# Knowledge acquisition for discrete event systems using machine learning

**Dunja Mladenić,**[1] and **Ivan Bratko**[2] and **Ray J. Paul**[3] and **Marko Grobelnik**[4]

**Abstract.** Knowledge acquisition for an understanding of discrete event simulation systems is a difficult task. Machine Learning has been investigated to help in the knowledge acquisition process. Our approach involves consultation with a domain expert, and the use of discrete event simulation models and machine learning as tools for the intelligent analysis of simulated systems. Current methods for the analysis and interpretation of such systems are restricted to statistical techniques that say much about the reliability of an output, but little about the output inter connectivity. The objective of our work is to improve the ability to interpret the model to the level of explanation that might loosely be described as "How the simulated system works". The new interpretation techniques are based on knowledge acquisition concerning the system using machine learning tools. "How the simulated system works" is important for a decision maker's understanding of the system in terms of relationships between the various parameters, the utilisation of resources and the location of (potential) bottlenecks. This sort of understanding is important when managing queuing problems in production systems, or when planning a new system. The outputs from three discrete event simulators were processed as machine learning problems. In these experiments, the simulation outputs were interpreted by two attribute-based learning programs, namely RETIS and ASSISTANT, and by an Inductive Logic Programming system, namely MARKUS. The knowledge gained from the machine learning tools was evaluated by the domain expert, who considered that this knowledge contributed to an understanding of the system, and how it might profitably be modified.

## 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. Knowledge acquisition for an understanding of discrete event simulation systems is a demanding task. In this paper we propose automatisation of this task by means of machine learning techniques. Our approach involves consultation with a domain expert, the use of discrete event simulation models and the machine learning as an intelligent analysis of the simulated systems (see Bratko, Mozetič, Lavrač 1989). It should be emphasised that the goal of the presented work is not a prediction nor an optimisation. The main goal here is to find interpretations of the simulation output and discovering regularities, thus helping the user to develop an intuitive understanding of the domain (see Fig. 1).

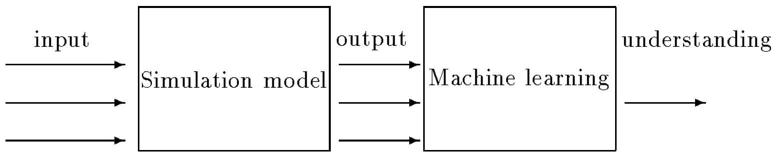


**Figure 1.** Using machine learning for simulation output analysis

Discrete event simulation produces example situations that can be used as input data for machine learning tools. In the presented research, three simple and commonly used discrete event simulators were interpreted using different machine learning tools. The attribute based learning systems RETIS (Karalič 1991, Karalič 1992) and ASSISTANT (Cestnik, Kononenko and Bratko 1987) were chosen as appropriate for learning in noisy, real-world domains. The interpretation obtained by these systems was intuitive but obviously expressed in a complicated way. To enable a more powerful knowledge representation, the Inductive Logic Programming (ILP) system MARKUS (Grobelnik 1992) was used. In addition to compact knowledge representation, MARKUS also highlighted some attribute combinations that could be useful in the attribute based learning.

Section 2 describes the discrete event simulation domains used in our experiments with machine learning. Section 3 gives a brief description of the three machine learning algorithms used to interpret the results from the discrete event simulations. The experiments and the proposed methodology are discussed in Section 4.

## 2 Domain descriptions

Three discrete event simulators: the supermarket, the pub and the steelworks were used in the development of a methodology for knowledge acquisition for discrete event systems by means of the machine learning techniques. The first two simulators are simple and were used in the first phases of the method-

[1] J. Stefan Institute, Ljubljana, Slovenia
[2] Faculty of Electr. Eng. and Computer Sc., Slovenia
[3] Brunel University, Uxbridge, England
[4] J. Stefan Institute, Ljubljana, Slovenia


ECAI 94. *11th European Conference on Artificial Intelligence* Edited by A. Cohn
Published in 1994 by John Wiley & Sons, Ltd.

ology development. They were used with the run-in time parameter set to 50 minutes and the simulation duration set to 1000 minutes, while the third, the steelworks simulator is slightly more complex and was used with the run-in time parameter set to 4000 minutes and the simulation duration set to 10000 minutes (circa 7 days). The simulators were generated using *The Extended Lancaster Simulation Environment eLSE* (Paul and Balmer 1993). The interaction of objects for each example simulator are described by *Activity Cycle Diagrams* (Fig. 2 and Fig. 3). These diagrams use two types of objects: circles to denote queues and rectangles to denote activities. There are some additional conditions incorporated in the simulators that are (for the purpose of simplicity) not included in the diagrams.

## 2.1 The supermarket simulator domain

In the supermarket domain (Fig. 2, left hand graph), 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 process, 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)
- time between two arrivals, `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). Each combination of attribute values corresponds to a run of the simulator producing a machine learning example. In this way, domains for all possible attribute values were generated, producing $5 \times 10 \times 10 = 500$ examples.

## 2.2 The pub simulator domain

In the pub simulator (Fig. 2, right hand graph), 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 beer and wash glasses. A glass can be either clean (and empty), full or dirty. 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)

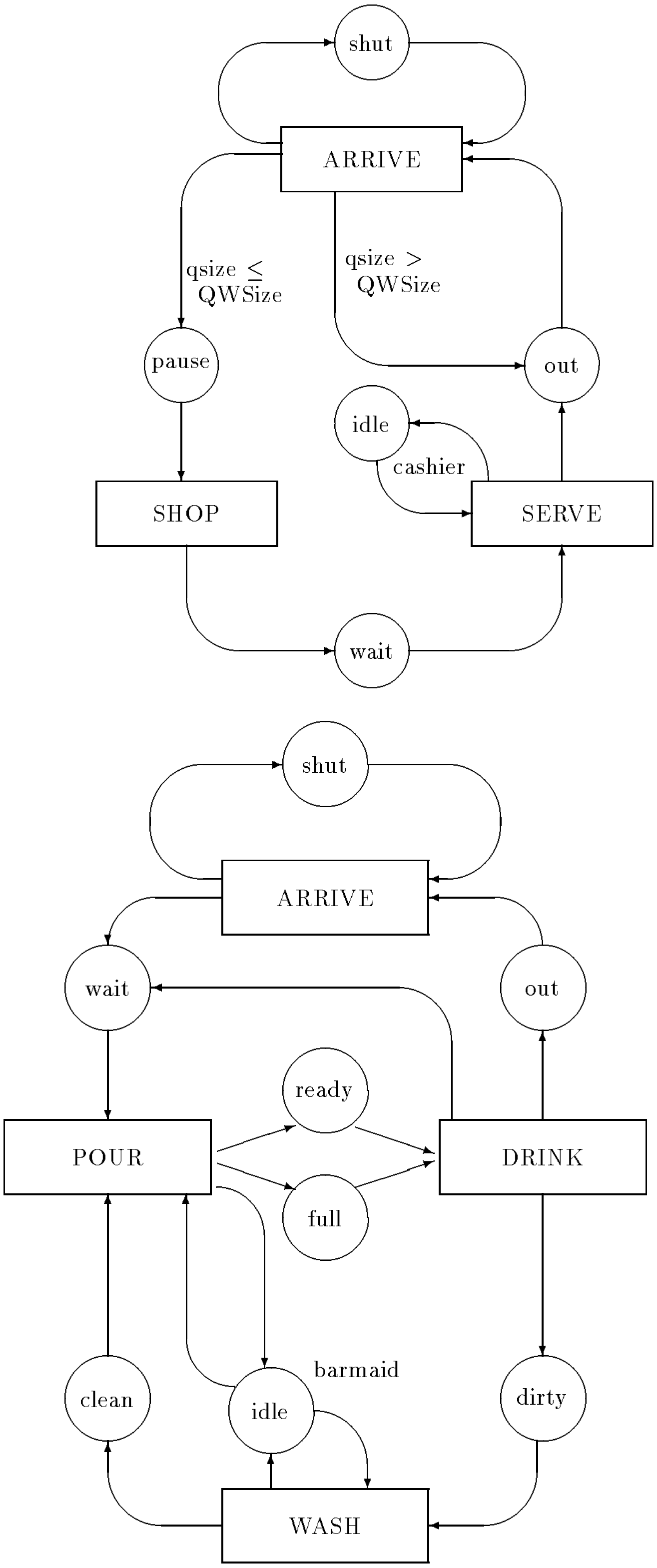


**Figure 2.** Activity Cycle Diagram for the supermarket (up) and the pub (down) simulation

- number of glasses in the pub, `Glasses` (varied: 50, 100, 150)
- time between two arrivals, `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
- length of the customers wait queue
- length of the clean glasses wait queue

From the above parameters and values, three learning problems can be formulated, each investigating the relationship between the above attributes and one of the three class values (e.g. influence of number of barmaids, number of glasses, arrival rate and drinking time on barmaid utilisation). Combining all possible attribute values, domains were generated producing $5 \times 3 \times 13 \times 5 = 975$ examples.

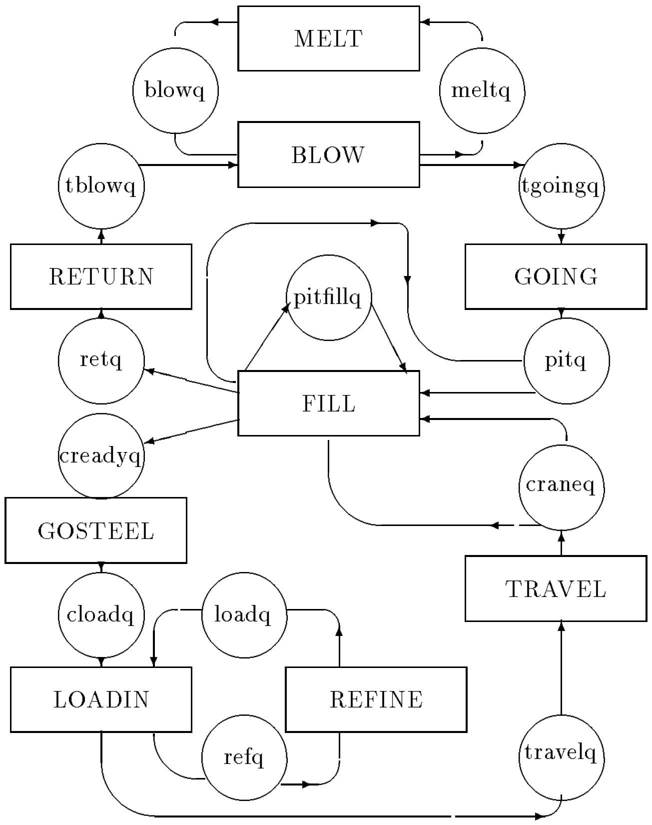


**Figure 3.** Activity Cycle Diagram for the steelworks simulation

## 2.3 The steelworks simulator domain

In the steelworks simulator (Fig. 3) molten iron is cast in one of two blast furnaces. When a blast furnace empties, the molten iron is put into one or more railway trucks called torpedoes (if sufficient are available, currently 2) and transported to an unloading area called a pit. If torpedo is not available, the molten iron is dropped on the floor and waste is produced. At the pit a crane is loaded with exactly enough iron to charge a steel furnace. When a steel furnace is empty and a crane is properly loaded, the crane moves to the steel furnace, loads the steel furnace, and then travels back to the pit. The steel furnace converts the iron to steel after a time, and is then available again. All times in the system are sampled from stochastic distributions, and the transfer of iron is in variable quantities. Whilst ostensibly one might be trying to determine the number of torpedoes and cranes such a system should have, it is in fact more useful to know why such combinations produce the statistical results that they do. There are non-linearities in the system that statistics does not explain, and above average commonsense rarely gets there (about 5% of students, who are bright for their equivalent peer groups, find the cause of the major non-linearity).

The described process is obviously more complex then the previous two, having a higher number of attributes (81 attributes). All the attributes can be grouped as follows:

- number of entities; blast furnaces (set to 2), pit (set to 1), steel furnaces (set to 5), `Cranes` (varied from 1 to 5), `Torpedos` (varied from 1 to 12)
- utilisation of entities; `BlastUtil, TorpUtil, PitUtil, CraneUtil, SteelUtil`
- number of activations of each activity
- nontrivial statistics for time and length of each queue

We considered the occurrence of the `Waste` as interesting for our analysis.
From the above parameters, several learning problems were formulated. We wanted to explain the occurrence of the waste at the following levels: 0, 200, 500, 1000, 1500, 2000, 5000, 10000, 20000, 50000. Combining all interesting inputs to the simulator (values for number of torpedos and cranes), domain was generated having $5 \times 12 = 60$ examples.

# 3 Machine learning algorithms used

This section briefly describes the three machine learning algorithms used to interpret the results of the discrete event simulations. The first two algorithms are attribute based learning algorithms Assistant (Cestnik, Kononenko and Bratko 1987) and Retis (Karalič 1991, Karalič 1992) while the third one is an inductive logic programming system Markus (Grobelnik 1992).

ASSISTANT is an attribute based learning algorithm for the construction of binary classification trees. It belongs to the Top Down Induction of Decision Trees (Quinlan 1986) family of algorithms. RETIS is an attribute based learning algorithm for the construction of regression trees. Regression trees are similar to binary classification trees constructed by the ASSISTANT algorithm. 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 $x_1, x_2, \ldots, x_n$. Both algorithms incorporate several noise handling mechanisms that are used in our experiments.
MARKUS is an Inductive Logic Programming (ILP) system designed for inferring Prolog programs from examples and counterexamples of there 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 such as the "greater-than" (>) predicate or complex procedures written in Prolog.

## 4 Experiments and results

Each discrete event simulator 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). One way to design a machine learning problem for a simulator interpretation is to use user-defined parameters to describe one of the statistical parameters. This approach was successfully used in the first phases of our work, on the supermarket and the pub simulators.
In the knowledge acquisition for the third simulator, our approach was slightly modified. We decided to analyse the steelworks simulator using user-defined and statistical parameters, in the light of waste of melted iron that occurs during the simulation period (circa 7 days). For this purpose, machine learning domains were constructed using a dynamic selection of attributes and class. In the first experiments, all 81 parameters were selected as attributes and `Waste = 0` as the Boolean class. After an analysis of these results and consultation with the domain expert, 31 parameters were selected for attributes and different waste thresholds set for classes (see Section 2.3). Depending on the results of machine learning, a new problem was automatically formulated and the experiments repeated. This approach enabled a more flexible analysis of the simulator from different, dynamically chosen, views. The developed methodology can be described with the following algorithm:

**input:** problem definition (eg. domain with 81 attributes and `Waste = 0` as class)
generate description for chosen problem (eg. classification tree)
**repeat**
  select attribute from description to be class for new problem (eg. attribute in the root of classification tree)
  exclude selected attribute from attribute set
  recursively repeat the whole algorithm on the new problem
**until** all attributes from the description are explained
**output:** set of descriptions (eg. classification trees)

For the supermarket and the pub simulator, independent experiments were performed on six learning domains (three for each simulator). Selected results are presented in the following Sections (4.1, 4.2). The attribute based learning systems that were used, MAGNUS ASSISTANT (Mladenić 1990) and RETIS (Karalič 1991, Karalič 1992) perform *tree pruning* as noise handling mechanism. They were therefore executed with various settings of the parameters for pruning, and some smaller trees were chosen for the analysis.
For the steelworks simulator, experiments were performed using the above described methodology for the analysis of discrete event simulators by means of machine learning tools. The attribute based learning system, MAGNUS ASSISTANT (Mladenić 1990) was adapted for more efficient handling of discrete event simulation data. Results are presented in Section (4.3).

### 4.1 Supermarket simulator

As an example of the sort of regression trees generated by RETIS, let us consider the tree for customer utilisation (Fig. 4). The three left-most leaves correspond to `Cashiers` = 1. Customer utilisation here monotonically increases between 0.31 and 0.63. Customer utilisation under these conditions is independent of customer 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 customer utilisation does not depend on `QWSize`. It can be clearly seen in Fig. 4 that customer utilisation does not depend on `QWSize` in the whole left-hand side of the tree. `QWSize` begins to affect the results for the combinations `Arrival` $\geq 6$ or `Arrival` $\geq 2$ and `Cashiers` $\geq 3$. At lower arrival rates and/or higher number of cashiers, `QWSize` becomes important.
We analysed the simulation results with MARKUS with respect to the question: what combinations of the supermarket attributes will result in customer utilisation below or equal to 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.
add( 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 $\leq 90\%$. The following two Prolog clauses were induced by MARKUS:

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*Arriv` $\leq 5$ or `Cashs*Arriv` $\leq 6$ and `QWSize` $= 1$
**then** `CustomerUtil` $\leq 90\%$.

Markus nicely discovered the law that holds in situations with a high input of customers and a 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 `Cashiers*Arrival`. At lower input rates or for a higher number of cashiers, the parameter `QWSize` begins to affect the performance which can also be seen in the rule above.

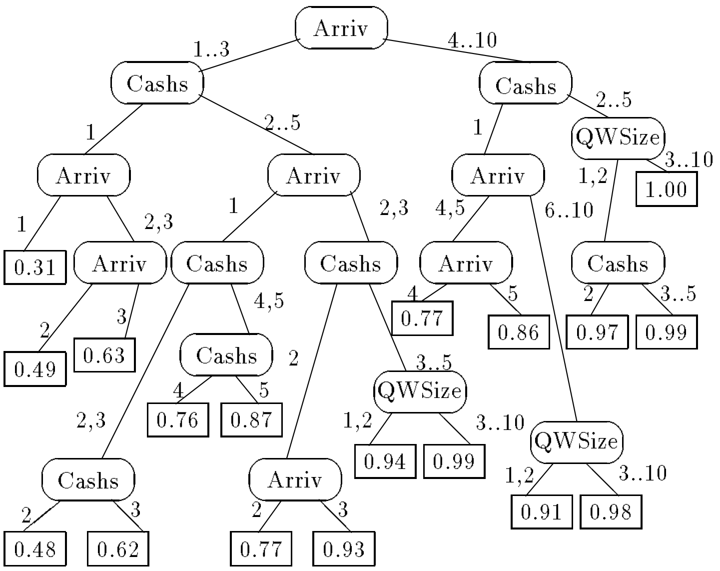


**Figure 4.** Regression tree for customer utilisation generated by Retis (postpruning with m=2).

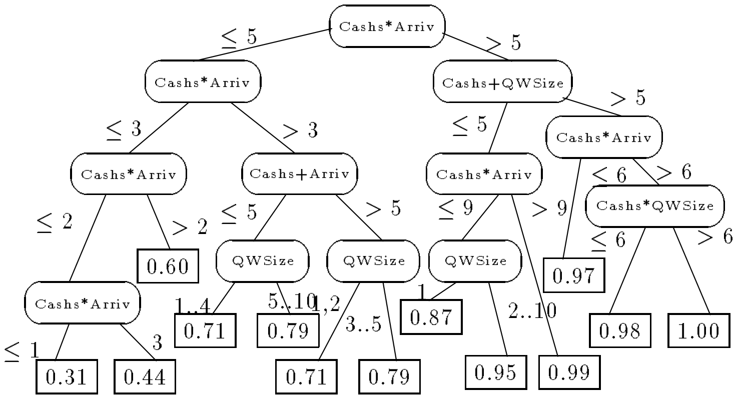


**Figure 5.** Regression tree for customer utilisation (added attributes) generated by Retis (postpruning with m=2).

After seeing the Markus results that highlighted multiplication of attributes, we constructed new attributes (in a similar way as the ILP system Linus (Lavrač, Džeroski and Grobelnik 1990) does) with four basic algebraic operations ($*$, $/$, $+$, $-$) applied to pairs of the original attributes. Experiments with Retis were repeated on such data sets. Additional experiments on the problems reformulated to have Boolean, instead of continuous class, were performed by the Assistant algorithm. Utilisation problems were reformulated into learning when utilisation is below or equal to 90%. The mean length of the customers wait queue was reformulated into the problem of queue length being below or equal to 1 and the mean length of clean glasses wait queue into the problem of queue length being below or equal to 25. Both learning systems confirmed the results obtained by Markus. Moreover, in all three domains for the supermarket simulator, the same combined attribute (`Cashiers*Arrival`) is recognised as the "best" one. Adding attributes in the data, results in a smaller regression tree (the tree in Fig. 5 is smaller than the tree in Fig. 4), under the same pruning conditions. It can be clearly seen in Fig. 5 that the left-most part of the tree (`Cashiers*Arrival` $\leq 3$) is similar to the left-most part of the tree in Fig. 4 (generated without added attributes). Customer utilisation here monotonically increases between 0.31 and 0.60 and is independent of the customers tolerance (`QWSize`). The classification tree generated by the Assistant algorithm on the data with added attributes (see Fig. 6), has the same attribute (`Cashiers*Arrival`) in the root as the regression tree in Fig. 5. It is also interesting to compare the Prolog clauses induced by Markus with the trees constructed by Retis and Assistant. The same attribute with the same threshold 5 (`Cashiers*Arrival` $\leq 5$) appears as the "best" one.

## 4.2 Pub simulator

A regression tree was generated by Retis for the customers wait queue length in the pub, as a function of the pub attributes: number of barmaids, number of glasses, customer arrival rate and drinking time.

The trees in Figures 7 and 8 are generated with additional attributes constructed from the four original attributes with two algebraic operations ($*$, $+$). As in the regression tree generated without additional attributes, the glasses become an important resource (Fig. 7) in the case of a low arrival rate (`Arrival` time between 10 and 15). It is interesting that here glasses are important also in the right-most leaves of the tree (`Barmaids*Arrival` $> 15$). In these leaves the customer wait length $\leq 1$ for the number of glasses $> 75$ or `Barmaids*Arrival` $> 19$. The classification tree generated by Assistant (Fig. 8) show a similar regularity: customer wait length $\leq 1$ for the combinations `Barmaids*Arrival` $> 25$ or `Barmaids*Arrival` $> 20$ and `Glasses*Arrival` $> 500$. It is interesting that in the Assistant tree, glasses are important for $20 <$`Barmaids*Arrival` $\leq 25$. In other situations the answer to the question about customer queue length $\leq 1$ is very clearly determined by the arrival rate and the number of barmaids.

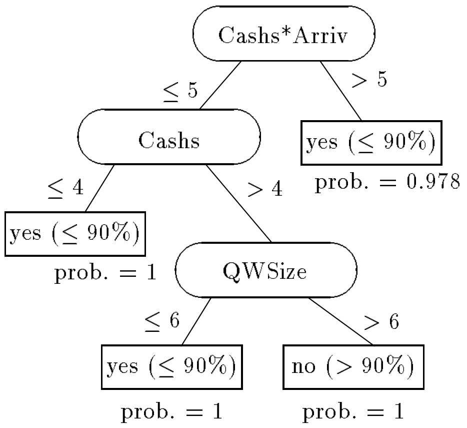


**Figure 6.** Classification tree for customer utilisation (added attributes) generated by Assistant (prepruning: 0,98,0).

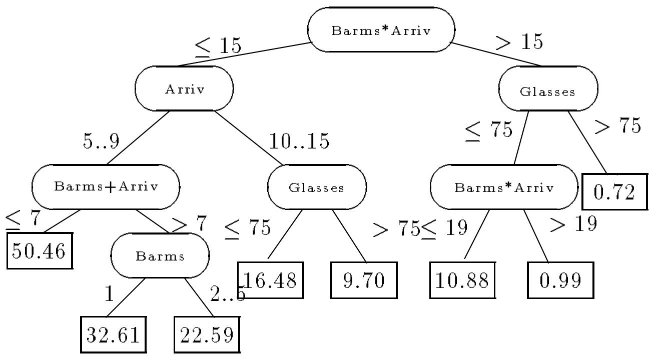


**Figure 7.** Regression tree for length of the customer wait queue (added attributes) generated by Retis (prepruning with m=2).

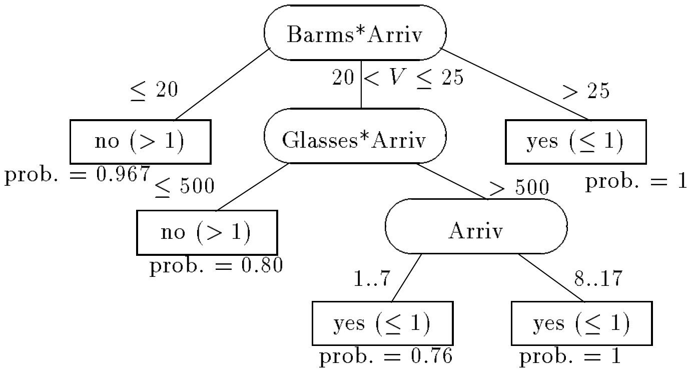


**Figure 8.** Classification tree for length of the customer wait queue (added attributes) generated by Assistant (prepruning: 0,98,4).

## 4.3 Steelworks simulator

There are several interpretations generated for the steelworks system using the same simulator. Some of the typical interpretations generated by the students are as follows:

- The results obtained from the computer model show that operating two cranes instead of one has the most impact in reducing the amount of waste iron. Using two cranes and two torpedoes will result in a 60% reduction in the amount of molten iron wasted in comparison with a single crane and the same number of torpedoes.
- It was felt that in comparing the system with different configurations a two crane system using 6 torpedoes was far superior and is highly recommended. Even when using one crane with 8 torpedoes the system will generate approximately 12% of wastage in comparison with 0% in the case of two cranes.

The main disadvantage of the system interpretations generated without machine learning tools is lack of explanation "How the simulated system works". Provided knowledge hardly goes beyond descriptions concerning mainly optimisation problem (optimal number of cranes and torpedos considering waste). Using machine learning tools for knowledge acquisition enables deeper understanding of the simulated system. In our experiments discrete event simulator produces example situations that were used as input for different machine learning problems. On the same simulator output, several machine learning problems were defined to enable analysis from different aspects. Knowledge generated in a such way provides more complete understanding of the simulated system. This knowledge is currently represented with a set of classification trees (generated by Magnus Assistant) that are very simple even if they exactly match the data (no pruning was used). A strong correlation between attributes enables equivalences of different explanations for the same situation. These equivalences appear in the tree construction as the attributes with equal informativity. For example, the trees generated using

31 attributes and class `Waste = 0` in Fig. 9 are equivalent. Moreover, for each tree node, there are some equivalences:

1. level node (root)
`tblowqTime > 65.64`
`tblowqLen > 2.18`

2. level node
`tblowqTime <= 117.5`
`tblowqLen <= 3.89`

3. level node
`cranes > 3`
`craneUtil < 0.31`
`pitqTime <= 3.9`
`pitqLen <= 0.37`
`craneqTime > 12.54`
`craneqLen > 1.2`

For example, the first equivalence (for the root of the tree) is recognised from the right hand side tree in Fig. 9.

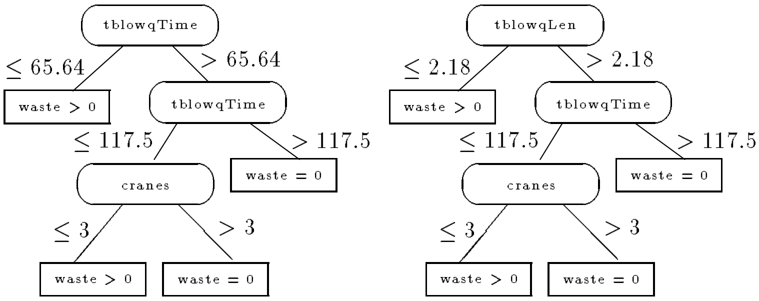


**Figure 9.** Classification trees for waste threshold 0 generated by Assistant.

The left most part of the left hand side tree in Fig. 9, show that waste is greater than 0 if empty torpedos are not waiting long ($\leq$ 65.64). With longer (but still not long enough) waiting time ($\leq$ 117.5) the number of cranes becomes important (`cranes` $\leq$ 3). The right most branch of this classification tree guarantees waste = 0 under the condition of enough empty torpedos (waiting time > 117.5). Equivalent to the torpedo waiting time is, as expected, the torpedo queue length (see list of equivalences above). Domain expert found the equivalences for the number of cranes (or crane utilisation) nontrivial and thus more interesting. The lower number of cranes, the higher their utilisation (`cranes > 3` is equivalent to `craneUtil < 0.31`) and the lower number of cranes, the shorter empty cranes queue (`cranes > 3` is equivalent to `craneqLen > 1.2`) (or waiting time: `craneqTime > 12.54`). In the first moment, domain expert was surprised by the connection of the queue of full torpedos and the number of cranes in the system. Short queue of full torpedos waiting for pit and cranes (`pitqLen <= 0.37`) is equivalent to the greater number of cranes. Originally, short queue of full torpedos indicates fast fluctuation of full torpedos that can appear only if the number of cranes is sufficiently high.
Experiments with waste threshold 200 resulted with similar trees. Attribute `tblowqTime > 65.64` is again the most important and combined with `craneUtil <= 0.61` (higher than in previous problem `Waste = 0`) results with `Waste <= 200`. In all other situations, the waste is predicted to be greater than 200. For both problems, the same attribute with the same threshold `tblowqTime > 65.64` seems to be the most important and its explanation would help in understanding the simulated steelworks. At this point, the developed methodology explained at the beginning of this section is applied. The new problem with class `tblowqTime > 65.64` is considered with the same set of attributes (except itself) and classification tree for it generated. The tree is very simple, having only one node indicating that empty torpedos are waiting more that 65.64 minutes if action of filling crane from torpedo occurs more than 558 times (`tblowqTime > 65.64` if `fillAct > 558`); in the selected simulation period of circa 7 days. There is again number of equivalences:

`fillAct > 558`
`returnAct > 195`
`loadinAct > 375`
`travelAct > 375`
`refineAct > 375`
`pitUtil > 0.46`
`tblowqLen > 1.88`
`pitfillqTime <= 5.69`
`loadqTime <= 14.72`

This explanation can be "chained" with the previous knowledge. Instead of just saying that waste is greater that 0 if empty torpedos are not waiting much (`tblowqTime <= 65.64`), it can be added that in this situation fill activity occurs less than 558 times or return activity occurs less than 195 times etc. (we are considering equivalences for both problems). Using the same methodology, explanations for `tblowqTime > 117.5` and `cranes > 3` were generated (see Fig. 10).

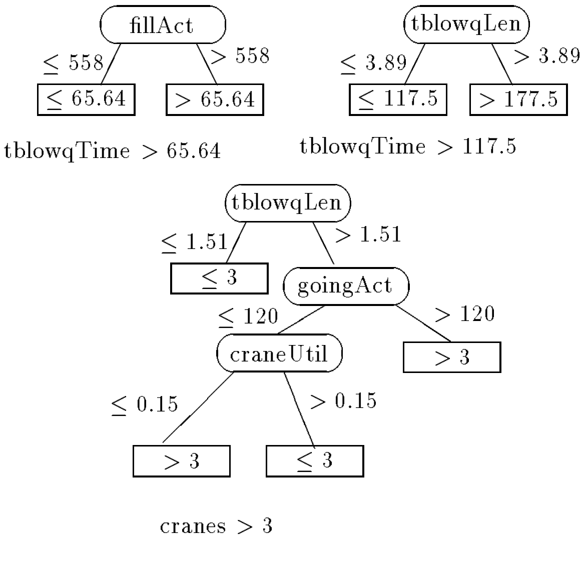


**Figure 10.** Classification trees for `tblowqTime > 65.64`, `tblowqTime > 117.5`, `cranes > 3` generated by Assistant.

## 5 Conclusions

In the paper the machine learning has been investigated to help in the knowledge acquisition process for an understanding of discrete event simulation system. Two attribute based and one Inductive Logic Programming systems were used for the knowledge acquisition on the three simple and commonly used example discrete event simulators. The knowledge gained from the machine learning tools was evaluated by the domain expert, who considered that this knowledge contributed to an understanding of the system, and how it might profitably be modified. This is important because the success of the present study should not be measured in terms of prediction accuracy, but in terms of the interpretative ability of the induced descriptions.

A methodological point of interest is the combination of ILP learning with attribute-based learning, and "chaining" of classification trees. (1) The ILP system MARKUS, could not really cope well with the noise in the learning data, but made important contribution as a generator for new attribute combinations. These were then beneficially applied in the subsequent attribute-based learning. (2) The knowledge provided by classification trees generated for the target problem (waste in steelworks) was analysed by domain expert and decided to be helpful but not enough detailed. Thus, depending on the already generated classification tree, a new target problem was formulated and the experiments repeated. This approach enables a more flexible analysis of the simulator from different, dynamically chosen, views and was considered by domain expert to be very beneficial for the simulated system understanding.

## 6 Acknowledgements

We thank Aram Karalič for providing a copy of the RETIS program used in the experiments in this paper. This work was partially supported by the Slovenian Ministry for Science and Technology, the European ESPRIT III Basic Research Action Project No.6020 Inductive Logic Programming and the British Council in Slovenia.

## REFERENCES

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

[2] 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.

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

[4] Karalič, A., Cestnik, B. (1991), The Bayesian Approach to Tree-Structured Regression. *Proceedings of ITI-91*, Cavtat, Croatia.

[5] Karalič, A. (1992) Employing Linear Regression in Regression Tree Leaves.*Proc. of ECAI 92*, pp. 440-441. Vienna, Austria, John Wiley & Sons.

[6] Lavrač, N., Džeroski, S. and Grobelnik, M. (1991) Learning nonrecursive definitions of relations with LINUS. *Proc. of EWSL 91*, pp. 265-281. Porto, Portugal, Springer-Verlag.

[7] Mladenić, D. (1990) Machine learning system Magnus Assistant (in Slovene). Unpublished *BSc Thesis*, University of Ljubljana, Faculty of Electrical Engineering and Computer Science.

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

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

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