Learning Qualitative Models From Example Behaviours

Enrico W. Coiera *
Department of Computer Science, University of New South Wales
P.O. Box 1 , Kensington 2033 N.S.W., Australia
email: ric@cheops.eecs.unsw.au

1 Introduction

Qualitative models provide the domain knowledge for qualitative reasoning systems, and the automatic generation of such models is an important knowledge acquisition task. Qualitative Simulation is a key qualitative reasoning technique. It is proposed that simulation systems like Kuipers' QSIM [Kuipers 86] contain a powerful description language, based as they are on qualitative mathematics, for the automation of model capture. A simulation system uses qualitative models and mathematical knowledge to generate behaviours. The Genmodel program described here uses the same knowledge as a description language to proceed in the opposite direction, from behaviours to models.

It is assumed that a set of dynamic qualitative system behaviours are available to form a training set of positive examples. These behaviours are either presented by a domain expert or are extracted from numerical data by curve fitting or other techniques. The program's task is to search for qualitative relationships between functions that will permit the example behaviour. These relationships together form a model of the system that produced the behaviours.

2 Related Work

In QSIM, a model is a collection of qualitative differential equations. Some work has been done on the automated discovery of quantitative equations [Langley 81] [Falkenheimer 86], but the qualitative models used by Kuipers and others have been hand-constructed by experts or derived from expert protocols [Kuipers 84] [Forbus 88]. [Mozetic 87] reports a system for learning qualitative models represented as non-recursive typed Horn clauses. The system is given partial knowledge of the model - its structure (components and connections), along with some instances of the model's behaviour.

Genmodel does not require structural information, just component names and behaviours. Because it describes data in qualitative terms, it does not force a 'best' solution like the quantitative equation learning programs. Assuming the data is error free, qualitative equations are correct, not just approximate. The qualitative constraint language allows the qualitative *form* of general polynomial equations, including functions like exponentiation and *log*, to be discovered. This is because qualitative differential equations can map onto many different ordinary differential equations. Genmodel assumes that data is error free.

3 Background Knowledge

The qualitative mathematical formalism used in QSIM can be viewed as a description language for model development. The language contains the background knowledge needed to generate models, and specifies:

Form of Functions QSIM specifies that qualitative functions are real valued parameters which vary continuously over time and are continuously differentiable. The qualitative description of a function value comprises a pair $< qval, qdir >$. The *qval* is a real value or real interval, and the *qdir* is the sign of its time derivative. The *qualitative behaviour* of a function is defined as the temporal sequence of qualitative state values, and this is the form of the examples used with the model generation system.

Permissible Function Relationships The permissible relationships between functions correspond to the basic QSIM constraints. The constraints as defined in [Kuipers 86] are: $Add(f, g, h)$, $Mult(f, g, h)$, $Minus(f, g)$, $Deriv(f, g)$, $M^+(f, g)$ and $M^-(f, g)$.

The strong mathematical foundation of the language enhances its generality, and should allow it to be used in a variety of domains.

4 Generating the Model

Given the QSIM description language, and some examples of a system's behaviour, a space of possible relationships among the functions in the example is defined.

*This work was carried out in part with the support of a New South Wales Government Medical Research Training Scholarship, and has also been assisted in part by a research award from the Medical Engineering Research Organisation.

The task of the model generating program is to identify and narrow this space to leave only those constraint relationships consistent with the examples.

Genmodel proceeds in the following stages:

Generate Landmark Lists Inspection of the example behaviour allows landmark lists for each function to be constructed. Each distinct state in an example behaviour can produce sets of corresponding values. Both these sets of values are required for the application of the qualitative constraint language in the following stages. [Kuipers 86] discusses this in detail.

Generate Initial Search Space Given a set of functions and their observed behaviour over time, a large but finite number of qualitative constraints might exist between the functions, assuming a closed world. For n functions and m constraints that hold between p functions, the potential search space for relationships between functions is $m^n C_p$.

The QSIM description language is formed into a set of productions. An exhaustive search of the possible constraint relationships between functions in an initial example is performed. The productions are used to generate all constraints encountered in the search consistent with the example behaviour. For example, QSIM's derivative constraint is reformulated into the production:

$$deriv(f, g) \leftarrow f'(t) = g(t)$$

If the values of functions in the example behaviour are consistent with the right hand side of the production then the left hand relationship is generated between them as a possible model constraint. For example, an example state contains functions a and b with values $f(a) =< a(1), inc >$ and $f(b) =< 0/\infty, dec >$. The relationship $deriv(a, b)$ is supported because b is positive, and thus corresponds to the sign of the derivative of a ie inc.

Filtering Generated Constraints Each subsequent example behaviour can potentially reduce the number of generated constraints by filtering those that are not consistent with it. The productions are now used to test the initially generated constraints against the current example behaviour. Constraints are deleted from the set of generated constraints when no production can be found to support them with the current example.

Removing Redundant Constraints Once all the examples have been processed, the remaining constraints form the desired model. Many of these constraints are redundant, and can be also filtered. For example $M^+(a, b)$ and $M^+(b, a)$ specify the same relationship, and one of these can be eliminated. Similar redundancies exist for add, $mult$, $minus$ and M^-.

5 Learning the Bathtub

The Bathtub model can reach a steady partially filled state with an open drain and constant water flow running into the bathtub. The example behaviour below shows three distinct qualitative states. The state t(0) corresponds to the initial state as water flows into the tub, t(0)/t(1) is a transition state, and t(1) is the final equilibrium. Each function is shown alongside its qualitative description during the example state. The functions are: amount of fluid in tub (amt), level of fluid (lev), pressure of fluid (pre), drain (dra), out flow (out), in flow (inf) and net flow (net). 70 constraints can be found to support the initial state. Each further state in the behaviour reduces the possible constraints in the model, until only 8 are present at the end.

```
[example t(0)
     amt       0            inc
     lev       0            inc
     pre       0            inc
     dra       open         std
     out       0            inc
     if        if(0)        std
     net       0 / inf      dec]
Number of Constraints = 70

[example t(0) / t(1)
     amt       0 / full     inc
     lev       0 / top      inc
     pre       0 / inf      inc
     dra       open         std
     out       0 / inf      inc
     if        if(0)        std
     net       0 / inf      dec]
Number of Constraints = 18

[example t(1)
     amt       full         std
     lev       top          std
     pre       pre(1)       std
     dra       open         std
     out       out(1)       std
     if        if(0)        std
     net       0            std]
Number of Constraints = 8
```

The Bathtub model generated by Genmodel is:

```
deriv(amt, net)
deriv(lev, net)
deriv(pre, net)
deriv(out, net)
mplus(amt, lev)
mplus(lev, pre)
add(out, net, if)
mult(pre, dra, out)
```

There are three extra constraints in the learnt model not present in the target model:

```
deriv(lev, net)
deriv(pre, net)
deriv(out, net)
```

The models produced by Genmodel from positive examples are usually overconstrained because the search is specific to general. The most specific model capable of producing the example behaviours is generated. These extra constraints admit the example behaviour, and could be filtered if examples were presented in which these constraints were violated. When the models are used with a qualitative simulator, they reproduce the examples they were constructed from.

6 Adding Constraints with Negative Examples

When qualitative simulation is performed on a model, extra behaviours may be produced that are unwanted. These behaviours result either through the ambiguity of the qualitative simulation language, or because the model itself is underconstrained.

If the unwanted behaviours are treated as negative examples for the model generating program, and the model is underconstrained, then the examples can be used to identify additional constraints that will prohibit these behaviours. In this case, the program looks for constraints that admit the desired behaviours (positive examples) but which do not admit the negative examples.

Genmodel's strategy is:

- From a positive example, generate all possible constraints that are consistent with it.
- Test each of these constraints against the negative behaviour.
- If the constraint admits the negative behaviour, delete it.
- If the constraint fails to admit the negative behaviour, retain it as a new model constraint.

7 Results

Genmodel[1] has been tested on qualitative behaviours produced from small models. It has not attempted to generate models from real world data. Results for some common models in the qualitative simulation literature [Kuipers 84][Kuipers 86] are presented in Table 1.

It is clear from the table that Genmodel creates models of approximately the same size as the target model, but usually is unable to remove some constraints because of insufficient examples.

8 Conclusion

Genmodel's search is exhaustive at present because of its closed world assumption. For large search spaces heuristic guides may be needed, such as those used with other machine learning programs. Quantitative equation discovery systems look for hidden variables in the data presented through the construction of intermediates. When Genmodel is presented with behaviours from open world systems, it will need to use such heuristics.

Since the Genmodel output is a model using descriptors not present in the original observational statements, the modelling process can be regarded as constructive generalisation [Michalski 83]. The use of the monotonic descriptor is discussed in Michalski's paper (p 111). It may be that other rules for constructive generalisation are also applicable in Genmodel's domain.

Model	Number of Examples	Number of States	Target Number of Constraints	Generated Number of Constraints
Ball	1	5	2	2
Spring	1	8	3	4
U Tube	2	6	6	14
Bathtub	1	3	5	8
Starling	2	6	13	25

Table 1: *Genmodel's Performance with Some Test Models*

References

[Falkenheimer 86] B. C. Falkenheimer, R. S. Michalski, *Integrating Quantitative and Qualitative Discovery: The ABACUS System*, Machine Learning, 1, (1986), 367-401.

[Forbus 88] K. D. Forbus, *Intelligent Computer-Aided Engineering*, AI Magazine, (9), 1988, 23 - 36.

[Kuipers 84] B. Kuipers, J. P. Kassirer, *Causal Reasoning in Medicine: Analysis of a Protocol*, Cognitive Science, 8, (1984), 363 - 385.

[Kuipers 86] B. Kuipers, *Qualitative Simulation*, Artificial Intelligence, 29, (1986), 289-338.

[Langley 81] P. Langley, *Bacon.5:The Discovery of Conservation Laws*, Seventh IJCAI, (1981), 121 - 126.

[Michalski 83] R. S. Michalski, *A Theory and Methodology of Inductive Learning*, in R. S. Michalski, J. G. Carbonell, T. M. Mitchell (eds), Machine Learning: An Artificial Intelligence Approach, Paolo Alto, CA (1983).

[Mozetic 87] I. Mozetic, *The Role of Abstractions in Learning Qualitative Models*, Proceedings of the Fourth International Workshop on Machine Learning, (1987), 242 - 255.

[1] Genmodel is written in UNSW Prolog V4.2, running under Unix ©BSD 4.3 on a Pyramid 90X, and on Apollo DM3000 workstations