Learning Qualitative Models

Ivan Bratko and Dorian Šuc

■ In general, modeling 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 behaviors of a modeled system are used as examples for a learning algorithm that constructs a model that is consistent with the data. In this article, we review approaches to learning qualitative models, either from numeric data or qualitative observations. We describe the QUIN program that looks for qualitative patterns in numeric data and outputs the results of learning as "qualitative trees." We illustrate this using applications associated with systems control, in particular, the identification and optimization of controllers and human operator's control skill. We also review approaches that learn models in terms of qualitative differential equations.

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 behaviors from given qualitative models. However, an important practical question is how do we construct qualitative models in the first place. In general, model construction is usually the most demanding aspect of the modeling task. In this article, we look at research that aims at automating this task.

One idea is to use observations of the modeled system, obtained through measurements, and try to find a model that, when simulated, would reproduce the same, observed behaviors. 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 also making predictions in situations other than those literally included among the observations. This task can also be viewed as machine learning from examples. The observed system behaviors 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 article, we consider the particular problem of inducing a qualitative model from examples of system behaviors. 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 the identification of tacit control skills of human operators. In the last part of the article, we review some other representative approaches to the learning of qualitative models.

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

Qualitative Data Mining with QUIN

Qualitative induction (QUIN) is a learning program that looks for qualitative patterns in numeric data (Šuc 2001; Šuc and Bratko 2001). Induction of the so-called qualitative trees is similar to the well-known induction of decision trees (for example, CART [Breiman et al. 1984], C4.5 [Quinlan 1993]). The difference is that in decision trees, the leaves are labeled with class values, whereas in qualitative trees,

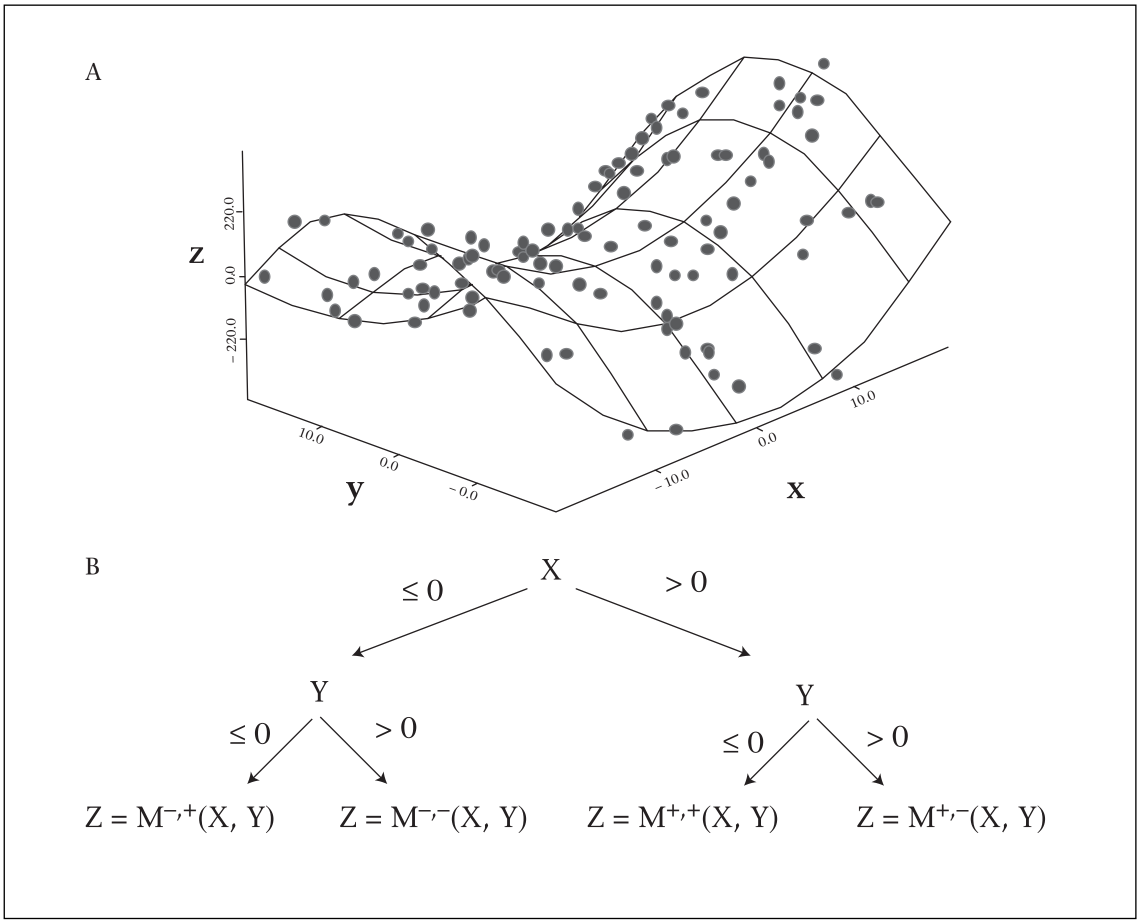

Figure 1. An Illustration of QUIN *Use.*

A. A data set of points where Z = $X^2 - Y^2$ plus some noise. B. A qualitative tree induced by QUIN from this data set.

the leaves are labeled with what we call *qualitatively constrained functions* (QCFs).

QCFs are a kind of monotonicity constraint that is widely used in the field of qualitative reasoning. A simple example of a QCF is $Y = M^{+}(X)$, which 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 can 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 also appear under the term *multivariate monotonic function constraints* (Wellman 1991).

QUIN takes as input a set of numeric 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 is 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 this 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, similar 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 *minimum description length* (MDL) *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 can be found in Šuc (2001) and Šuc and Bratko (2001), where QUIN's performance on noisy data is also studied.

A Qualitative Reverse Engineering Application of QUIN

In this section, we present an application of QUIN to reverse engineering of an artifact. 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 application, 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. Reuse of designs is made difficult, especially because the intuitions behind designs and their improvements are not explained in the documentation. Although there might 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 the designed system works. 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

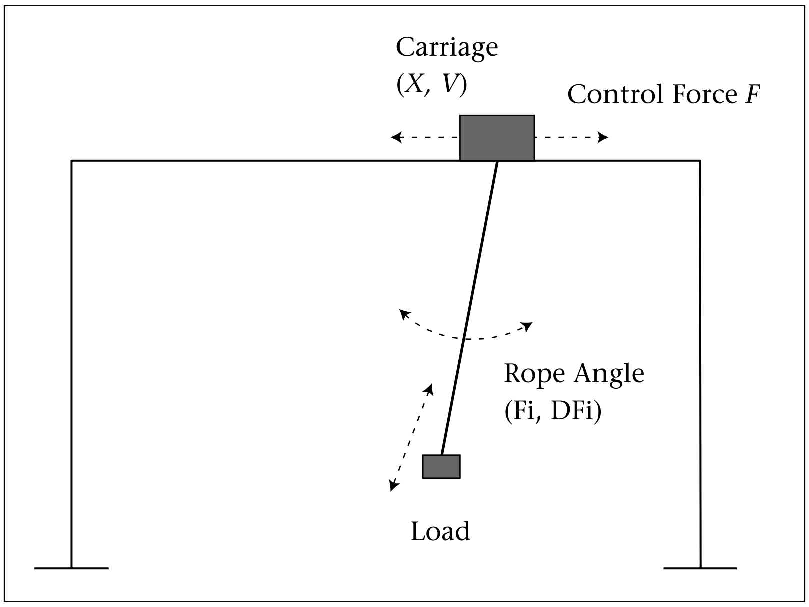

Figure 2. Gantry Crane.

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 analyze the behavior 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 might be completely different.

We illustrate an approach to qualitative reverse engineering with 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 a large swing of the load. The controller should carry out acceleration maneuvers in such a way as to minimize the swing; so, it should be capable of controlling the swing under large accelerations. This aspect is the most difficult of the crane control task.

Valasek designed an antisway industrial controller for gantry cranes (Valasek et al. 1996). This controller is now in everyday use in cranes in a Czech metallurgical factory.[1] The controller helps the crane operator to easily control the crane carriage velocity without causing a large swing of the load. The operator specifies

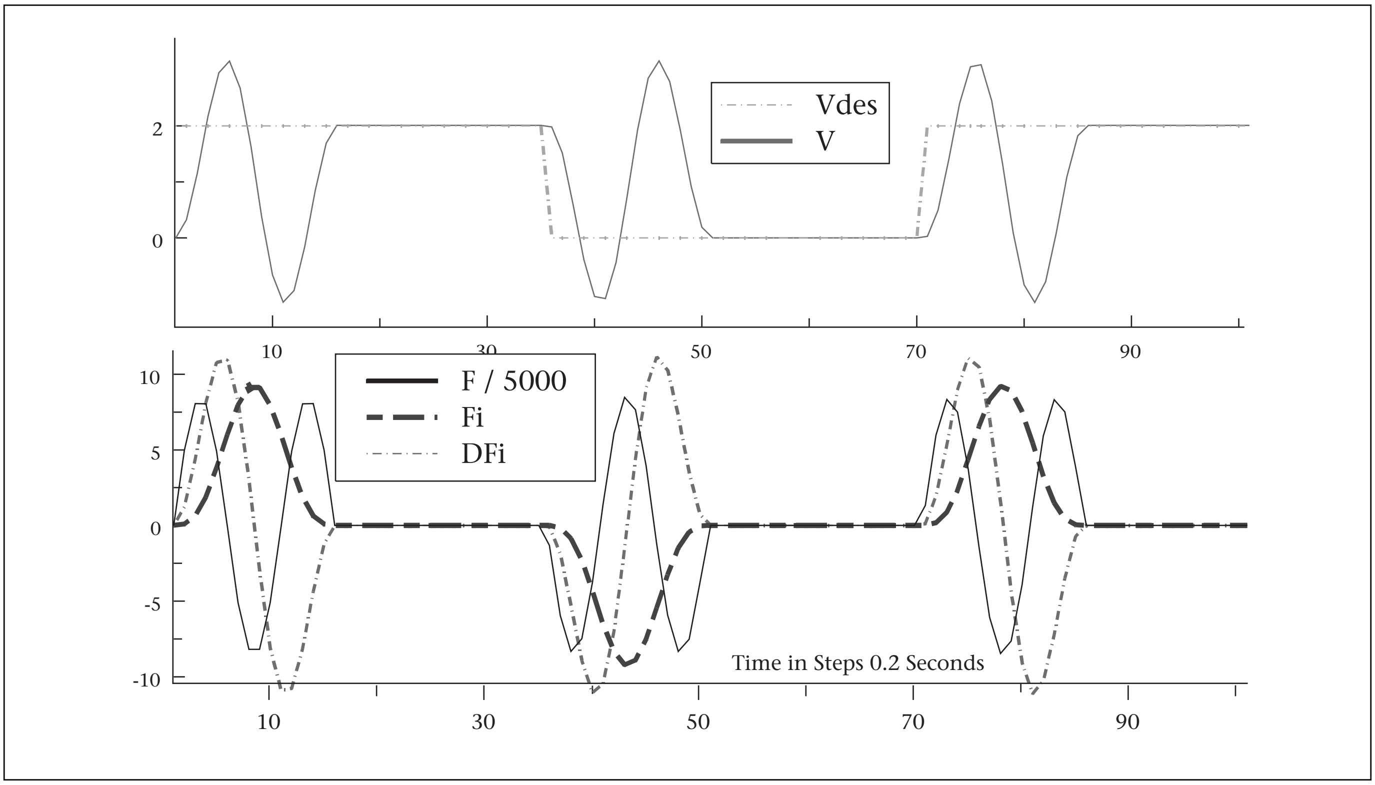

Figure 3. Execution Trace in Time of the Antisway Controller.

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 the desired velocity, but it does so in such a way that it only causes a small swing of the load. Therefore, this controller is also called the *antisway crane*. Figure 3 gives an example execution trace of the desired velocity *Vdes*, the controller's action (force *F* in time), the actual crane's velocity *V*, the rope angle *Fi,* and the 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 nontrivial way and so does the actual velocity.

Now consider that this controller is given as a black box. Its input and output 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 article, we are particularly interested in extracting from control traces 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 because such models usually provide a 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 behaviors of *Vdes*, *F, X, V, Fi,* and *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, which 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 might be, The greater the rope angle and the faster it increases, the greater the carriage velocity to "catch up" with the angle should be. This rule can be stated by the following 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

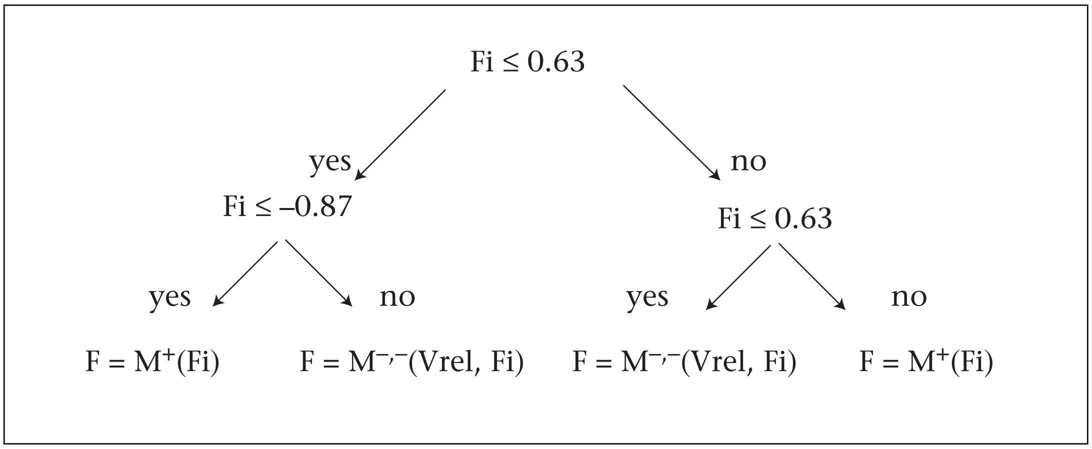

Figure 4. A Qualitative Controller Induced from the Trace of Figure 3.

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).

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*.

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

A qualitative control strategy, such as that in figure 4, cannot directly be used as a controller because it does not determine a precise, numeric 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 numeric 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) and Šuc and Bratko (2000a). Experiments described in Šuc and Bratko (2002) with this optimization procedure applied on qualitative control strategies for the antisway crane show that the obtained reconstructed controllers perform comparably to the original controller.

For comparison, one might consider whether it was essential to first reconstruct the control strategy from quantitative data qualitatively and then transform it into a quantitative strategy, or whether a straightforward numeric reconstruction, using numeric regression, would be equally successful? The most natural machine learning method for this numeric reconstruction is the 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 antisway controller. It is not quite clear yet why the straightforward numeric learning failed when qualitative learning (combined with the Q2Q transformation) succeeded. Just studying the learning data from the execution trace suggests that the numeric data are rather regression unfriendly, and it is hard for the regression procedure to decide whether some relatively small numeric differences are just the result of noise or numeric perturbations or simply reflect genuine dependences among the variables. However, it seems that QUIN is more robust at detecting subtle qualitative dependences because it pays less attention to the absolute values of numeric changes. In other words, what numerically looks like noise might 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

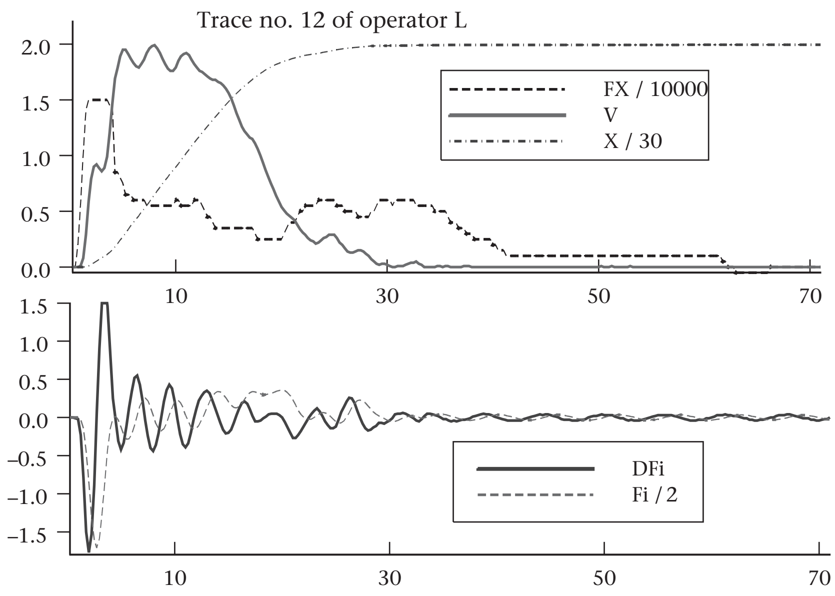

Figure 5. Execution Trace in Time of Operator L.

we have a skilled crane operator. Some questions of interest are as follows: How does he/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, where the operator would be expected to describe his/her own skill. This description would then be appropriately formalized and built into an automatic controller.

The problem with this approach is that the skill is subcognitive, 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) and 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 with machine learning techniques, which is also known as *behavioral cloning* (Michie 1993; Michie, Bain, and Hayes-Michie 1990).

One goal of behavioral 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 style similar to the operator, actually performs better and more consistently than the operator.

Performance improvement is, however, not the only goal of behavior cloning. Another important goal is to generate meaningful clones to help us understand the operator's skill. Understanding what exactly a human operator is doing and why can be of practical importance. We can 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, subgoals, plans, feedback loops, causal relations between actions and state conditions, and so on. These conditions are to be stated in terms of information that is easily accessible to the operator, for example, visually. It can be argued that such information should be largely qualitative as opposed to numeric. In the next section, we show how qualitative descriptions of control skills can be induced by QUIN from operator's control traces.

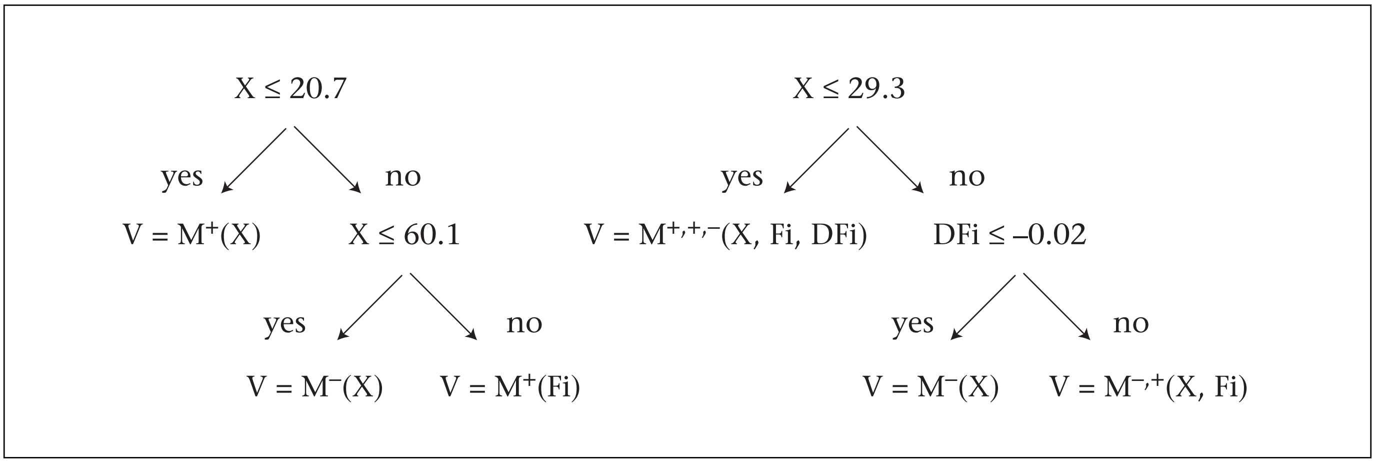

Figure 6. Qualitative Strategy of Operators S and L.

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.*

Looking for Qualitative Patterns in Dynamic Behaviors

Figure 5 shows an execution trace of a human operator controlling a simulated crane. This human control was one of the most successful (in terms of time to complete the task) 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/she was capable of controlling the swing and reducing it when approaching the goal position.

Formally, the task of behavioral 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. Thus, formally, find two functions

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

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

where *X, V,* and so on, 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 numeric values of the two variables *X* and *V*. However, 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 a long time to complete the task of transferring the load from start to goal. In contrast, operator *L* is more adventurous and does not dodge large accelerations, causing large swing, but operator *L* can afford this large swing because he/she is capable of confidently reducing the swing when necessary. Thus, *L* is able to achieve much shorter completion times than *S*.

Figure 6 nicely exposes the differences in the control styles of both operators. Although their corresponding qualitative trees have a 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 leftmost 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. Increasing velocity is expressed by the constraint in the leftmost 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. Paying attention to the angle can be seen in the rightmost leaf of S's tree that 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 is swinging to the right, accelerate the carriage to "follow" the load and, thus, reduce the angle. Operator L, however, considers the angle and angular velocity that is already at the early stage: The leftmost leaf of L's tree says that the carriage velocity should also depend on the rope angle Fi and angular velocity DFi as well as on X.

As in the case of the antisway crane, "qualitative clones" cannot be applied directly to the control of the crane. Again, we need a transformation of the QCFs into real-valued functions. This time, the natural optimization 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 and Bratko 2000b) 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

To this point, we have discussed the learning of qualitative models represented as qualitative trees. In this section, we 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 behavior. In the remainder of this article, we 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 1994, 1986), 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' numeric 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 nondetermined; so, if X = (pos, inc) and Y = (neg, dec), then Z can be (pos, inc), or (zero, inc), or (pos, std), or (neg, inc), and so on.

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 behaviors is, Given qualitative behaviors of a set of observed variables, find a QDE model, that is, a set of qualitative constraints that are consistent with the given behaviors.

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

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

Second, evaluate all the constraints constructed in step 1 on all the qualitative system's states in the given example behaviors. 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. Thus, the three components form a U-shape. Suppose that initially there is some water in container *A,* but container *B* is empty. The difference between the two water levels, Lev*A* and Lev*B,* causes a positive flow from *A* to *B*. Thus, Lev*A* will be decreasing and Lev*B* increasing until both levels become equal, and the flow becomes zero. This behavior can be stated qualitatively as a sequence of three qualitative states of the three variables (table 1).

Now assume that this behavior 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:

Lev*B* = M^-*(*Lev*A), add(*Lev*B, Flow,* Lev*A),* deriv*(*Lev*B, 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:

First, all the variables in the target model are observed, so they explicitly appear in the example behaviors. 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 behaviors). GENMODEL does not handle hidden variables.

Second, 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.

Third, the resulting model is assumed to apply to the complete state space of the dynamic system, which is not appropriate for the cases when the system can be in more than one operating region. For example, consider the 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, Lev*A* and Lev*B,* are observed only. The GENMODEL algorithm finds that the only constraint satisfied by all the states in the example behavior is

Lev*B* = M^-*(*Lev*A)*

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

Table 1. Qualitative Behavior of the U-Tube.

This model is underconstrained. It allows, for example, an obviously impossible behavior when Lev*A* becomes (zero, std), and at the same time, Lev*B* 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 also has to reconstruct the "hidden" variables, for example, the flow in our case. Say and Kuru's (1996) QSI algorithm introduces new variables as follows. It hypothesizes 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. Thus, in this U-tube example, QSI can introduce a new variable, *X,* by constructing the constraint deriv(Lev*B, X*). QSI executes the GENMODEL algorithm iteratively. In the second iteration, when *X* has been introduced, QSI will find three satisfied constraints:

Lev*B* = M^-(Lev*A*), add(Lev*B, X,* Lev*A*), deriv(Lev*B, 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 behavior, so QSI stops here and outputs this model as the final result.

Generally, QSI iteratively introduces new variables, which enable the addition of further constraints to the model. Each successive model is, therefore, more specific; that is, it allows only a subset of behaviors of the more general models. In successive iterations, the *depth* of the 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 behaviors. The user of QSI has to specify an acceptable degree of behavior 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 always arises 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, Kraan, and Kuipers 1992) and QOPH (Coghill, Garrett, and King 2002). Because 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 these 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, but 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, Kraan, and Kuipers 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 might appear. The connectedness requirement is, of course, merely a heuristic that might 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 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 implicit closed-world assumption, although this assumption is only enforced *softly;* that is, the QSI algorithm treats the states in the given example behaviors 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 as clear. As mentioned earlier, 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 might also be debatable. However, the user (for example, an expert) might well be able to specify negative examples on the basis of the background knowledge and the general understanding of the problem domain. Thus, the restriction to 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. Because 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 unrealistic to expect that the domain expert be capable of providing negative examples, unless the expert already knows the target model completely. However 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 typically 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/she believes can surely not happen. For example, in the case of modeling a U-tube, the modeler might not be able to define a complete and correct model. Still, he/she might easily be 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 that is similar to modeling. Building models, executable through simulation, is actually 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 might not quite be capable of writing down a completely correct program, he/she might 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 might also be able to specify some specific background knowledge that might 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, $BK \;\&\; QC \;\&\; M \vdash P$ and for each example *N* in NEG: $BK \;\&\; QC \;\&\; M \nvdash N$

Once the problem is so formulated in logic, a general-purpose ILP learning program can be applied. Bratko, Muggleton, and Varsek (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: (1) it was not necessary to develop a special-purpose learning program for QDE learning and (2) because 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 can cover one operating region. The QOPH system (Coghill, Garrett, and King 2002) also relies on using such a general ILP program called ALEPH.[2] QOPH does remarkably well with introducing new variables, although this relies on a number of heuristics that need further study.

The programs mentioned earlier learn qualitative models from given examples of qualitative behaviors. In an actual application, it is more likely, however, that the observed data are numeric. Most of these programs require a transformation of such numeric data into qualitative behaviors. Some of the discussed approaches (Coghill, Garrett, and King 2002; Say and Kuru 1996) also include such a transformation. It is not easy to do this well, especially when there is noise in the numeric data. Dzeroski's and Todorovski's (1995) QMN program is interesting in that it builds QDE models directly from numeric data, avoiding such a numeric-to-qualitative transformation. The program SQUID (Kay, Rinner, and Kuipers 2000) is also relevant in this respect. It learns *semi-quantitative models* (a combination of QDEs with numeric elements) from numeric data. Another way of handling numeric data is to apply QUIN in combination with QDE learning.

The QDE learning programs reviewed earlier were usually tested on small experimental domains and rarely on problems of realistic complexity. Probably the most impressive application to real-life data is described by Hau and Coiera (1997). Their system transforms signals measured in time into qualitative behaviors that are input into GENMODEL. They applied this system to actually measured cardiovascular signals from a number of patients and induced from these data qualitative models characterizing individual patients. Another ambitious application-oriented work is Mozetic's QUMAS (Bratko, Muggleton, and Varsek 1991), 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 modeling primitives but, rather, a set of problem-specific logical descriptions used by what would now be recognized as an ILP learning system.

Conclusion

In this article, two paradigms of learning qualitative models were reviewed: (1) the learning of qualitative trees and (2) the learning of QDE models. Learning qualitative models in comparison with traditional, numeric approaches to automated modeling or system identification should, in general, have the following advantages: (1) better comprehensibility, thus providing better insight into the mechanisms in the domain of investigation, and (2) a higher abstraction level, thus requiring less data to learn because only qualitative relations are to be determined, not precise quantitative dependencies.

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 numeric optimization 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 toward explicit treatment of time would be useful. Explicit treatment of time would better support explanation of phenomena that progress in time.

A number of systems exist for learning QDE models. Their development demonstrates improvements to several rather intricate problems involved in the learning of QDE models. It seems that further progress is still required in several respects, including better methods for transforming numeric data into qualitative data, targeted explicitly toward the particular qualitative modeling language; deeper study of principles or heuristics associated with the discovery of hidden variables and the generality and size of models; and more effective use of general ILP techniques.

Automating qualitative modeling is generally regarded as important for the application of qualitative modeling 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 cardiovascular signals (Hau and Coiera 1997). In another example, the QUIN program was recently applied to an industrial task in a unique way within the CLOCKWORK project. 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 from simulation data with QUIN. This qualitative model was then transformed into a simplified numeric model through the Q2Q transformation. In this way, a simplified numeric model was obtained that has the same qualitative behavior as the original model but is computationally much more efficient. In this experiment, preserving qualitative fidelity proved to be useful with respect to numeric accuracy of the simplified model. It was not possible to obtain similar numeric accuracy with standard numeric learning techniques that pay no attention to qualitative properties.

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

Improvements in methods for learning qualitative models along the lines mentioned earlier would alleviate these difficulties.

Acknowledgments

This work was partially supported by the Slovenian Ministry of Science, Education, and Sport and the European Commission (CLOCKWORK project). We want to thank M. Valasek and Z. Sika of the Czech Technical University for assistance with the antisway controller. We also want to thank the editors of this *AI Magazine* special issue, B. Bredeweg and P. Struss, and the anonymous referees for useful comments.

Notes

1. M. Valašek, personal communication, 2002.

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

References

Bratko, I., and Urbančič, T. 1999. Control Skill, Machine Learning, and Hand-Crafting in Controller Design. In *Machine Intelligence 15: Intelligent Agents*, eds. K. Furukawa, D. Michie, and S. Muggleton, 130–163. Oxford, U.K.: Oxford University Press.

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

Bratko, I.; Muggleton, S.; and Varšek, A. 1992. Learning Qualitative Models of Dynamic Systems. In *Inductive Logic Programming,* ed. S. Muggleton, 437–452. San Diego, Calif.: Academic.

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

Coghill, G. M.; Garrett, S. M.; and King, R. D. 2002. Learning Qualitative Models in the Presence of Noise. Paper presented at the QR'02 Workshop on Qualitative Reasoning, 10–12 June, Sitges, Spain.

Coiera, E. 1989. Generating Qualitative Models from Example Behaviors. DCS Report, 8901, Department of Computer Science, University of New South Wales.

Dzeroski, S., and Todorovski, L. 1995. Discovering Dynamics: From Inductive Logic Programming to Machine Discovery. *Journal of Intelligent Information Systems* 4(1): 89–108.

Hau, D. T., and Coiera, E. W. 1997. Learning Qualitative Models of Dynamic Systems. *Machine Learning* 26(2–3): 177–211.

Kay, H.; Rinner, B.; and Kuipers, B. 2000. Semi-Quantitative System Identification. *Artificial Intelligence* 119(1–2): 103–140.

Kuipers, B. 1994. *Qualitative Reasoning.* Cambridge, Mass.: MIT Press.

Kuipers, B. 1986. Qualitative Simulation. *Artificial Intelligence* 29(3): 289–338.

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

Michie, D. 1993. Knowledge, Learning, and Machine Intelligence. In *Intelligent Systems,* ed. L. Sterling, 2–19. New York: Plenum.

Michie, D.; Bain, M.; and Hayes-Michie, J. 1990. Cognitive Models from Subcognitive Skills. In *Knowledge-Based Systems in Industrial Control,* eds. M. Grimble, J. McGhee, and P. Mowforth, 71–99. London: Peter Peregrinus.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning.* San Francisco, Calif.: Morgan Kaufmann.

Quinlan, J. R. 1992. Learning with Continuous Classes. Paper presented at the Fifth Australian Joint Conference on Artificial Intelligence, 16–18 November, Hobart, Tasmania.

Richards, B. I.; Kraan, I.; and Kuipers, B. J. 1992. Automatic Abduction of Qualitative Models. In Proceedings of the Tenth National Conference on Artificial Intelligence, 723–728. Menlo Park, Calif.: American Association of Artificial Intelligence.

Say, A. C. C., and Kuru, S. 1996. Qualitative System Identification: Deriving Structure from Behavior. *Artificial Intelligence* 83(1): 75–141.

Šuc, D. 2001. Machine Reconstruction of Human Control Strategies. Ph.D. dissertation, Faculty of Computer and Information Science, University of Ljubljana.

Šuc, D., and Bratko, I. 2002. Qualitative Reverse Engineering. Paper presented at the International Conference on Machine Learning (ICML'2002), 8–12 July, Sydney, Australia.

Šuc, D., and Bratko, I. 2001. Induction of Qualitative Trees. In *Proceedings of the European Conference on Machine Learning (ECML'01),* 442–453. Lecture Notes in Artificial Intelligence 2167. Berlin: Springer-Verlag.

Šuc, D., and Bratko, I. 2000a. Qualitative Trees Applied to Bicycle Riding. *Electronic Transactions on Artificial Intelligence* 5(4): 125–140.

Šuc, D., and Bratko, I. 2000b. Skill Modeling through Symbolic Reconstruction of Operator's Trajectories. *IEEE Transactions on Systems, Man, and Cybernetics* 30(6): 61–624.

Urbančič, T., and Bratko, I. 1994. Reconstructing Human Subcognitive Skill through Machine Learning. Paper presented at the European Conference on Artificial Intelligence, 8–12 August, Amsterdam, The Netherlands.

Valašek, M.; Milacek, S.; Dedouch, K.; Kozanek, Y.; and Zolotarev, I. 1996. Position and Velocity Control of Gantry Crane. Paper presented at Mechatronic 96, 18–20 September, Guimares, Portugal.

Wellman, M. P. 1991. Qualitative Simulation with Multivariate Constraints. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning,* eds. J. Allen, R. Fikes, and E. Sandewall. San Francisco, Calif.: Morgan Kaufmann.

Ivan Bratko is a professor of computer science in the Faculty of Computer and Information Science, Ljubljana University, Slovenia. He heads the AI laboratory at the university and has conducted research in machine learning, knowledge-based systems, qualitative modeling, intelligent robotics, heuristic programming, and computer chess. His main interests in machine learning have been in learning from noisy data; the combining of learning and qualitative reasoning; and various applications of machine learning and inductive logic programming, including medicine, ecological modeling, and control of dynamic systems. Bratko is the author of *Prolog Programming for Artificial Intelligence,* 3d. ed. (Addison-Wesley, 2001). He coedited (with R. S. Michalsky and M. Kubat) *Machine Learning and Data Mining: Methods and Applications (*Wiley, 1998) and coauthored (with I. Mozetic and N. Lavrac) *KARDIO: A Study in Deep and Qualitative Knowledge for Expert Systems* (MIT Press, 1989). His e-mail address is ivan.bratko@fri.uni-lj.si.

Dorian Šuc is a teaching assistant in the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. He finished his M.Sc. in 1998 and received his Ph.D. in computer science and informatics from the University of Ljubljana in 2001. For his doctoral dissertation, he received the 2001 ECCAI Artificial Intelligence Dissertation Award, sponsored by the European Coordinating Committee for Artificial Intelligence. His research interests include machine learning, human skill reconstruction and behavioral cloning, reinforcement learning, qualitative modeling, and induction of qualitative models as a general method to discover qualitative relations in data. His e-mail address is dorian.suc@fri.uni-lj.si.