Induction of Qualitative Trees

Dorian Šuc[1] and Ivan Bratko[1]

Faculty of Computer and Information Science, University of Ljubljana,
Tržaška 25, 1000 Ljubljana, Slovenia
{dorian.suc, ivan.bratko}@fri.uni-lj.si

Abstract. We consider the problem of automatic construction of qualitative models by inductive learning from quantitative examples. We present an algorithm QUIN (QUalitative INduction) that learns *qualitative trees* from a set of examples described with numerical attributes. At difference with decision trees, the leaves of qualitative trees contain qualitative functional constraints as used in qualitative reasoning. A qualitative tree defines a partition of the attribute space into the areas with common qualitative behaviour of the chosen class variable.
We describe a basic algorithm for induction of qualitative trees, improve it to the heuristic QUIN algorithm, and give experimental evaluation of the algorithms on a set of artificial domains. QUIN has already been used to induce qualitative control strategies in dynamic domains such as controlling a crane or riding a bicycle (described elsewhere) and can be applied to other domains as a general tool for qualitative system identification.

1 Introduction

Various studies in recent years showed that for some tasks qualitative models are more suitable than classical quantitative (numerical) models. These tasks include diagnosis [1], generating explanation of the system's behaviour [7] and designing novel devices from first principles [17]. Qualitative reasoning about processes (QPT-Qualitative Process Theory) [8] and qualitative simulation (QSIM) [9] with Qualitative Differential Equations enable a kind of commonsense reasoning by abstracting numerical values into qualitative values and real functions into qualitative constraints.

Besides qualitative reasoning and qualitative simulation, qualitative models can also be used to guide machine learning [3] and offer a space for solution optimization [16,12]. However, building a qualitative model for a complex system requires significant knowledge and is a time-consuming process. For this reason, many researchers are addressing the problem of automatic generation of qualitative models. One approach is to build models from existing libraries of model fragments [8]. Another approach is to learn a model of a physical system from behaviours using existing knowledge of processes and mechanisms commonly found in physical systems [5]. Less knowledge-intensive approaches [4,2,6,10] use inductive learning of qualitative differential equations from a set of qualitative behaviours.

L. De Raedt and P. Flach (Eds.): ECML 2001, LNAI 2167, pp. 442–453, 2001.

In this paper we present algorithm QUIN for induction of qualitative constraint trees from examples described with numerical attributes. At difference with decision trees, the leaves of qualitative trees contain qualitative functional constraints that are inspired by qualitative proportionality predicates Q_+ and Q_- as defined by Forbus [8]. This is a novel approach to automatic generation of qualitative models. To our knowledge, no study has yet addressed the induction of similar tree-structured qualitative constraints from numerical examples.

The motivation for learning of qualitative trees came from applications in reconstruction of human skill to control dynamic systems, such as a plane or a crane. Our experiments in controlling a crane [12] and double pendulum called acrobot [13], showed that qualitative strategies are suitable as explanatory models of human control skill and can be used as spaces for controller optimization. These qualitative strategies were obtained as abstractions of quantitative control strategies induced from the logged data from skilled human operators. The learning of qualitative trees *directly* from numerical examples is a natural extension of that approach.

The structure of the paper is as follows. First we give the learning problem description for induction of qualitative trees and define monotonicity constraints, called qualitatively constrained functions. Then we describe how qualitatively constrained functions are learned from a set of numeric examples. This provides the basis for induction of qualitative trees in Section 4. Finally, we give experimental evaluation of the algorithms on a set of artificial domains and conclude.

2 Learning Problem Description

2.1 Qualitative Trees

We consider the usual setting of classification learning, but in our case the hypothesis language involves *qualitative constraints*. Let there be N learning examples. Each example is described by $n+1$ continuous variables $X_1, \ldots, X_{n+1}$. The variable X_{n+1} is called the *class*, and the others are called *attributes*.

Given the learning examples, our problem is to learn a hypothesis that separates the areas of attribute space which share a common qualitative behaviour of the class variable. We learn such hypotheses in the form of *qualitative trees*. A qualitative tree is a binary tree with internal nodes called splits and *qualitatively constrained functions* in the leaves. The splits define a partition of the state space into areas with common qualitative behaviour of the class variable. A split consists of a split attribute and a split value. Qualitatively constrained functions (abbreviated QCFs) in leaves define qualitative constraints on the class variable.

Fig. 1 shows an example of qualitative tree induced from a set of example points for the function $z = x^2 - y^2$.

2.2 Qualitatively Constrained Functions

Qualitatively constrained functions are inspired by the qualitative proportionality predicates Q_+ and Q_- as defined by Forbus [8] and are also a generalization

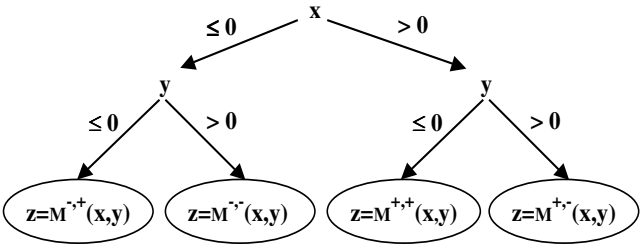


Fig. 1. A qualitative tree induced from a set of examples for the function $z = x^2 - y^2$. The rightmost leaf, applying when attributes x and y are positive, says that z is strictly increasing in its dependence on x and strictly decreasing in its dependence on y.

of the qualitative constraint M^+, as used in [9]. We use QCFs to define qualitative constraints on the class variable. A QCF constrains the qualitative change of the class variable in response to the qualitative changes of the attributes.

A *qualitatively constrained function* $M^{s_1,\ldots,s_m}$: $\Re^m \mapsto \Re$, $s_i \in \{+,-\}$ represents an arbitrary function with $m \leq n$ continuous attributes that respect the qualitative constraints given by signs s_i. The qualitative constraint given by sign $s_i = +$ ($s_i = -$) requires that the function is strictly increasing (decreasing) in its dependence on the i-th attribute. We say that the function is *positively related* (negatively related) to the i-th attribute. $M^{s_1,\ldots,s_m}$ represents any function which is, for all $i = 1, \ldots, m$ positively (negatively) related to the i-th argument, if $s_i = +$ ($s_i = -$).

Note that the qualitative constraint given by sign $s_i = +$ only states that when the i-th attribute increases, the QCF will also increase, *barring other changes*. It can happen that a QCF with the constraint $s_i = +$ decreases even if the i-th attribute increases, because of a change in another attribute. For example, consider the behaviour of gas pressure in a container: $Pres \times Vol/Temp = const$. We can express the qualitative behaviour of gas by QCF $Pres = M^{+,-}(Temp, Vol)$. This constraint allows that the pressure decreases even if the temperature increases, because of a change in the volume. Note however, that gas qualitative behaviour is not consistent with $Pres{=}M^+(Temp)$.

QCFs are concerned with qualitative changes and qualitative change vectors. *Qualitative change* q_j is the sign of change in continuous variable X_j, where $q_j \in \{pos, neg, zero\}$, corresponding to *positive*, *negative* or *zero* change. A *qualitative change vector* is a vector of qualitative changes of the variables. We define *QCF-prediction* $P(s_i, q_i)$ as:

$$P(s_i, q_i){=}\begin{cases} \text{pos, if } (s_i{=}+ \wedge q_i{=}pos) \vee (s_i{=}- \wedge q_i{=}neg) \\ \text{neg, if } (s_i{=}+ \wedge q_i{=}neg) \vee (s_i{=}- \wedge q_i{=}pos) \\ \text{zero, otherwise} \end{cases}$$

A qualitative change vector $q = (q_1, \ldots, q_{n+1})$ is *consistent* with a given QCF $M^{s_1,\ldots,s_m}$, if the QCF does not reject the qualitative change of the class variable, that is, if either (a) class qualitative change is zero, (b) all attribute's QCF-

predictions are zero, or (c) there exists an attribute whose QCF-prediction is equal to the class's qualitative change.

A QCF does not always uniquely predict the class's qualitative change given the qualitative changes of the attributes. *Qualitative ambiguity*, i.e. ambiguity in the class's qualitative change appears whenever there exist both positive and negative QCF-predictions or whenever all QCF-predictions are zero. In this case any qualitative class change is consistent with the QCF.

3 Learning Qualitatively Constrained Functions

When learning a QCF from a set of numerical examples we are interested in a QCF that is consistent with most of the examples, i.e. in the *"minimal cost"* QCF. For this reason we define *error-cost* $E(g)$ of a QCF g (defined later in Eq. 3) that penalizes g with inconsistent and ambiguous qualitative change vectors at every example. The "minimal cost" QCF is learned from a set of numerical examples by first forming qualitative change vectors from examples and then minimizing error-cost of a QCF over all possible QCFs.

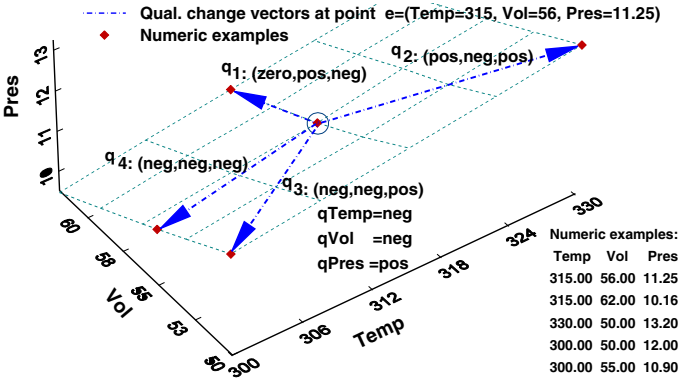
First, every pair of examples e and $f \neq e$ is used to form a qualitative change vector with qualitative changes $q_{(e,f),j} \in \{pos, neg, zero\}$, $j = 1, \ldots, n{+}1$ defined as:

$$q_{(e,f),j} = \begin{cases} \text{pos, if } x_{f,j} > x_{e,j} + Tzero_j \\ \text{neg, if } x_{f,j} < x_{e,j} - Tzero_j \\ \text{zero, otherwise} \end{cases} \quad (1)$$

where $Tzero_j$ denotes a user-defined steady threshold defining negligible changes of j-th attribute. The default value of $Tzero_j$ is 1% of the difference between maximal and minimal value of j-th attribute. Typically, many pairs of examples map into the same qualitative change vector. A qualitative change vector is either consistent or not consistent with a given QCF. Note that a consistent qualitative change vector can also be ambiguous for a given QCF.

We illustrate the method to find the "minimal cost" QCF by an example of gas in the container. Fig. 2 gives five numerical examples described with attributes $Temp$ and Vol and class $Pres$, giving gas temperature, volume and pressure according to equation $Pres = 2\ Temp/Vol$. There are five numerical points, each with four qualitative change vectors with respect to other points. Fig. 2 illustrates qualitative change vectors q_1, q_2, q_3 and q_4 at the circled point e=($Temp$=315, Vol=56, $Pres$=11.25). To find the "minimal cost" QCF at point e, qualitative change vectors that are inconsistent and ambiguous with each possible QCF are counted. Consider for example QCF $Pres = M^{+}(Temp)$. Qualitative change vector q_3=(q_{Temp}=neg, q_{Vol}=neg, q_{Pres}=pos) is not consistent with this QCF since the QCF-prediction of the only attribute ($P(+, q_{Temp})$=neg) is different than the qualitative class change q_{Pres}=pos. Qualitative change vector q_1=(q_{Temp}=$zero$, q_{Vol}=pos, q_{Pres}=neg) is ambiguous with respect to this QCF.

The table in Fig. 2 gives qualitative change vectors at point e that are inconsistent with and ambiguous for each possible QCF. QCF $M^{+,-}(Temp, Vol)$ is



QCF	$Inconsis.$	$Ambig.$	QCF	$Inconsis.$	$Ambig.$
$Pres{=}M^{+}(Temp)$	q_3	q_1	$Pres{=}M^{+,+}(Temp,Vol)$	q_1, q_3	q_2
$Pres{=}M^{-}(Temp)$	q_2, q_4	q_1	$Pres{=}M^{+,-}(Temp,Vol)$	/	q_3, q_4
$Pres{=}M^{+}(Vol)$	q_1, q_2, q_3	/	$Pres{=}M^{-,+}(Temp,Vol)$	q_1, q_2	q_3, q_4
$Pres{=}M^{-}(Vol)$	q_4	/	$Pres{=}M^{-,-}(Temp,Vol)$	q_4	q_2

Fig. 2. Gas in the container example: five numerical examples are represented as points in the attribute space. The arrows denote the qualitative change vectors at the circled point e. For example, the point ($Temp$=300, Vol=50, $Pres$=12) gives qualitative change vector q_3=(q_{Temp}=neg, q_{Vol}=neg, q_{Pres}=pos) with point e. The table bellow gives qual. change vectors that are inconsistent and ambiguous for each possible QCF.

the only QCF consistent with all qualitative change vectors and is the "minimal cost" QCF. It also minimizes the error-cost (defined below) over all QCFs.

The error-cost of a QCF is based on the minimum description length principle [11]. Basically it is defined as the number of bits needed to code the QCF plus the number of bits to code the inconsistent and ambiguous qualitative change vectors as follows. Let $\mathcal{C}_e(q)$ and $n_e(q)$ denote respectively the set and the number of all examples f that form, with e, qualitative change vector q (with qualitative changes q_j):

$$\mathcal{C}_e(q) = \{f | \forall j = 1, \ldots, n+1 : q_{(e,f),j} = q_j\}, \quad n_e(q) = |\mathcal{C}_e(q)| \tag{2}$$

In the above gas in the container example, the circled point e and qualitative change vector q_3 give $\mathcal{C}_e(q_3)$ = {(Temp=300,Vol=50,Pres=12)} and $n_e(q_3)$=1. The *error-cost* $E(g)$ of a QCF g that mentions m out of all n attributes is:

$$\begin{aligned} E(g) = &\log_2 n + m(\log_2 n + 1) + \\ &\log_2 N_{nonamb} + N_{reject}(\log_2 N_{nonamb}) + \log_2 N_{amb} + N_{amb} \end{aligned} \tag{3}$$

Here N_{reject} denotes the number of example pairs (e,f), $f \neq e$, that form a qualitative change vector that is not consistent with QCF g and is computed as the sum of $n_e(q)$ over all examples e and over all qualitative change vectors q that are not consistent with g. Similarly, N_{amb} and N_{nonamb} denote the sum of $n_e(q)$

of vectors q that are respectively ambiguous and not ambiguous for g. This error-cost is based on the following encoding: we code the QCF, the indexes of N_{reject} inconsistent qualitative change vectors (each index requires $\log_2 N_{nonamb}$ bits since ambiguous qualitative change vectors are always consistent) and one bit for each ambiguous qualitative change vector. Note that we specify just the signs of class changes for ambiguous qualitative change vectors and we do not need to specify which qualitative change vectors are ambiguous, since all qualitative change vectors of particular forms are ambiguous for a given QCF.

The given error-cost penalizes inconsistent and ambiguous qualitative change vectors formed from every pair of examples. We later define (Eq. 7) a different error-cost and refer to here defined error-cost also as *ep-QUIN error-cost* (ep standing for every pair).

4 Learning Qualitative Trees

Here we describe how a qualitative tree is learned from a set of numeric examples. The algorithm uses top-down greedy approach and is guided by error-cost of a QCF. We define two learning algorithms that use different error-costs. First we give ep-QUIN algorithm that uses error-cost defined in Section 3. Then we present an example showing the myopia of ep-QUIN algorithm and (in Section 4.3) give an improvement of ep-QUIN, i.e. QUIN algorithm.

4.1 Algorithm ep-QUIN

Algorithm ep-QUIN uses top-down greedy approach similar to the ID3 algorithm. Given the examples, ep-QUIN chooses the best split by comparing the partitions of the examples they generate: for every possible split, it splits the examples into two subsets, finds the "minimal cost" QCF in both subsets, and selects the split which minimizes the *tree error-cost* (defined below). It puts the best split in the root of the tree and recursively finds subtrees for the corresponding example subsets, until the best split does not improve the tree error-cost.

The *tree error-cost* of a leaf is the error-cost $E(g)$ (Eq. 3) of the QCF g that is induced from the examples in the leaf. The tree error-cost of an internal node is the sum of error-costs of both subsets plus the cost of the split, i.e. the number of bits needed to encode the split:

$$\begin{aligned} E_{tree} &= E_{left} + E_{right} + SplitCost \\ SplitCost &= \log_2 n + \log_2(Splits_i - 1) \end{aligned} \tag{4}$$

Here E_{left} and E_{right} denote the error-costs in both subsets, n is the number of variables and $Splits_i$ is the number of possible splits for the split variable, i.e. the number of different values of the variable X_i.

4.2 An Example

Here we give a simple example showing the myopia of ep-QUIN's error-cost. Fig. 3 shows the learning dataset consisting of 12 learning examples with one

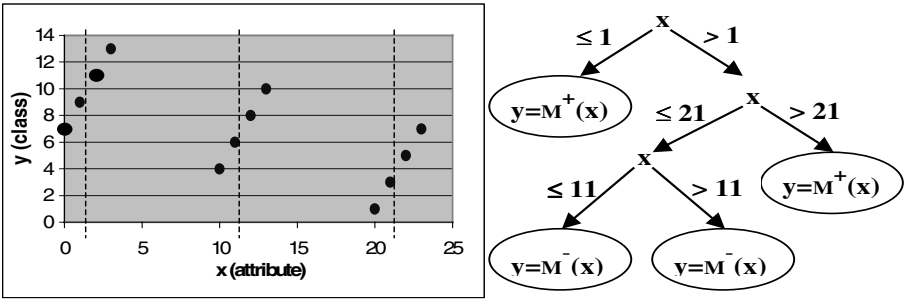


Fig. 3. Example partition and a qualitative tree induced by ep-QUIN. The points on the left figure give 12 learning examples with attribute x and class y. The vertical lines correspond to the example partition from the qualitative tree on the right.

attribute x and class y. The examples correspond to 3 linear functions that are increasing in its argument x:

$$y = \begin{cases} 2x + 7, \text{ x= } 0, 1, 2, 3 \\ 2x - 16, \text{ x= } 10, 11, 12, 13 \\ 2x - 39, \text{ x= } 20, 21, 22, 23 \end{cases} \tag{5}$$

Given these examples ep-QUIN algorithm proceeds as follows. It first finds the "most-consistent" QCF for the whole attribute space and then tries to reduce the error-cost by splitting the attribute space into subregions. There are 12 examples, therefore ep-QUIN forms $12 \times (12-1)$=132 qualitative change vectors. With x as the only attribute there are only two possible QCFs: QCF $y = M^+(x)$ that is consistent with 50 (out of 132) qual. change vectors and $y = M^-(x)$ that is consistent with 82 qual. change vectors. Split minimizing ep-QUIN error-cost is $x \leq 1$, splitting the examples into two subsets. In the first subset ($x \leq 1$) there are only two examples, forming two qualitative change vectors, both consistent with $y = M^+(x)$. The second subset ($x > 1$) is then further divided, resulting in the qualitative tree given in Fig. 3. Note that the split at $x = 11$ splits the examples into two leaves with the same QCF $y = M^-(x)$. A split giving *the same leaves* would be unusual for decision trees, but is correct for qualitative trees since the split divides the examples into monotonic areas.

Clearly the induced qualitative tree is not optimal. We would prefer a qualitative tree with 3 leaves (each with QCF y=$M^+(x)$), that would divide the attribute space into 3 regions corresponding to definition areas of 3 linear functions. One reason is that ep-QUIN does not consider the *locality* of qualitative changes. A human observing Fig. 3 might notice 3 groups of near-by example points and consider the proximity of examples when estimating qualitative changes at a particular example point. In this way a qual. change of nearby points (x=1 and x=2 for example) would be weighted more than a qual. change of far-away points (x=1 and x=13). One might also consider the *consistency* of qualitative class change for a particular vector of qualitative attributes changes. One way to asses the consistency of qual. class change for a particular vector of

qual. attributes changes q_x is to consider 3 nearest neighbors (ignoring the class distance) of a given point that form q_x with this point. Then the confidence in the positive class change (q_y=*pos*) for vector of qual. attributes changes q_x=*pos* at the point at x=0 would be higher than at the point at x=2 since all three corresponding neighbors (with q_x=*pos*) of the first point give positive class change, whereas the second point has one neighbor giving positive class change (point at x=3) and two neighbors giving negative class change (points at x=10,11). Here only neighbors giving q_x=*pos* with the reference point are considered since this is confidence for vector of qualitative attributes changes q_x=*pos*.

An improvement of ep-QUIN, algorithm QUIN follows this idea, i.e. it considers also the locality and consistency of qualitative change vectors. Note that QUIN algorithm, from this learning dataset, induces the above mentioned qualitative tree with 3 leaves, each with QCF y=$M^+(x)$.

4.3 Algorithm QUIN

QUIN is an improvement of the ep-QUIN algorithm that follows the ideas described in the example above. The algorithms differ in their error-cost. ep-QUIN's error-cost (Eq. 3) uses only counts of inconsistent and ambiguous qualitative change vectors. QUIN's error-cost (defined later in Eq. 7) uses heuristic confidence estimates of qualitative change vectors that take into account the proximity of examples and the consistency of qualitative change vectors.

The proximity of the examples e and f is evaluated by Gaussian weight $w(e,f) = e^{\frac{-d(e,f)}{2K^2}}$, where $d(e,f)$ is the normalized Euclidean distance of the examples that considers only the attributes of the two examples and K is a user-defined parameter with the default value computed as 10 % of maximal Euclidean distance of two examples from the learning dataset. By changing K, more (smaller K) or less importance is assigned to the distance between examples and therefore more local (smaller K) or more global estimates are used.

We say that example f is a *q-neighbor* of example e if the qualitative change vector $q_{(e,f)}$ is equal to q in all the qualitative attribute changes (but not necessarily in the qualitative class change). Let $\mathcal{N}_e(q)$ denote the set of all q-neighbors of e and $\mathcal{N}_e^k(q)$ the set of k *nearest q-neighbors of example e*. The confidence estimate $w_e(q)$ of the qualitative change vector q at example e takes into account the proximity of the k nearest q-neighbors and the consistency of qualitative class change w.r.t. the k nearest q-neighbors. The intuition in estimates $w_e(q)$ is in modelling the local probability of qualitative class change q_{n+1} given the attribute's changes $q_i, i = 1, \ldots, n$:

$$\begin{aligned}
&\mathcal{N}_e(q) = \{f | \forall j = 1, \ldots, n : q_{(e,f),j} = q_j\} \\
&\mathcal{N}_e^k(q) = \text{set of } k \text{ examples } f \in \mathcal{N}_e(q) \text{ minimizing } d(e,f) \\
&\mathcal{C}_e^k(q) = \{f \in \mathcal{N}_e^k(q) | q_{(e,f),n+1} = q_{n+1}\} \\
&w_e(q) = \frac{1}{|\mathcal{N}_e^k(q)|} \sum_{f \in \mathcal{C}_e^k(q)} w(e,f)
\end{aligned} \tag{6}$$

Here $\mathcal{C}_e^k(q)$ denotes the set of k nearest q-neighbors of e with the qualitative class change equal as in q and k is a user-defined parameter with the default value 5. Note that using $\left|\mathcal{N}_e^k(q)\right|$ in denominator penalizes estimate $w_e(q)$ if nearest q-neighbors of e differ in qualitative class change.

Lets consider the example in Fig. 3. Suppose that e_0 and e_2 are respectively the points at x=0 and x=2. Then for qualitative change vector q=(pos, pos) the set $\mathcal{N}_{e_2}^3(q)$ are the examples at x=3,10,11 and $\mathcal{C}_{e_2}^3(q)$ is the example at x=3. The confidence estimate $w_{e_2}(q)$ is one third $(1/\left|\mathcal{N}_{e_2}^3(q)\right|)$ of the Gaussian weight $w(e_2, f)$ that evaluates the proximity of examples e_2 and the example at x=3. The sets $\mathcal{N}_{e_0}^3(q)$ and $\mathcal{C}_{e_0}^3(q)$ are the examples at x=1,2,3. Therefore qualitative change vector q=(pos, pos) has higher confidence estimate at e_0 than at e_2.

QUIN's error-cost is similar to ep-QUIN's error-cost, but with QUIN's error-cost the qualitative change vectors q are weighted according to their confidence estimates $w_e(q)$ (Eq. 6). Because of this weighting QUIN's error-cost is heuristic and does not correspond to a particular coding of examples. The error-cost $E(g)$ of a QCF g that mentions m out of all n attributes is:

$$\begin{aligned} E(g) = &\log_2 n + m(\log_2 n + 1) + \\ &\log_2 W_{nonamb} + W_{reject}(\log_2 W_{nonamb}) + \log_2 W_{amb} + W_{amb} \end{aligned} \tag{7}$$

Here W_{reject} denotes the sum (over all examples e) of estimates $w_e(q)$ of qualitative change vectors q that are not consistent with g. Similarly W_{amb} and W_{nonamb} denote the sum of estimates $w_e(q)$ of vectors q that are respectively ambiguous and not ambiguous for g.

Besides the improved error-cost, QUIN uses also a more efficient algorithm for selecting the "minimal cost" QCF. ep-QUIN finds the "minimal cost" QCF by a simple exhaustive search algorithm that forms all possible QCFs and selects the one with the smallest error-cost. This requires the number of error-cost computations that is exponential in the number of attributes. Instead of the exhaustive search, QUIN uses a greedy heuristic algorithm that requires the number of error-cost computations that is quadratic in the number of attributes, but does not guarantee the "minimal cost" QCF. The idea is to start with the QCF that minimizes error-cost over all QCFs that use only one attribute, and then use error-cost to refine the current QCF with another attribute.

5 Experimental Evaluation of QUIN Algorithm

Here we experimentally evaluate learning qualitative trees from numerical examples on a set of artificial domains. Since the usual error measures, such as accuracy or mean squared error, are not suitable for qualitative models, we define *qualitative tree performance* as a measure of qualitative tree prediction error. Then we describe the details of our experiments and give results.

We define *qualitative tree performance* as a pair of *qualitative consistency* and *qualitative ambiguity*, that are the percentages of qualitative change vectors that are respectively consistent with, and ambiguous for a qualitative tree. The examples are partitioned according to the splits of the tree and for each example

Table 1. Experimental results: the second column gives the description of the domains, i.e. the class variables c as the function of uniformly distributed attributes. The third column gives qualitative performance of the optimal qualitative tree. The fourth and the fifth column give qualitative performance of trees induced by ep-QUIN and QUIN.

Domain	Class variable	Optimal	ep-QUIN	QUIN
Sin	$c = \sin(\pi \times \frac{x}{10})$	100/6	100.0/6.4	100.0/5.8
SinLn	$c = \frac{x}{10} + sign(x) \times \sin(\pi \times \frac{x}{10}))$	100/4	96.2/11.8	**98.7/3.1**
Poli	$c = \ln(10^4 + \lvert(x+16)(x+5)(x-5)(x-16)\rvert)$	100/10	96.9/27.4	**96.4/6.3**
Signs	$c = \begin{cases} sign(u+0.5)(x-10)^2, \text{ if } v \geq 0 \\ sign(u-0.5)(y+10)^2, \text{ otherwise} \end{cases}$	100/2	93.2/19.4	**97.4/8.1**
QuadA	$c = x^2 - y^2$	100/44	99.0/27.1	**99.7/33.9**
QuadB	$c = (x-5)^2 - (y-10)^2$	100/44	98.3/17.4	99.7/27.2
SQuadB	$c = sign(u)((x-5)^2 - (y-10)^2)$	100/38	96.6/32.7	98.0/31.1
YSinX	$c = y \sin(\pi \times \frac{x}{10})$	100/39	94.0/62.0	**81.6/12.4**

partition the qualitative change vectors are formed. Consistency is the percentage of qualitative change vectors that are consistent with QCFs in the corresponding leaves. Ambiguity is the percentage of qualitative change vectors which give ambiguous QCF-prediction in the corresponding leaves.

When giving the qualitative tree performance we use the notation c/a, where c denote the consistency and a ambiguity of a qual. tree on the example set. We prefer a qual. tree with high consistency and low ambiguity. However, low ambiguity is not possible when many attributes are used in the same QCF.

Ambiguity is used in qualitative tree performance since consistency of the qualitative tree does not give full information about the qualitative tree prediction strength. Consider for example a dataset with two attributes x_1 and x_2 that are pairwise equal for all the examples. Then QCF $M^{+,-}(x_1, x_2)$ is consistent with all the examples, but is not useful in predicting qualitative class change, since it is also ambiguous for any qualitative change vector from the dataset. The qualitative tree performance would in this case be 100%/100%.

The learning algorithms were evaluated on a set of artificial domains with uniformly distributed attributes $x, y \in [-20, 20]$ and $u, v \in [-1, 1]$. The class variables c, i.e. the variables to be predicted by a qualitative trees were computed as given in Table 1. For learning we used just the relevant attributes plus 2 uniformly distributed (noise) attributes that do not affect the class value.

To compare qualitative performance of the trees produced by the two algorithms we used learning sets consisting of 200 examples and a separate test set consisting of 500 examples. The results in Table 1 are averages of 10 runs. The table also gives the qualitative performance of the *optimal qualitative trees*, i.e. the smallest trees with 100% consistency. An example of such optimal qualitative tree for the domain *QuadA* is given on Fig. 1. Note that the ambiguity of the optimal trees is not zero. This is either because QCFs in leaves use more than one attribute or because of zero change of all the attributes used in a QCF.

On most of the domains, QUIN has better qualitative performance, i.e. higher qualitative consistency and lower ambiguity. Usually QUIN induces a "near-optimal" tree, i.e. small tree with consistency over 96%. The exception is the domain $YSinX$ where the optimal tree has 16 leaves. ep-QUIN and QUIN fail to find a "near-optimal" tree in this domain.

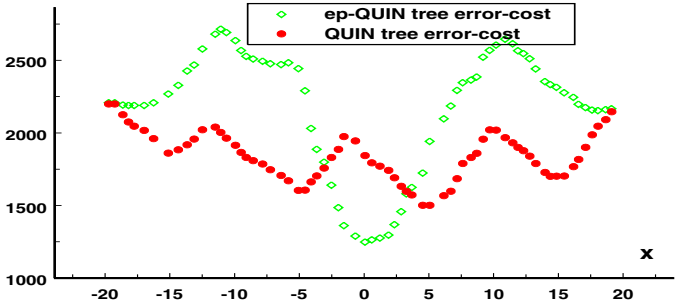


Fig. 4. ep-QUIN's and QUIN's criterion of split goodness (i.e. tree error-cost) for attribute x in domain Sin. Note that x values with minima of QUIN's error cost (x=5,-5,-15, 15) correspond to points dividing $sin(\pi x/10)$ into monotonic areas.

When we compared the trees induced by ep-QUIN and QUIN, we observed that QUIN trees are usually smaller and better correspond to the human intuition. An example is the domain Sin where QUIN usually induces optimal tree with 5 leaves, whereas ep-QUIN induces larger tree. Fig. 4 gives ep-QUIN's and QUIN's tree error-cost for the only relevant attribute x in the domain Sin. Both algorithms put the split that minimizes the tree error-cost in the root of the qualitative tree. Minima of QUIN tree error-cost (x=5,-5,-15, 15) correspond to splits dividing $sin(\pi x/10)$ into monotonic areas, whereas minimum of ep-QUIN tree error-cost (x=0) is not a good split since it splits a monotonic area $x \in [-5, 5]$. QUIN puts the split $x \leq 5$ in the root of the qualitative tree, whereas ep-QUIN chooses a suboptimal split $x \leq 0$ as the topmost split.

6 Conclusions

We presented the QUIN algorithm for induction of qualitative trees and evaluated it on a set of artificial domains. To our knowledge, no study has yet addressed the induction of similar tree-structured qualitative constraints from numerical examples. QUIN, a heuristic improvement of the basic ep-QUIN algorithm, usually produces preferable qualitative trees and is more time-efficient. Our experiments described here and in [14] show that QUIN can handle noisy data, and, at least in simple domains, produces qualitative trees that correspond to human intuition.

In [14,15] QUIN has already been used to induce qualitative control strategies in dynamic domains such as controlling a crane or riding a bicycle. In both

domains some surprising and non-trivial aspects of human control skill have been induced. QUIN can be applied to other domains as a general tool for qualitative system identification.

Acknowledgements. The work reported in this paper was partially supported by the European Fifth Framework project Clockwork and the Slovenian Ministry of Education, Science and Sport.

References

1. Bratko, I., Mozetič, I., Lavrač, N.: *KARDIO: a Study in Deep and Qualitative Knowledge for Expert Systems*. MIT Press (1989).
2. Bratko, I., Muggleton, S., Varšek, A.: Learning qualitative models of dynamic systems. In *Proc. of the 8th Int. Conf. on Machine Learning* (1991).
3. Clark, P., Matwin, S.: Using qualitative models to guide inductive learning. In *Proc. 10th Int. Conf. on Machine Learning*, 49–56. Morgan Kaufmann (1993).
4. Coiera, E.: Generating qualitative models from example behaviours. *Technical Report Technical report 8901*, University of New South Wales (1989).
5. Doyle, R.: *Hypothesizing Device Mechanisms: Opening Up the Black Box*. Ph.D. Dissertation, Massachusetts Institute of Technology (1988).
6. Džeroski, S., Todorovski, L.: Discovering dynamics. In *Proceedings of the 10th International Conference on Machine Learning* 97–103. Morgan Kaufmann (1993).
7. Falkenhainer, B., Forbus, K.: Self explananatory simulations: an integration of qualitative and quantitative knowledge. In *Proceedings of the 4th International Workshop on Qualitative Physics* (1990).
8. Forbus, K.: Qualitative process theory. *Artificial Intelligence* (1984) 24:85–168.
9. Kuipers, B. J.: Qualitative simulation. *Artificial Intelligence* (1986) 29:289–338.
10. Richards, B., Kraan, I., Kuipers, B: Automatic abduction of qualitative models. In *Proc. of the National Conf. on Artificial Inteligence*. AAAI/MIT Press (1992).
11. Rissanen, J.: Modelling by shortest data description. *Automatica* (1978) 14:465–471.
12. Šuc, D., Bratko, I.: Symbolic and qualitative reconstruction of control skill. *Electronic Transactions on Artificial Intelligence, Section B* (1999) 3:1–22. http://www.ep.liu.se/ej/etai/1999/002/.
13. Šuc, D., Bratko, I.: Skill modelling through symbolic reconstruction of operator's trajectories. *IEEE Transaction on Systems, Man and Cybernetics, Part A* (2000), 30(06):617–624.
14. Šuc, D: *Machine reconstruction of human control strategies*. Ph.D. Dissertation, Faculty of Computer and Information Sc., University of Ljubljana, Slovenia (2001). http://ai.fri.uni-lj.si/dorian/MLControl/MLControl.htm.
15. Šuc, D., Bratko, I: Qualitative induction. In *Proceedings of the 15th International Workshop on Qualitative Reasoning* (2001). Accepted for publishing.
16. Varšek, A., Urbančič, T., Filipič, B.: Genetic algorithms in controller design and tuning. *IEEE Trans. on Systems, Man and Cybernetics* (1993), 23(6):1330–1339.
17. Williams, B.: Interaction-based invention: designing devices from first principles. In *Proceedings of the 4th International Workshop on Qualitative Physics* (1990).