Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

AUTOMATIC INSTANCE GENERATION USING SIMULATION FOR INDUCTIVE LEARNING

Sima Parisay
Behrokh Khoshnevis

Industrial and Systems Engineering Department
University of Southern California
Los Angeles, CA 90089-0193, U. S. A.

ABSTRACT

Inductive learning can be used to extract rules required for an expert system which assists in output analysis for system simulation. However, several expamples of the system, constituting an instance set, are required for learning to take place. Generating the required instance set to be used by an inductive learning algorithm is time consuming and complex. This paper is an attempt to clarify this problem, discuss its complexity, and suggest context related solutions. A procedure for automatic instance generation is then proposed. The proposed procedure is a combination of three search methods (grid based, forward search, backward search).

1 INTRODUCTION

Inductive learning can be used to extract rules required for an expert system which assists in output analysis for system simulation. This idea as applied to queuing systems simulation was discussed in two previous papers, Khoshnevis and Parisay (1993), and Parisay and Khoshnevis (1993). In learning the behavior of a system, several examples of the system's operation are required for learning to take place. In the absence of real examples of a system, such examples may be generated using a simulated model of the system. Generating examples using simulation should be performed systematically. Moreover, it is desirable to automate the process of generating usable examples.

In order to apply the inductive learning algorithm, in this study, a simulation model is considered to have several *control-features*. These control-features may be some of the input parameters, some of the output results, a combination of the two, or some performance data collected during model execution. Examples of such control-features in a queuing system are: mean inter-arrival time, mean service time, mean waiting time in a queue, and system throughput.

To be able to classify some examples of a simulated model, the required *classes* are initially defined. For example, class *goal* is defined for successful simulation runs and class *no-goal* for unsuccessful runs. An *instance* is then a set of specified values for control-features of a simulation model and their related class. Therefore, any change in the values of the model parameters creates a new instance. An instance is represented as a vector of control-features and their values as well as the class to which it belongs. The following is an example of an instance which belongs to class *goal*:

{*inter-arrival time: Exponential(3), service time: Uniform(2,4), ..., throughput: 1.5, class: goal*}

In order to generate an instance the following procedure is performed:

1. A value is selected for each input parameter. This set of input values is called a *test point*.
2. The model is then simulated using the test points. Consequently a set of values are obtained as the output result. This set of output values is called an *output point*.
3. The output point is analyzed and compared with the definition of different classes, then the class that best suits the situation is selected.

In Figure 1, the above procedure is depicted to facilitate later discussions. Here, the collection of all feasible points for input parameters have created the *input space*. Similarly, the set of all obtainable output points as an output result is called the *output space*. The defined classes constitute the *class space*. A simulation process, which is represented as *FS()*, maps a test point to an output point. A class selection criterion, called *class criterion*, which is represented as *FC()*, maps the output point to a class.

To perform an inductive learning, an *instance set* is required. The instance set should have the following characteristics: a) The number of instances should be sufficient for effective learning, b) Each instance should be unique (not repeated). Also, the quality of instance set may be of concern, meaning:

- The instances should appropriately represent the input space, called *input space coverage*.

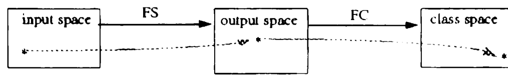

Figure 1: Elements of the Problem

- The number of instances in each class should be suitable for learning, called *class space coverage*.
- The instances should be unique but also represent distinct senarios, called *instance distinction*.

2 INSTANCE GENERATION

The process for generating an instance set is called the *instance generation procedure*. Such procedure should be designed to satisfy the required characteristics of an instance set as mentioned in the previous section. Also, possible time limitations should be considered in the procedure. Otherwise, it may cause premature termination of the procedure (i.e., prior to generation of the desired number of instances.) The uniqueness of instances can be checked while instances are generated. Distinction of instances may be enforced using additional constraints on the degree of closeness of two test points. Input space coverage can be enforced by partitioning the search space by equal size grids or by using Factorial designs (Law and Kelton 1991) when the number of input parameters is large.

The most difficult quality of an instance set to achieve is the class space coverage. As depicted in Figure 1, the class selection depends on class criterion, FC(), and simulation, FS(), which is a stochastic process. Therefore, there is no guarantee that the selected test points will result in a specific class, unless there is a considerable amount of domain knowledge which is generally unlikely to exist. Moreover, there is usually no information about the possible number of instances which belong to any of the identified classes. Each class is defined with an anticipation for the existence of a number of test points which map to it. In practice, however, some classes may never be used in any one of the generated instances.

The class space coverage problem necessitates the use of some kind of search method. Selection of a search method depends on the definition of classes. For example, if there are only two classes as *goal* and *no-goal* then a partitioning (gridding) of input space can be performed. Each test point in a grid element (square, cube, etc.) will either achieve or not acheive the goal, and consequently it will belong to either the class *goal* or the class *no-goal*. If number of instances in any of these classes is not sufficient, a finer gridding can be performed in a smaller area which is promising for obtaining test points from that class.

In this study several classes are considered. The first class is called *goal* and is assigned to a test point that has acheived the goal. The second class is called *no-improvement* and is assigned to a test point that cannot achieve the goal within constraints of the procedure. Other classes are some specified modifications that can be applied to input parameters in order to obtain improved test points (toward the goal). For example, the modification class *Ma* is defined as: increase inter-arrival mean by ten percent.

The output space has unknown characteristics. It is not easy to estimate whether the output's mesh surface is unimodal or multimodal. Search techniques in Operations Research field (Bazaraa and Shetty 1992) are usually successful in cases that meet specific conditions (i.e., continuity, uni-model, and so on). Such conditions cannot easily be proven to exist in our case. The Genetic Algorithm (GA) can be used to find the goal but it cannot indicate the effect of modification classes on each test point which is of importance in this study. However, GA can be used for pre-optimization to find the promising regions and then to apply another search method for fine tuning (Syrjakow and Szczerbicka 1993). Artificial Intelligence (AI) search techniques which are of hill climbing nature suit this study the most. Two heuristic search methods which are used in this study are briefly explained in the following sections.

2.1 Backward Search

In backward search, as illustrated in Figure 2, the attempt is in finding those test points (*i-1*) that can lead to a given point (*i*), when a modification class is applied to them. Therefore, for each modification class an inverse modification, called *inverse modification*, should be defined such that when applied to a given test point *i*, it provides a new test point *i-1*. A new instance is then generated by considering the provided test point, *i-1*, and the modification class. The purpose of backward search is to find test points such that the goal test point can be reached by applying one of the modification classes.

2.2 Forward Search

An AI hill climbing search technique, for example *nearest neighbor*, can be applied to a *no-goal* test point. The purpose of the forward search is to find a path of test points, and a related modification class for each test point, so that the output of any new test point in the path has a better chance of getting closer to the goal.

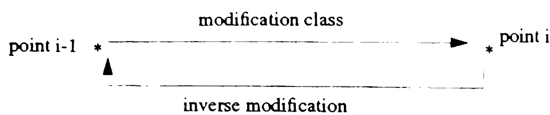

Figure 2: Inverse Modificaiton

3 AN AUTOMATIC INSTANCE GENERATION PROCEDURE

Previous sections attempted to clearify the instance generation problem and instance set requirements. This section proposes a practical automatic instance generation procedure which can potentially fulfill the requirements. The required information for each instance is a test point, an output point, and its related class in the class space. Classes are *goal, no-improvement*, and several modification classes. Automatic instance generation has the following steps:

Step 1:

1-1: Grid the input space as desired.

1-2: Select a test point from each grid and simulate it. If output point has achieved the goal go to step 2; otherwise go to step 3.

Step 2:

2-1: Each test point that its output has achieved the goal is assigned the class *goal* and it is used as an initial point in a backward search.

2-2: The backward search is continued for each recently found test point to arrive at new test points. The modification class which is used to arrive at the new test point is assigned as the test point's class.

2-3: If the required instance set is obtained stop, otherwise go to step 1-2.

step 3:

3-1: Each test point that has not achieved the goal in the first step is used as an initial point in a forward search.

3-2: The forward search is continued until (a) a test point has been reached that achieves the goal, (b) a test point has been reached which cannot be further improved, or (c) a limit on the number of forward iterations has been reached. A *no-improvement* class is assigned to the two later types of test points. Test points that have achieved the goal are assigned the class *goal*.

3-3: The test points in a forward path are used as instances with their related modification class; the leaf will have *goal* or *no-improvement* class. Step 2 is applied to the newly found *goal* test points.

3-4: If the required instance set is obtained, stop; otherwise go to step 1-2.

The maximum number of iterations in backward search and forward search is subjective and can be decided based on each case.

An analysis of this procedure indicates its potential for generating a suitable instance set. The required number of instances in the instance set can be fulfilled by repeating the procedure from step 1 and gridding to as detailed levels as necessary. Before any simulation attempt, each proposed test point is checked with the current instance set for its uniqueness.

Step 1 can be used as a means to ensure the input space coverage of vthe instance set. Class space coverage is fulfilled to some extent by Steps 2 and 3. Step 2 serves to provide useful information (i.e., test points belonging to modification classes) about the area close to the goal as quickly as possible. Step 3 may provide test points belonging to the *no-improvement* class. Even though from the class space coverage point of view it is desired to obtain several test points belonging to the *no-improvement* class, from the learned rules point of view obtaining such test points is not critical. However, it is critical to obtain at least one test point with the class *goal*, in step 1 or 3. A flow chart of the instance generation procedure is presented in Figure 3.

4 CONCLUSION

We have introduced the general problem of automatic instance generation as a means of providing the required instance set for an inductive learning algorithm. The elements (input space, output space, class space, simulation process, and class criterion) of the instance generation process are briefly analyzed to clarify the scope of problems in the instance generation process. An overview of search techniques is mentioned. An automatic instance generation procedure and its resultant instance set are sensitive to the number and the content (definition) of proposed classes. These classes are in fact actions to be taken upon analysis of simulation output. Also, the resultant instance set is sensitive to the nondeterministic nature of the simulation process.

An automatic instance generation procedure is proposed which is an attempt to include the forgoing discussion in relation to our research needs. It is believed that this paper may be used as a base for design of automatic instance generation procedures for different situations. Generated

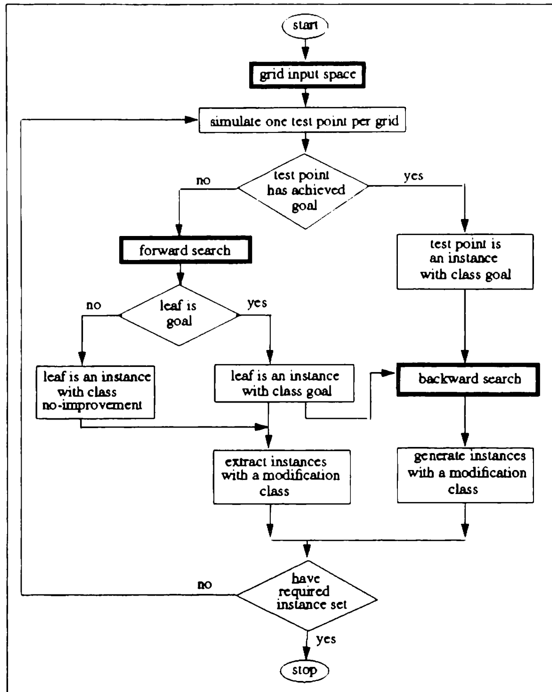

Figure 3: Flow Chart of Instance Generation Procedure

instance sets are specifically valuable in cases where there are no other types of examples of the conditions of the system under study.

REFERENCES

Bazaraa, M. S., and C. M. Shetty. 1979. *Nonlinear Programming Theory and Algorithms.* John Wiley & Sons.

Khoshnevis, B. 1994. *Discrete System Simulation.* McGraw Hill, Inc.

Khoshnevis, B., and S. Parisay. 1993. Machine Learning and Simulation - Application in Queuing Systems. *Simulation* 61:294-302.

Law, A. M., and W. D. Kelton. 1991. *Simulation Modeling & Analysis.* McGraw-Hill, Inc.

Parisay, S., and B. Khoshnevis. 1993. Application of simulation and inductive learning in design of queuing systems. In *Proceedings of the 1993 International Simulation Technology Conference SIMTEC 93*, ed. Martin Dost Consultant, 250-255. The Society for Computer Simulation, San Francisco, California.

Syrjakow, M., and H. Szczerbicka. 1993. REMO - A tool for the automatic optimization of performance models. In *Proceedings of the European Simulation Symposium ESS'93*, eds. A. Verbraeck, and E. Kerckhoffs, 597-603. Delft, Netherland.

AUTHOR BIOGRAPHIES

SIMA PARISAY is a lecturer in Industrial and Manufacturing Engineering Department at California State Polytechnic University, Pomona. She received a B.S. degree in Industrial Engineering from Sharif University of Technology in Iran in 1973. She graduated from Aston University in England with a M.S. degree in Production Engineering in 1975. She is currently completing her Ph.D. degree in Industrial and Systems Engineering from University of Southern California. She has several years of experience in lecturing and consulting. Her research interest is in intelligent simulation experimental design and analysis.

Dr. BEHROKH KHOSHNEVIS is the Director of Manufacturing Engineering Program and is an Associate Professor of industrial and systems engineering at the University of Southern California. He has taught computer simulation for more than ten years to university students and industrial practitioners. He has designed the EZSIM general-purpose simulation software, and has consulted with major software firms in the simulation software design process. Dr. Khoshnevis' primary research interests are in intelligent simulation environments and in automated process planning and concurrent engineering. Dr. Khoshnevis is a member of the Board of Directors of the Society for Computer Simulation. He is a senior member of the Society of Manufacturing Engineers and the Institute of Industrial Engineers. His book on "Discrete Systems Simulation" has been recently published by McGraw Hill, Inc.