUIN to reverse engineering of an artefact. The task of reverse engineering is in a sense formally equivalent to system identification, although the application context is different.

First we explain the motivation for this kind of reverse engineering applications, as observed in the European project Clockwork. This project aims at creating tools to support engineering design. Accumulated engineering design knowledge in a company often takes the form of a library of designs and corresponding simulation models. A typical common problem with such libraries is incomplete documentation. Such libraries contain numerous versions of

models (designs) where comparative advantages and drawbacks of alternative models are not well documented. Re-use of designs is made difficult specially because the intuitions behind designs and their improvements are not explained in the documentation. Although there may be complete mathematical models and working simulation programs included in the library, the user of the library is impeded by lack of understanding of how does the designed system work. What are the basic ideas of a design? For example, how does a controller of a dynamic system achieve the goal of control? What is the idea behind the improvement in an alternative design?

Here we show how the task of recovering the underlying ideas of designs can be tackled by qualitative machine learning, in particular using the QUIN program. We assume a model in an engineering library is complete so that it can be executed on a simulator. The simulated system can thus be observed as a black box, but the internal structure of the system is obscure to the user because it is too complex to be understood without explanation. To help the user develop some intuitive understanding of how the black box works, machine learning tools can be used to analyse the behaviour of the variables in the model and detect meaningful relations among these variables. We refer to this as *qualitative reverse engineering* (Šuc and Bratko 2002). This task is formally similar to qualitative system identification (Say and Kuru 1996), although the context may be completely different.

We will illustrate an approach to qualitative reverse engineering by an application from the control of gantry cranes (Figure 2). The task is to move the load from some start position to a goal position. The goal position can, for example, be a truck. For safety, when the load is entering the truck, it should not be swinging. The criterion is to transfer the load as quickly as possible, that is at high velocity. However, high velocity requires large acceleration, which in turn causes large swing of the load. The controller should carry out acceleration manoeuvres in such a way as to minimize the swing, so it should be capable of controlling the swing under large accelerations. This is the difficult aspect of the crane control task.

M. Valašek designed an “anti-sway” industrial controller for gantry cranes (Valašek et al. 1996). This controller is now in every day use in cranes in a Czech metallurgical factory (Valašek, personal communication). The controller helps the crane operator to easily control the crane carriage velocity without causing large swing of the load. The operator specifies the

desired carriage velocity. The controller works between the crane and the operator, and computes for the given desired velocity a control force that achieves a good approximation to

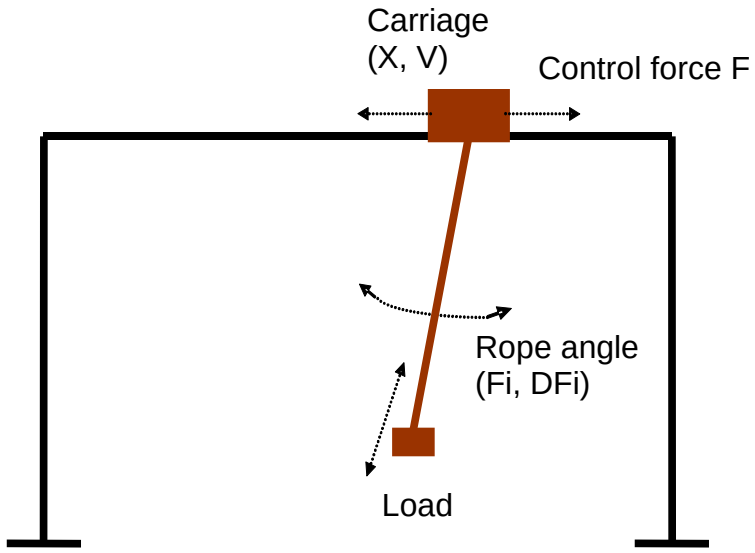

Figure 2: Gantry crane.

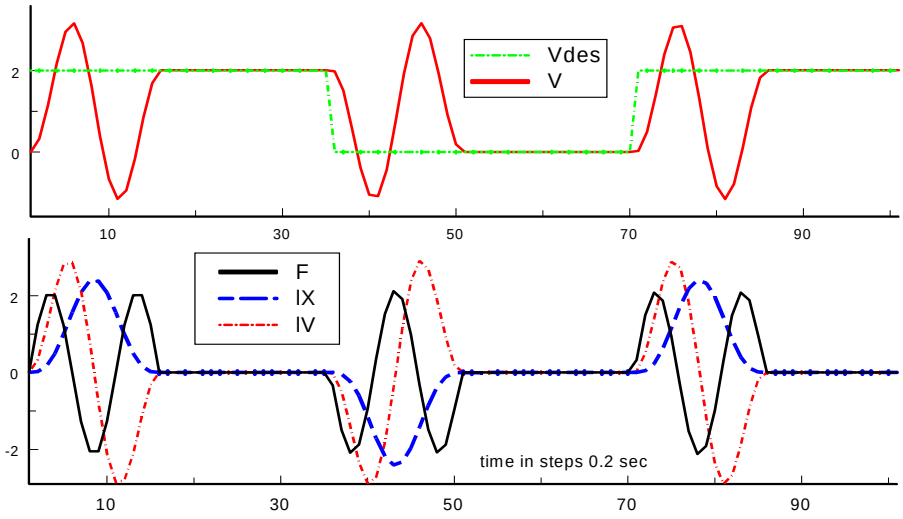

Figure 3: Execution trace in time of the anti-sway controller.

the desired velocity, but does this in such a way that it only causes a small swing of the load. Therefore this controller is also called the “anti-sway crane”. Figure 3 gives an example execution trace of the desired velocity Vdes, controller’s action (force F in time), the actual crane’s velocity V, rope angle Fi and angular velocity DFi. This trace shows that the controller achieves the desired velocity in time causing only one sway of the load, without any periodic oscillation. To prevent oscillation, the force changes in time in a non-trivial way, and so does the actual velocity.

Now consider that this controller is given as a black box. Its inputs and outputs can be observed, but there is no documentation about how it works. The problem is to reverse engineer the controller, given some observed control traces such as that in Figure 3.

In this paper we are particularly interested in extracting from control traces such a description of the underlying controller that uncovers the basic intuition about how the controller works. To this end we seek to recover from control traces a *qualitative* model of the controller since such models usually provide better explanation of how the system works than other types of models. The task of qualitative reverse engineering for our crane case is then defined as: given examples of time behaviours of Vdes, F, X, V, Fi, DFi, find a qualitative relation between the control force F and the other quantities.

Let us illustrate how a control strategy can be described qualitatively, using typical formalisms in qualitative physics. For example, consider the crane at rest in the initial state. To start the crane moving, both velocity V and position X should be increasing simultaneously. This can be stated by the usual qualitative constraint:

$$V = M^{+}(X)$$

It should be noted that this is not a law of the physics of the crane system, but it is a *control* law enforced by a controller. Another qualitative rule about controlling the swing may be: the greater the rope angle and the faster it is increasing, the greater should be the carriage velocity to "catch up" with the angle. This can be stated by the QCF:

$$V = M^{+,+}(Fi,DFi)$$

The controller identification task can be formulated in various ways, depending on which variable is the class, and what variables are included among the attribute variables. Figure 4 shows an example qualitative tree induced by QUIN from the execution trace of Figure 3. This tree gives the qualitative constraints on the function that maps the variables X, V, Fi, DFi and the relative carriage velocity Vrel (relative to the desired velocity Vdes, Vrel = V – Vdes) into the control force F.

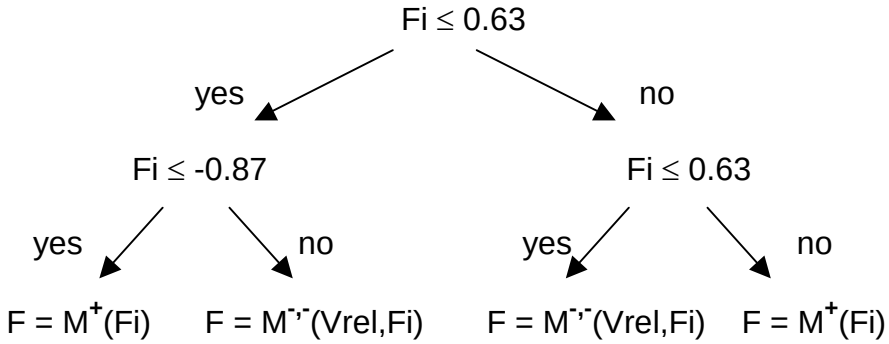

Figure 4: A qualitative controller induced from the trace of Figure 3. The tree shows how the control force F qualitatively depends on the rope angle Fi and the relative carriage velocity Vrel = V-Vdes (the difference between the actual velocity and the desired velocity).

The qualitative tree of Figure 4 exposes some interesting properties of the anti-sway control strategy. When the rope angle is large positive or negative, then the controller takes care about the angle. When the angle is small, then the carriage is pushed in the direction of desired velocity and, surprisingly, increasing the absolute angle of the rope.

A qualitative control strategy, such as that in Figure 4, cannot be directly used as a controller because it does not determine a precise, numerical value of the control force. The qualitative tree just tells that, for example, the greater the rope angle, the lesser the control force. To make a qualitative control strategy operational, we have to transform the QCFs in the leaves into actual numerical functions. The QCFs constrain the choice of these functions. The problem of this qualitative-to-quantitative transformation (Q2Q transformation) can be viewed as an optimization problem. The optimization criterion has to be defined in such a way that it maximizes the fit in time between the actual carriage velocity and the desired velocity, and minimizes the swing in time. One such optimization procedure to solve this optimization problem is described in (Šuc 2001; Šuc and Bratko 2000b). Experiments described in (Šuc and Bratko 2002) with this optimization procedure applied on qualitative control strategies for the anti-sway crane show that the so obtained reconstructed controllers perform comparably to the original controller.

For comparison, an interesting question is: Was it essential to first reconstruct the control strategy from quantitative data qualitatively, and then transform it into a quantitative strategy? Or would a straightforward numerical reconstruction, using numerical regression, be equally successful? The most natural machine learning method for this is induction of regression trees

(Breiman et al. 1984), or its variant model trees (Quinlan 1992). The experiments with model trees on the same task, also reported in (Šuc and Bratko 2002), somewhat surprisingly did not lead to a successful reconstruction of the anti-sway controller. It is not quite clear yet why the straightforward numerical learning failed when qualitative learning (combined with the Q2Q transformation) succeeded. Just studying the learning data from the execution trace suggests the following answer to be likely: the numerical data are rather regression-unfriendly and it is hard for the regression procedure to decide whether some relatively small numerical differences are just due to noise or numerical perturbations, or they reflect genuine dependences among the variables. On the other hand, it seems that QUIN is more robust at detecting subtle qualitative dependences because it pays less attention to the absolute values of numerical changes. In other words, what numerically looks like noise, may qualitatively be significant. This conjecture requires further investigation.

Identification of operator's skill

Human operator's control of a complex dynamic system, such as a crane or an aircraft, requires skill acquired through experience. Imagine that we have a skilled crane operator, then some questions of interest are: How does he or she do it? Can we understand such a tacit human skill? Can we reconstruct the skill as an automatic controller and further optimize it? One attempt would be to extract the skill from the operator in a dialogue fashion whereby the operator would be expected to describe his own skill. This description would then be appropriately formalised and built into an automatic controller.

The problem with this approach is that the skill is sub-cognitive and the operator is usually only capable of describing it incompletely and approximately. The operator's descriptions are not operational in the sense of being directly translatable into an automatic controller. Such difficulties of skill reconstruction through introspection in crane control were experimentally studied in (Urbančič and Bratko, 1994; Bratko and Urbančič 1999).

Given the difficulties of skill reconstruction through introspection, an alternative approach is to identify the skill from the *manifestation* of the skill. The skill is manifested in the form of traces of the operator's actions. One idea is to use these traces as examples for Machine Learning, and extract operational models of the skill by Machine Learning techniques. This is also known as *behavioural cloning* (Michie 1993; Michie et al. 1990).

One goal of behavioural cloning is to generate *good performance* clones, that is those that can reliably carry out the control task. Such clones can replace the original human operator. Often it turns out that a clone, while carrying out the task in a similar style to the operator, actually performs better and more consistently than the operator.

Performance improvement is, however, not the only goal of behaviour cloning. Another important goal is to generate *meaningful* clones, in order to help us to understand the operator's skill. Understanding what exactly a human operator is doing and why may be of practical importance. We may capture the skill of an outstanding operator and transfer it to less gifted operators.

The operator's control strategy should ideally be understood in terms of goals, sub-goals, plans, feedback loops, causal relations between actions and state conditions etc. These conditions are to be stated in terms of information that is easily accessible to the operator, e.g. visually. It can be argued that such information should be largely qualitative, in contrast to numerical. In the next section we show how qualitative descriptions of control skills can be induced by QUIN from operator's control traces.

Looking for qualitative patterns in dynamic behaviours

Figure 5 shows an execution trace of a human operator controlling a simulated crane. This was one of the most successful (in terms of time to complete the task) human controls among a number of human subjects that in an experimental study learned to control this crane simulator. In particular, this operator was able to afford large accelerations, causing large swing of the load, because he was capable of controlling the swing and reducing it when approaching the goal position.

Formally, the task of behavioural cloning is: Given a trace such as that in Figure 5, find the operator's "control strategy". That is: Find a rule that for any given dynamic state of the crane determines appropriate control actions. The possible actions are the horizontal control force Fx acting on the carriage, and the vertical control force Fy pulling the rope. Formally this means, find two functions

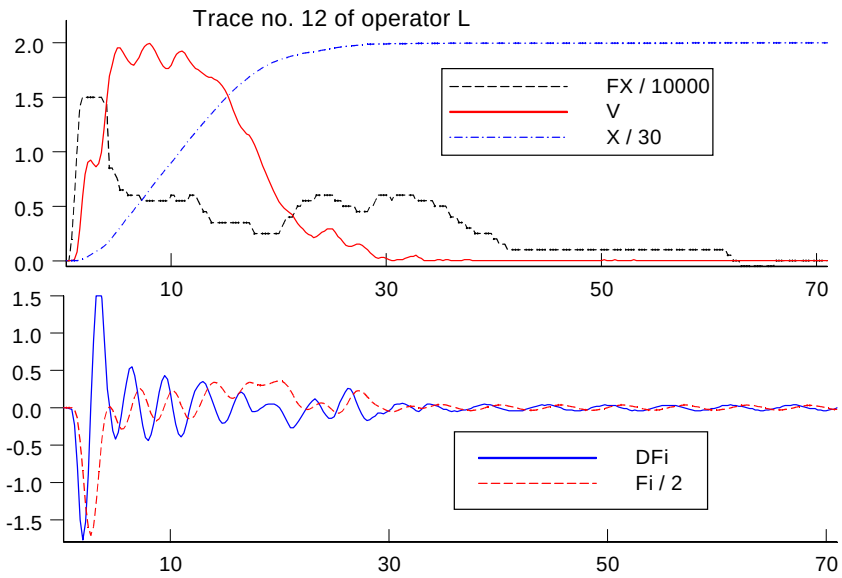

Figure 5: Execution trace in time of operator L.

Fx = Fx(X, V, Fi, DFi, L, LV)

Fy = Fy(X, V, Fi, DFi, L, LV)

where X, V, etc. are the state variables of the dynamic system (carriage position, carriage velocity, rope angle, angular velocity, length of the rope and length velocity).

In the qualitative approach to this task we first try to identify the control strategy qualitatively. We look for qualitative patterns in the execution trace data that tell us something about how the operator controls the system. For example, one qualitative property that is easy to see in this trace is that initially, when X is small and increasing, the velocity V is also small and increasing. This relation can be written as $V = M^{+}(X)$ (V is a monotonically increasing function of X). Notice that this qualitative statement does not tell anything about the precise numerical values of the two variables X and V. But it does tell us that initially, to start the crane moving, the carriage velocity is to increase.

Other, more subtle properties of the operator's strategy that QUIN detected in the data of Figure 5 are shown in Figure 6 (the tree on the right). These properties tell us how the operator goes about controlling the swing. Also, such properties induced from traces of different operators point out qualitative differences in the control styles of the operators.

Figure 6 shows two qualitative trees induced by QUIN from control traces of operators S and L. Operator S controls the crane very cautiously, avoiding large velocities and accelerations, and therefore never producing large swinging of the load. This conservative strategy is reliable, but not very efficient. It is slow and requires large time to complete the task of transferring the load from a start to a goal. In contrast to this, operator L is more adventurous and does not dodge large accelerations. This causes large swing, but operator L can afford this because he is capable of confidently reducing the swing when necessary. This enables L to achieve much shorter completion times than S.

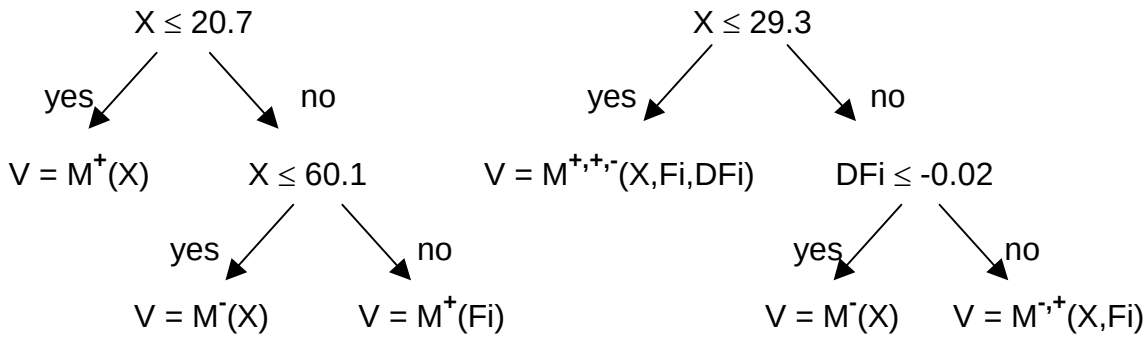

Figure 6: Left: Qualitative strategy of operator S; Right: Qualitative strategy of operator L. The trees show how the target carriage velocity qualitatively depends on the carriage position X, rope angle Fi and rope angular velocity DFi.

Figure 6 nicely exposes the differences in the control styles of both operators. Although their corresponding qualitative trees have similar structure, they significantly differ in the QCFs in the leaves. Looking at the corresponding QCFs, it is obvious that S's conservative strategy is much simpler than that of L. For example, the left-most leaf of the left tree in Figure 6 shows that, at the starting stage of the task when X is small, S just keeps increasing velocity (in a cautious way) and does not pay attention to the rope angle. This is expressed by the constraint in the left-most leaf $V = M^{+}(X)$ (when carriage position X is increasing, the target carriage velocity is also increasing). Only much later, when close to the goal, S starts paying attention to the angle. This can be seen in the right-most leaf of S's tree which says $V = M^{+}(Fi)$. This principle of controlling the swing can be intuitively explained as: when the rope angle is increasing, that is the load swinging to the right, accelerate the carriage to "follow" the load and thus reduce the angle. Operator L, on the other hand, considers the angle and angular velocity already at the early stage: the left-most leaf of L's tree says that the carriage velocity should depend also on the rope angle Fi and angular velocity DFi, as well as on X.

As in the case of the anti-sway crane, "qualitative clones" cannot be directly applied to the control of the crane. Again, we need a transformation of the QCFs into real-valued functions. This time, the natural optimisation criterion is the task completion time. It should also be noted that the trees in Figure 6 only suggest the control actions indirectly. Namely, a tree only determines the desired velocity and not the control force. The reconstruction of skill therefore also requires the learning of a simple local model of the crane's dynamics. This model is then used to determine a control force that achieves the desired velocity. The resulting "indirect controllers" (Šuc, Bratko 2000a) carry out the task in style that is qualitatively equal to that of the corresponding human operators, but typically perform better in terms of the evaluation criterion (Šuc 2001).

Learning models in terms of qualitative differential equations

So far we have discussed the learning of qualitative models represented as qualitative trees. In this section we will look into learning models expressed as Qualitative Differential Equations (QDEs). There have been a number of attempts at automatically constructing QDE models from examples of system's behaviour. In the remainder of this paper we will review this work. For completeness we first give a brief introduction to QDEs.

QDEs are an abstraction of ordinary differential equations. In simulation based on QDEs, time is usually treated differently than other variables. An example is the QSIM qualitative simulation algorithm (Kuipers 1986; 1994) which is largely based on the assumption that variables in the QDE model behave continuously and smoothly in time. QDE models are usually written as sets of constraints of the following types:

$Y = M^{+}(X)$	Y is a monotonically increasing function of X
$Y = M^{-}(X)$	Y is a monotonically decreasing function of X
add(X, Y, Z)	Z = X + Y
minus(X, Y)	Y = -X
mult(X, Y, Z)	Z = X * Y
deriv(X, Y)	Y = dX/dt

All these constraints are applied to "qualitative states" of variables rather than on variables' numerical values. For example, such a qualitative state of variable X can be: positive and increasing in time, written as: X = (pos,inc). The possible directions of change are "inc" (for increasing), "std" (for steady), and "dec" (for decreasing). The add(X,Y,Z) constraint is satisfied, for example, by the following qualitative states: X = (pos,inc), Y = (pos,inc), Z= (pos,inc). For some values of X and Y, Z is non-determined. So if X=(pos,inc) and Y=(neg,dec), then Z can be (pos,inc), or (zero,inc), or (pos,std), or (neg,inc), etc.

A QDE model is defined by a set of variables, their possible qualitative values, and a set of constraints among these variables. The problem of learning QDE models from example system behaviours is: given qualitative behaviours of a set of observed variables, find a QDE model, that is a set of qualitative constraints that are consistent with the given behaviours.

The basic QDE learning algorithm, introduced by Coiera (1989) in his program GENMODEL, constructs a model from examples as follows:

1. Construct all the syntactically possible constraints, using all the observed variables (that is those appearing in the example behaviours) and the given repertoire of types of qualitative constraints.

2. Evaluate all the constraints constructed in step 1 on all the qualitative system's states in the given example behaviours. Retain those constraints that are satisfied by all the states, and discard all other constraints. The set of retained constraints constitute the induced qualitative model.

It is possible to induce, using this simple method, correct models for some simple systems, such as the U-tube or the spring-mass oscillator. For example, consider a possible, somewhat simplified scenario of learning a qualitative model of the U-tube with GENMODEL. The U-tube system consists of two containers A and B, connected at the bottom by a thin pipe. So the three components form a U-shape. Suppose that initially there is some water in container A while container B is empty. The difference between the two water levels, LevA and LevB, causes a positive flow from A to B. Thus LevA will be decreasing and LevB increasing, until both levels become equal, and the flow becomes zero. This behaviour can be stated qualitatively as a sequence of three qualitative states of the three variables (Figure 7).

LevA	LevB	Flow
(pos,dec)	(zero,inc)	(pos,dec)
(pos,dec)	(pos,inc)	(pos,dec)
(pos,std)	(pos,std)	(zero,std)

Figure 7: Qualitative behaviour of the U-tube.

Now assume that this behaviour in time was observed through measurements. GENMODEL will generate the possible qualitative constraints among the three observed variables, and find that three of these constraints are satisfied in all three observed qualitative states. These three constraints in conjunction constitute the model induced by GENMODEL:

$$\text{LevB} = M^{-}(\text{LevA}),\ \text{add}(\text{LevB}, \text{Flow}, \text{LevA}),\ \text{deriv}(\text{LevB}, \text{Flow})$$

This is actually a correct model of the U-tube. However, in general the GENMODEL algorithm is very limited because it relies on some strong assumptions:

(1) That *all* the variables in the target model are observed, so they explicitly appear in the example behaviours. The problem is, what to do if not all the variables in the target model are observed. In such a case we say that a variable that should appear in the model is "hidden" (that is, it is not mentioned in the example behaviours). GENMODEL does not handle hidden variables.

(2) The approach is biased toward learning the most *specific models* in the sense that these models contain *all* the possible constraints that are satisfied by the example data. There is of course no guarantee that all these constraints should actually be part of the target model.

(3) The resulting model is assumed to apply to the *complete* state space of the dynamic system. This is not appropriate for the cases when the system can be in more than one "operating region". For example, consider water level increasing in a container. When the level reaches the top of the container, the level can no longer keep increasing, and the system starts behaving according to a different law.

The difficulty with hidden variables can be illustrated by the U-tube example when the two levels LevA and LevB are observed only. The GENMODEL algorithm finds that the only constraint satisfied by all the states in the example behaviour is:

$$LevB = M^{-}(LevA)$$

This model is under constrained. It allows for example an obviously impossible behaviour when LevA becomes (zero,std) and at the same time LevB is (pos,std). The GENMODEL algorithm cannot find a more specific model (that is one with more constraints) because such a model requires the introduction of new variables. Therefore a more general algorithm has to reconstruct also the "hidden" variables, for example the flow in our case. Say and Kuru's (1996) QSI algorithm introduces new variables as follows. It hypothesises the existence of a new variable, and constructs a possible qualitative constraint between this variable and existing variables. Such a constraint qualitatively defines the new variable. So QSI can in this U-tube example introduce a new variable, X, by constructing the constraint deriv(LevB,X). QSI executes the GENMODEL algorithm iteratively. In the second iteration, when X has been introduced, QSI will find three satisfied constraints:

$$LevB = M^{-}(LevA),\ add(LevB, X, LevA),\ deriv(LevB, X)$$

In this way QSI discovers the hidden variable X that precisely corresponds to the flow of water. This model only allows the given observed behaviour, so QSI stops here and outputs this as the final result.

Generally, QSI iteratively introduces new variables, which enables the addition of further constraints to the model. Each successive model is therefore more specific, that is, it allows only a subset of behaviours of the more general models. In successive iterations, the "depth" of model also increases. The *depth of a model* is defined as the maximal depth of a variable in the model. The *depth of a variable* is determined by the way the variable was introduced. The observed variables have depth zero. A new variable is introduced with a constraint in which the new variable appears together with at least one old variable. The depth of the new variable is one plus the depth of the old variable. These iterations stop when the model is "sufficiently specific". A model is accepted as sufficiently specific when it only allows an

acceptable number of behaviours. The user of QSI has to specify an acceptable degree of behaviour branching allowed by a model, which is related to the generality of the model. In this way QSI rather nicely determines an appropriate number of new variables, and thus achieves an appropriate generality of the model.

In essence, the problem of defining just the "right degree of generality" of a model arises always when models are learned from positive only examples. This is the case in both GENMODEL and QSI, as well as in several other systems including MISQ (Richards et al. 1992) and QOPH (Coghill et al. 2002). Since there are no negative examples given, the completely unconstrained model (empty model, with no constraints) is consistent with the learning data. Such a model, although consistent with the observations, is of course useless. Therefore such models should be avoided by an appropriate learning bias, which should prevent useless, although consistent hypotheses. All the above mentioned systems are biased toward selecting a most specific model. GENMODEL simply selects the most specific model constructed from the given types of constraints and the observed variables. This is where GENMODEL stops, because it does not introduce new variables. However, it is not always possible to construct a sufficiently specific model just using the observed variables. MISQ (Richards et al. 1992) is similar to GENMODEL, but it introduces new variables aiming at a connected model; that is a model in which all the observed variables are connected by chains of constraints in which new variables may appear. The connectedness requirement is of course merely a heuristic that may not result in the intended model. The QSI system controls the introduction of new variables in a more general way that results in a more sophisticated learning bias: QSI constructs the most specific model using the observed variables and all the new variables up to the maximal depth. The depth is determined by the acceptable branching of the model. The model has to be sufficiently deep to prevent excessive branching. In this respect QSI makes a kind of an implicit closed world assumption, although this assumption is only enforced "softly". Namely, the QSI algorithm treats the states in the given example behaviours as positive examples, and the siblings of these states as potential negative examples.

The learning from positive only examples in this context is often considered as an advantage of a learning system because the observations are supposed to come from nature which only provides positive examples. We cannot observe in nature things that are not possible. However, this advantage of learning from positive only examples is not so clear. As

mentioned above, QSI makes a kind of closed world assumption by which it considers some things that were not observed, effectively as negative examples. Also, to compensate for the lack of negative examples, these algorithms adopt the bias toward the most specific models, which may also be debatable. On the other hand, the user (e.g. an expert) may well be able to specify negative examples on the basis of the background knowledge and the general understanding of the problem domain. So the restriction to the learning from positive *only* examples does not seem to be really necessary in practice.

Let us further discuss the situation regarding the availability of negative examples. Since nature can only provide positive examples, negative examples have to come from some other source, most naturally from a domain expert. It is sometimes considered that it is not realistic to expect that the domain expert be capable of providing negative examples, unless the expert already knows the target model completely. But then, if the model is already known, there would be no point in learning a model from data. We believe that such a view is mistaken, because it assumes that the expert either knows the right model *completely*, or has *no idea at all*. However, model building in practice is usually between these two extremes. The expert normally does have some ideas about the domain (often referred to as "background knowledge"), but these are insufficient to immediately put together a completely correct model. The incomplete expert's knowledge can often be expressed in terms of negative examples: the expert just states what he or she believes can surely not happen. For example, in the case of modelling a U-tube, the modeller may not be able to define a complete and correct model. Still, he may be easily capable of stating, by means of negative examples, that the amount of water in a container cannot be negative, and that the total amount of water in the whole system is constant. Such use of incomplete knowledge through negative examples can also be illustrated by program writing, a task similar to modelling. Building models, executable through simulation, actually is a kind of program writing where a "program" is interpreted by the simulation software. Consider a programmer writing a program to sort lists. Although the programmer may not be quite capable of writing down a completely correct program, he may still be able to confidently provide negative examples: the result of sorting the list [3, 1, 2] is not [2, 1, 3], nor is it [1, 2].

In addition to negative examples, the expert may also be able to specify some specific background knowledge that may be useful in the learning of qualitative models in the particular domain. Inductive Logic Programming (ILP) is the machine learning framework

that ideally suits this situation. The following is the ILP problem formulation that applies to the learning of qualitative models from background knowledge BK, QDE constraints QC, and sets POS and NEG of positive and negative examples respectively. Given BK, QC, POS and NEG, find a model M such that:

For each example P in POS: BK & QC & M |-- P
and for each example N in NEG: BK & QC & M |-\- N

Once the problem is so formulated in logic, a general purpose ILP learning program can be applied. Bratko et al. (1991) used such an ILP system GOLEM in an experiment to induce a model of a U-tube from positive and negative examples. Although this exercise was impeded by some technical limitations of GOLEM, it showed the advantages of using ILP: (a) it was not necessary to develop a special purpose learning program for QDE learning, and (b) since GOLEM introduces new variables itself, there was no special care needed in respect of hidden variables. Another advantage that comes automatically with ILP is the learning of models with multiple operating regions. General purpose ILP programs generate multiple clause logic programs, so each clause may cover one operating region. The QOPH system (Coghill et al. 2002) also relies on using such a general ILP program called Aleph (Srinivasan 2000). QOPH does remarkably well with introducing new variables, although this relies on a number of heuristics that need further study.

The programs mentioned above learn qualitative models from given examples of qualitative behaviours. In an actual application, it is more likely, however, that the observed data are numerical. Most of these programs require a transformation of such numerical data into qualitative behaviours. Some of the above discussed approaches (Say and Kuru, 1996; Coghill et al. 2002) also include such a transformation. It is not easy to do this well, specially when there is noise in the numerical data. Džeroski's and Todorovski's (1995) QMN program is interesting in that it builds QDE models directly from numerical data, avoiding such a numerical-to-qualitative transformation. Program SQUID (Kay et al. 2000) is also relevant in this respect. It learns "semi-quantitative" models (a combination of QDEs with numerical elements) from numerical data. Another idea to handle numerical data is to apply QUIN in combination with QDE learning.

The QDE learning programs reviewed above were usually tested on small experimental domains, and rarely on problems of realistic complexity. Probably the most impressive application on real-life data is described by Hau and Coiera (1997). Their system transforms signals measured in time into qualitative behaviours which are input into GENMODEL. They applied this system to actually measured cardio vascular signals from a number of patients and induced, from these data, qualitative models characterising individual patients. Another ambitious application oriented work is Mozetič's program QuMAS (Bratko et al., chapter 5) which learned models of the electrical system of the heart capable of explaining many types of cardiac arrhythmias. QuMAS did not use QDE constraints as modelling primitives, but a set of problem-specific logical descriptions used by what would now be recognised as an ILP learning system.

Conclusions

In this paper two paradigms of learning qualitative models were reviewed: the learning of qualitative trees, and the learning of QDE models. Learning qualitative models in comparison with traditional, numerical approaches to automated modelling or system identification, should in general have the following advantages: (a) better comprehensibility, thus providing better insight into the mechanisms in the domain of investigation; (b) qualitative learning may be possible when numerical approaches are not, because qualitative models are at a higher abstraction level and should require less data to learn because just qualitative relations are to be determined, and not precise quantitative dependences.

The learning of qualitative trees (in combination with Q2Q transformation) has been shown in a number of case studies to be competitive with traditional quantitative methods even in terms of criteria usually intended for quantitative methods. In addition to offering a comprehensible model, qualitative trees define a space for numerical optimisation among qualitatively equivalent solutions. A deficiency of qualitative trees might be that they explicitly define static relations only. For some applications, an extension of this approach towards explicit treatment of time would be useful. This would better support explanation of phenomena that progress in time.

A number of systems exist for learning of QDE models. Their development demonstrates improvements in respect of several rather intricate problems involved in the learning of QDE

models. It seems that further progress is still required in several respects. These include: better methods for transforming numerical data into qualitative data, targeted explicitly towards the particular qualitative modelling language; deeper study of principles or heuristics associated with the discovery of hidden variables, the generality and the size of models; more effective use of general ILP techniques.

Automating qualitative modelling is generally regarded to be very important for the application of qualitative modelling techniques. There are some very encouraging experimental applications of learning qualitative models from data. One example is the induction of patient-specific models from the patients' measured cardio vascular signals (Hau and Coiera 1997). In another example, the QUIN program was recently experimentally applied within the Clockwork project to an industrial task in a way that was not thought of before. The German car simulation company INTEC wanted to simplify their car wheel suspension model for efficiency reasons. A qualitative model of the suspension system was obtained with QUIN from simulation data. This qualitative model was then transformed into a simplified numerical model through the Q2Q transformation. In this way a simplified numerical model was obtained that has the same qualitative behaviour as the original model, but is computationally much more efficient. In this experiment, preserving qualitative fidelity proved to be useful with respect to numerical accuracy of the so-obtained simplified model. It was not possible to obtain similar numerical accuracy with standard numerical learning techniques that pay no attention to qualitative properties.

In general, however, the impact of qualitative model learning techniques on the practice of qualitative modelling has been rather slow. This is particularly surprising in the view of enormous increase in the past decade of machine learning applications in other areas (Michalski et al. 1998). There are several probable reasons for this slow impact of machine learning on the practice of qualitative modelling:
(1) Relatively weak awareness of the existing work in the learning of qualitative models.
(2) Hypothesis languages used in learning qualitative models (such as QDEs) are relatively complicated; therefore the use of machine learning in qualitative modelling is comparatively more demanding for the user than in other typical applications of machine learning.
(3) The use of existing methods for learning qualitative models is impeded by lack of technical maturity (robustness, scalability, predictability).

Improvements of methods for learning qualitative models along the lines mentioned above would alleviate these difficulties.

Acknowledgements

This work was partially supported by the Slovenian Ministry of Science, Education and Sport, and the European Commission (Clockwork project). We thank M. Valašek and Z. Šika of the Czech Technical University for assistance with the "anti-sway" controller. We also thank the editors B. Bredeweg and P. Struss of this AI Magazine special issue, and the anonymous referees for useful comments.

References

Bratko, I., Mozetič, I., Lavrač, N.. (1989) *KARDIO: A Study in Deep and Qualitative Knowledge for Expert Systems*. Cambridge, MA: MIT Press.

Bratko, I., Muggleton, S., Varšek, A. (1991) Learning qualitative models of dynamic systems. *Proc. Inductive Logic Programming ILP-91*, Viana do Castelo, Portugal, pp. 207-224. Also in: *Inductive Logic Programming* (ed. S. Muggleton) Academic Press 1992, pp.. 437-452.

Bratko, I., Urbančič, T. (1999) Control skill, machine learning and hand-crafting in controller design. In: *Machine Intelligence 15: Intelligent Agents* (eds. K. Furukawa, D. Michie, S. Muggleton), pp. 130-163, Oxford University Press, 1999

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984) *Classification and Regression Trees.* Monterey, CA: Wadsford.

Coiera, E. (1989) Generating qualitative models from example behaviours. DCS Report 8901, Dept. of Computer Sc., Univ. of New South Wales, Sydney, Australia.

G. M. Coghill, S. M. Garrett and R. D. King (2002) Learning Qualitative Models in the Presence of Noise, *QR'02 Workshop on Qualitative Reasoning*, Sitges, Spain 2002.

Džeroski, S., Todorovski, L. (1995) Discovering dynamics: from inductive logic programming to machine discovery. *J. Intell. Information Syst*., 4:89-108.

Hau, D.T., Coiera, E.W. (1997) Learning qualitative models of dynamic systems. *Machine Learning*, 26: 177-211.

Kuipers, B. (1986) Qualitative simulation. *Artificial Intelligence*, 29: 289-338.

Kuipers, B. (1994) *Qualitative Reasoning*. Cambridge, MA: MIT Press.

Kay, H., Rinner, B., Kuipers, B. (2000) Semi-quantitative system identification. *Artificial Intelligence*, 119: 103-140.

Michalski, R.S., Bratko, I., Kubat, M. (1998, eds.) *Machine Learning and Data Mining: Methods and Applications*. Wiley.

Michie, D. (1993) Knowledge, learning and machine intelligence. In: *Intelligent Systems* (ed. L. Sterling), pp. 2-19. New York: Plenum Press.

Michie, D., Bain, M., Hayes-Michie, J. (1990) Cognitive models from subcognitive skills. In: *Knowledge-Based Systems in Industrial Control* (eds. M. Grimble, J. McGhee, P. Mowforth), pp. 71-99. Peter Peregrinus.

Quinlan, J.R. (1992) Learning with continuous classes. *Proc. Fifth Australian Joint Conf. on Artificial Intelligence*, Hobart, Tasmania. World Scientific, pp. 343-348.

Quinlan, J.R. (1993) *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Richards, B.I., Kraan, I., and Kuipers, B.J. (1992) Automatic abduction of qualitative models. *Proc. Tenth National Conf. Artificial Intelligence*, 723-728.

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

Srinivasan, A. (2000) Aleph web site: web.comlab.ox.ac.uk/oucl/research/areas/mach-learn/Aleph/aleph_toc.html

Šuc, D. (2001) *Machine Reconstruction of Human Control Strategies*. Ph. D. Thesis, Univ. of Ljubljana, Faculty of Computer and Information Sc.

Šuc, D., Bratko, I. (2000a) Skill modelling through symbolic reconstruction of operator's trajectories. *IEEE Trans. on Systems, Man and Cybernetics*, Part A, 30: 617-624

Šuc, D., Bratko, I. (2000b) Qualitative trees applied to bicycle riding. *ETAI Journal* (*Electronic Transactions on Artificial Intelligence*), Vol. 4 (2000), Section B, pp. 125-140. http://www.ep.liu.se/ej/etai/2000/014/.

Šuc, D., Bratko, I. (2001) Induction of qualitative trees. *Proc. ECML'01 (European Conf. Machine Learning)*, Freiburg, Germany. Springer-Verlag, LNAI Series.

Šuc, D., Bratko, I. (2002) Qualitative reverse engineering. *Proc. ICML'2002 (Int. Conf. Machine Learning)*, Sydney, Australia, July 2002

Urbančič, T., Bratko, I. (1994) Reconstructing human subcognitive skill through Machine Learning. *Proc. ECAI-94 (European Conf. AI)*, Amsterdam, 1994, 498-502.

Valašek, M. *et al*.: Position and velocity control of gantry crane. *Proc. of Mechatronic 96*, Guimaraes 1996, pp. I-203 - I-208

Wellman, M. P. (1991) Qualitative simulation with multivariate constraints. *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning* (J. Allen, R. Fikes and E. Sandewall, eds.) San Mateo, CA: Morgan Kaufmann.