Using machine learning techniques to interpret results from discrete event simulation

Dunja Mladenić[1], Ivan Bratko[1,2], Ray J. Paul[3], Marko Grobelnik[1]

[1]J. Stefan Institute, Dpt. of Computer Sc., Jamova 39, Ljubljana, Slovenia

[2]Faculty of Electr. Eng. and Computer Sc., Tržaška 25, Ljubljana University

[3]Brunel University, Dpt. of Computer Sc., Uxbridge, England

e-mail: dunja.mladenic@ijs.si

Abstract

This paper describes an approach to the interpretation of discrete event simulation results using machine learning techniques. The results of two simulators were processed as machine learning problems. In the experiments, the simulation results were interpreted by an attribute-based learning program Retis, and by an Inductive Learning Programming system Markus. Some interesting regularities were thereby automatically discovered.

Key words: discrete event simulation, machine learning, artificial intelligence

1 Introduction

Discrete simulation involves several stages: design and implementation of a model for the system under study, performing experiments with the simulator, analysis and presentation of experimental results. Analysis and presentation of discrete event simulation results is usually done intuitively. Deriving useful conclusions from the simulation results is a demanding task. In this paper we propose to automate this task by means of machine learning techniques.

Machine learning domains can be constructed from the results of discrete event simulations. Each discrete event simulation produces example situations that can be used as examples for machine learning.

Section 2 describes discrete event simulator domains used in our experiments with machine learning. Section 3 describes two machine learning algorithms used to interprete the results of discrete event simulation. Results are discussed in Section 4.

2 Domains description

In our experiments we used two discrete event simulators, the supermarket and the pub. They were both generated by *The Extended Lancaster Simulation Environment eLSE* (Paul and Balmer 1993) and used with the run-in time parameter set to 50 minutes and simulation duration set to 1000 minutes. The interaction of objects for each example simulator is described by *Activity Cycle Diagrams*. These diagrams use two types of objects: circles to denote queues and rectangles to denote activities.

Each discrete event simulation has parameters that can be changed by the user (e.g. the number of cashiers in the supermarket simulation) and some statistical indicators calculated by the simulator (e.g. barmaid utilisation in the pub simulation). The user defined parameters can be viewed as input to the simulator while the statistical parameters are the outputs. In machine learning terminology the input parameters are called *attributes* and the output parameters *classes*. Usually, the result of machine learning describes the relation between a single *class* and the *attributes*. Therefore, for each simulator and for each output value, a machine learning domain was constructed and an experiment with it performed.

2.1 The supermarket domain

In the supermarket domain (Fig. 1), customers arrive and spend some time shopping. After shopping, they have to wait to be served by a cashier. A customer who on arrival sees a queue of more than `QWSize` people waiting to be served, leaves without entering the supermarket. In the described proces, there are some attributes whose influence we want to investigate:

- number of cashiers in the supermarket, `Cashiers` (varied from 1 to 5)
- maximum queue size tolerated by a customer, `QWSize` (varied from 1 to 10)
- arrival rate, `Arrival` (exponentially distributed with a mean of 1 to 10 minutes)

We considered as interesting the following statistical outputs, used as classes in learning:

- customer “utilisation” (ratio between the number of customers that enter the shop and the number of customers that arrive)
- cashier utilisation (fraction of cashier’s time spent actually serving customers)
- length of the customers’ wait queue

From the above parameters, three learning problems can be formulated, each concerned with the relation between the attributes and one of the statistical outputs (e.g. influence of the number of cashiers, customers’ tolerance and arrival rate on customer utilisation).

2.2 Pub simulator domain

In the pub simulator (Fig. 1), customers arrive and wait for the barmaid to pour them a glass of drink. After drinking, they leave or return to the queue for another drink. The barmaids just pour and wash glasses. A glass can be either clean (and empty), full or dirty.

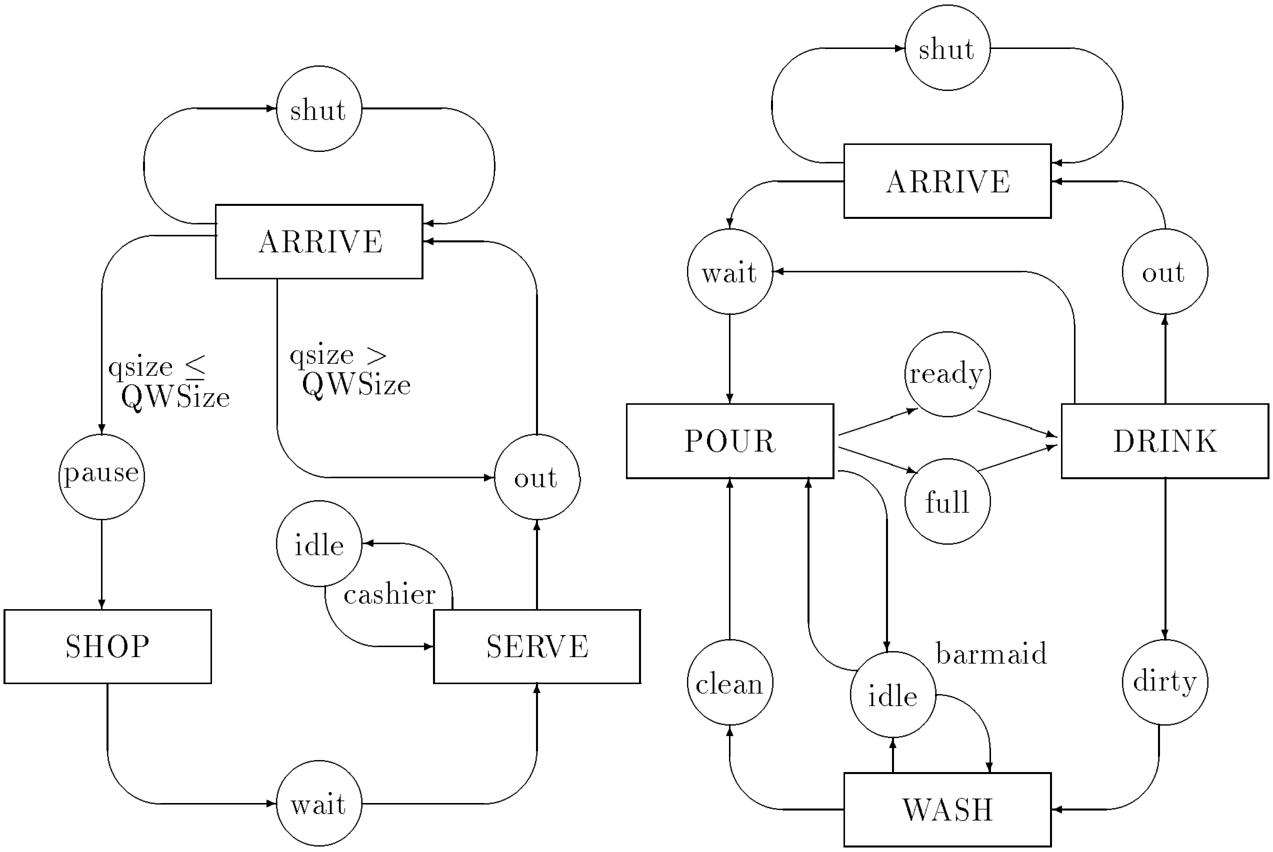

Figure 1: Activity Cycle Diagram for the supermarket (left) and the pub (right) simulation

In the described process, there are the following attributes whose influence on simulation we want to investigate:

- number of barmaids in the pub, `Barmaids` (varied from 1 to 5)
- number of glasses in the pub, `Glasses` (varied: 50, 100, 150)
- arrival rate, `Arrival` (exponentially distributed with a mean of 5 to 17 minutes)
- drinking time, `Drink` (uniformly distributed between X and $X + 3$ minutes; X is varied from 3 to 7)

We considered as interesting the following class values calculated by the simulator:

- barmaid utilisation
- glass utilisation
- length of the customers wait queue
- length of the glasses wait queue

From the above parameters and values, four learning problems can be formulated, each investigating the relation between the above attributes and one of the four class values (e.g. influence of number of barmaids, number of glasses, arrival rate and drinking time on barmaid utilisation).

3 Machine learning algorithms used

This section describes two machine learning algorithms used to interpret the results of discrete event simulation. The first algorithm is an attribute-value learning algorithm RETIS and the second is an inductive logic programming system MARKUS.

3.1 RETIS, an attribute-value learning program

Learning from examples where induced theory is represented with attribute-based descriptions is called *attribute-value* learning. RETIS (Karalič 1991) is an attribute-value learning system for the construction of regression trees. Regression trees are similar to binary classification trees in Top Down Induction of Decision Trees (Quinlan 1986), for example trees constructed by the ASSISTANT algorithm (Cestnik Kononenko and Bratko 1987). The most important difference is that, while classification trees are used to classify objects into *discrete* classes, regression trees are used when the class is *continuous*. A regression tree actually implements a function $y(x_1, x_2, \ldots, x_n)$ of n continuous or discrete attributes $x_1, x_2, \ldots, x_n$.

Since in regresion trees the class is continuous, an estimate of standard deviation of the class values is used as the measure of impurity of a node when searching for the best attribute. The best split of examples in a node is therefore the one which minimizes the expected standard deviation.

Once built, a regression tree is interpreted in the following way: each internal node of the tree contains a test on the value of an attribute. According to the result of the test, the interpretation of the tree proceeds to the left or right subtree of the node. A leaf prescribes a value to the function, approximated by the regression tree.

3.2 MARKUS, an Inductive Logic Programming system

Learning from examples where the induced theory is represented by a logic description is called *Inductive Logic Programming* (ILP). MARKUS (Grobelnik 1992) is an ILP system designed for inferring Prolog programs from examples and counterexamples of its behaviour. The MARKUS algorithm has its basis in Shapiro's MODEL INFERENCE SYSTEM (Shapiro 1983), but extends it in several ways.

A Prolog program induced by MARKUS uses background-knowledge predicates defined by the user. Used predicates can be very simple as the "greater-than" (>) predicate or complex procedures written in Prolog.

4 Experiments and results

Experiments were perfomed on seven learning domains (three for the shop and four for the pub simulator). The RETIS (Karalič 1991) program performs *tree pruning* (mechanism for noise handling). RETIS was therefore executed with various settings of the parameters for pruning (in particular, parameter m (Karalič 1991)).

The MARKUS system induced two Prolog clauses about customer utilisation for the supermarket domain.

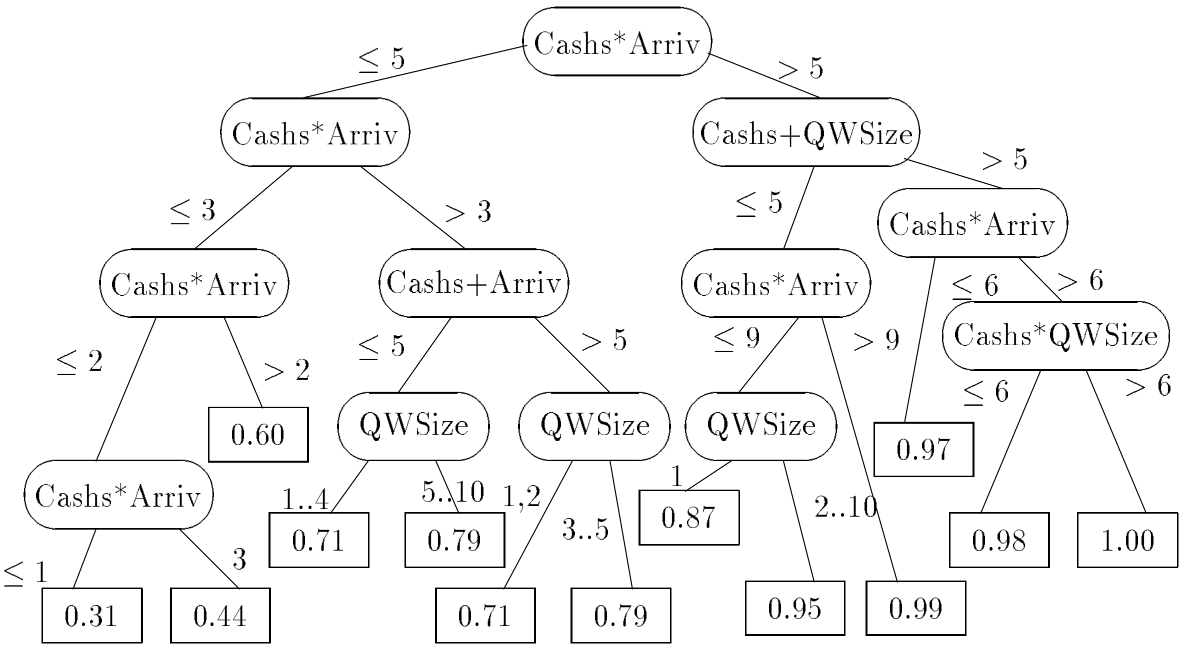

Figure 2: Decision tree for customer utilisation generated by Retis (postpruning with m=2).

As an example of regression trees generated by RETIS, let us consider the tree for customer utilisation (Fig. 2). The tree left-most leaves correspond to `Cashiers` = 1. Customers utilisation here monotonically increases between 0.31 and 0.63. Customer utilisation is under these conditions independent of customers tolerance (`QWSize`). This can easily be explained as follows: for a small number of cashiers and a high arrival rate of customers (attribute `Arrival` small), the waiting queue in the steady state of the system is always long and occasionally drops below `QWSize`. The actual absolute length of the queue therefore doesn't matter and customers utilisation does not depend on `QWSize`. This is shown in the tree of Fig. 2. It can be clearly seen in Fig. 2 that customer utilisation does not depend on `QWSize` in the whole left-hand side of the tree. `QWSize` begins to effect the results for the combinations `Arrival` ≥ 6 or `Arrival` ≥ 2 and `Cashiers` ≥ 3. At higher arrival rates and/or number of cashiers, `QWSize` becomes important. Moreover, it can be concluded from Fig. 2 that the critical value for `QWSize` is 2. The performance is not significantly affected when `QWSize` drops below the threshold 3.

We analysed the simulation results with MARKUS with respect to the question: what combinations of the supermarket attributes will result in customer utilisation below 90%. For this experiment, we used the available 500 simulation results as follows. To avoid problems with small statistical fluctuations, we disregarded the results that were close (within $\pm 5\%$) of the target threshold 90%. Accordingly, we considered a simulation result as an example of low performance if `CustomerUtil` $\leq 85\%$ and a counter example if `CustomerUtil` $> 95\%$. This resulted in 90 positive and 377 negative examples. We defined the following predicates as background knowledge for MARKUS:

```
mult( A, B, C ) :- C is A × B.
gtr1( X ) :- X > 1.
gtr2( X ) :- X > 2.
⋮
gtr9( X ) :- X > 9.
```

The target predicate for learning was: `shop( Cashs, QWSize, Arriv)`, which is true for customer utilisation $< 90\%$. The following two Prolog clauses were induced by MARKUS in a few CPU minutes on a SUN SPARC 1:
shop(Cashs, QWSize, Arriv) :- mult(Arriv, Cashs, X), not gtr5(X).
shop(Cashs, QWSize, Arriv) :- mult(Arriv, Cashs, X), not gtr1(QWSize), not gtr6(X).

These clauses can be read as:
if `Cashs` $\times$ `Arriv` ≤ 5 or `Cashs` $\times$ `Arriv` ≤ 6 and `QWSize` $= 1$
then `CustomerUtil` $\leq 90\%$.
MARKUS nicely discovered the law that holds in situations with high input of customers and low number of cashiers. In the extreme case the flow of customers can be viewed as a continuous liquid flow with the waiting queue never empty. It can be shown that under these conditions, customer utilisation is proportional to the product `Cashs` $\times$ `Arriv`. At higher input rates or higher number of cashiers, the parameter `QWSize` begins to effect the performance which can also be seen in the rule above.

A regression tree was generated by RETIS for the customers wait length in the pub, as a function of the pub attributes: number of barmaids, number of glasses, customer arrival rate and drinking time. In that tree (which we can not include here for space reasons), it was interesting to note that the glasses become an important resource in the case of one barmaid and low arrival rate (`Arrival` between 12 and 17). On average, an increase from 50 to 100 glasses reduces the waitlength from 11.50 to 5.02.

5 Conclusion

In the paper, we experimentally analysed the results of two discrete event simulators by means of machine learning programs. The results obtained by machine learning are encouraging in that the induced descriptions provide some useful insights into the dependences between the parameters of a discrete event system and its performance. Descriptions obtained by the ILP approach can be particularly concise and interesting.

6 Acknowledgements

We thank Aram Karalič for providing a copy of the RETIS program used in the experiments in this paper.

References

[1] Cestnik, B., Kononenko, I. and Bratko, I., (1987), ASSISTANT 86: A Knowledge-Elicitation Tool for Sophisticated Users, in *Progress in Machine Learning*, ed. by I.Bratko and N.Lavrač, Sigma Press, Wilmslow.

[2] Grobelnik, M. (1992) Markus - an optimized Model Inference System, *ECAI-92 Workshop on Logical Approaches to Machine Learning*, Vienna, Avstrija.

[3] Karalič, A. (1991) *Machine Learning of Regression Trees from Imperfect Data* (in Slovene). MSc Thesis, University of Ljubljana, Faculty of Electrical Engineering and Computer Science.

[4] Paul, R.J., Balmer, D.W. (1993) *Simulation Modelling*, Chartwell-Bratt, Stockholm.

[5] Quinlan, J.R. (1986) Induction of Decision Trees, *Machine Learning Journal*, Vol. 1, pp. 81-106.

[6] Shapiro, E.Y. (1983) *Algorithmic Program Debugging.* Cambridge, MA: MIT Press.