A Machine Learning Approach to the Automatic Synthesis of Mechanistic Knowledge for Engineering Decision-Making

Kaihu Chen      Stephen C-Y. Lu
Knowledge-Based Engineering Systems Research Laboratory
Department of Mechanical and Industrial Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

ABSTRACT

Inductive learning is proposed in this paper as a tool for synthesizing domain knowledge from data generated from a model-based simulator. In order to use an inductive engine to generate decision rules, a pre-classification process is necessary in the presence of multiple competing objectives. Instead of relying on a domain expert to perform this pre-classification, a clustering algorithm is used to eliminate the human-bias involved in the selection of a classification function for the pre-classification. It is shown that the use of a clustering algorithm for pre-classification not only further automates the process of knowledge synthesizing, but also improves the quality of the rules generated by the inductive engine.

1 INTRODUCTION

This paper presents an approach of using inductive inference techniques, developed from machine learning research in artificial intelligence (AI), as a computer-based synthesis tool to support engineering decision making. This subject is critical to the future development of a more comprehensive computer-aided engineering (CAE) environment in which computers are used for the engineering analysis and synthesis. The work reported in this paper is based on an intelligent engineering decision framework proposed by Lu [Lu, 1987a]. This framework consists of model-based mechanistic simulations, inductive reasoning, and knowledge-based expert systems to integrate deterministic, heuristic and observational knowledge for engineering tasks. It uses simulations (analysis) to generate information, induction (synthesis) to acquire knowledge from that information, and knowledge-based expert systems, to utilize that induced knowledge.

Engineering decision making requires both analysis and synthesis with proper combinations of the science-based and the experience-based knowledge. Current approaches in computer-aided engineering (CAE) address mainly the analysis phase of decision making, using model-based mechanistic simulations for the science-based knowledge and expert systems for the experience-based knowledge separately. Under this paradigm, there are two major limitations which prohibit computers from being used more effectively in engineering decision making. The first limitation is caused by the missing link between model-based mechanistic simulations and knowledge-based expert systems approaches, which results in a serious gap between the science-based and the experience-based knowledge. The second limitation is due to the lack of computer tools to aid the problem synthesis tasks; as a result, while engineering analysis is highly computerized, engineering synthesis is still heavily based on human experience. To improve engineers' decision-making productivity, there is a pressing need for a new paradigm which can tie mechanistic process simulations with knowledge-based expert systems approaches, and which can address both the analysis and the synthesis phases of engineering decision making. The present paper represents such an effort to respond to the above need.

To understand our approach, the differences between the following three pairs of terms must be clearly defined. First, we defined "analysis" as the process by which specific instances can be predicted from given domain models, whereas "synthesis" is the process which seeks to construct domain models when many specific instances of the system are presented. Second, analysis can be viewed as finding values through a set of given "functions" (forward functions) from a specific set of parameters, whereas synthesis is viewed as the search for "inverse functions" for these analytical functions. Third, analysis always involves deductive reasoning whereas synthesis normally calls for inductive reasoning. Putting the above definition in the context of our present study, a simulation engine represents a body of domain knowledge (e.g., a set of governing equations) that enables us to predict the outcome (or performance) of an operation by feeding control parameters to the simulation engine. Mathematically, a simulation engine can be viewed as a complex function that maps a set of control parameters to their corresponding simulated outcomes. For example, a finite element program (a simulation engine) that is designed for simulating a face milling process can take depth of cut, cutting speed, part/fixture stiffness, etc., as inputs (control
parameters) and then calculates the surface roughness, tool life, material removal rate, etc. as outcome (goal parameters). For the purpose of engineering decision making, it is, however, often necessary to derive an "inverse function" of the simulation engine that maps a simulated outcome to its corresponding control parameters. For instance, in planning for a face milling process, engineers are often given the range of acceptable surface roughness, desired tool life, etc., and are asked to decide on the values of cutting speed, depth of cut, (control parameters) etc., that will satisfy these process requirements.

Knowledge acquisition has been a major problem in the construction of an expert system. The process of interviewing domain experts is time-consuming, and the result is often unreliable. In engineering domains where a simulation engine is available, the knowledge acquisition process can be automated by using a learning program for acquiring knowledge by learning from the examples generated by the simulation engine. In the previous approach [Lu, 1987b], it was necessary to use a domain expert to pre-classify the examples generated by the simulation engine, thus creating a subjective link in the automation of the knowledge acquisition process. This subjective link is eliminated in our approach presented in this paper, with the use of a clustering algorithm that automatically creates classifications for subsequent processing by a learning program that produces generalized decision rules.

In our early works [Lu, 1987b], weighing factors were taken from human experts in those cases to simplify multiple goal interactions for the classification of training examples. In this paper, an enhanced inductive engine that combines the learning from examples and learning from observation paradigms [Michalski, 1985] is used to deal with the problem of goal interaction in a more objective manner. A case study in the domain of face milling processes is reported to demonstrate the effectiveness of this approach.

2 BACKGROUND REVIEW

A framework that combines both computer-aided analysis and computer-aided synthesis for decision making was proposed by Lu [Lu, 1987a]. The main components in this framework consist of a simulation engine, an inductive engine, and a deduction engine. A simulation engine can be a conventional model-based analysis program (e.g., a finite element package) which is built upon mechanistic models of the domain, and is able to predict various system performances (outputs) given different input parameters. A deduction engine can be a typical knowledge-based expert system which contains specific domain knowledge, and is able to assist less-experienced users to reach conclusions deductively through interactive dialogue. One of the unique features of this new paradigm is the use of an inductive engine for the automatic acquisition of domain knowledge through information generated from a simulation engine and data gathered from laboratory experiments. This domain knowledge is then used by the deductive engine for making recommendations. Here, inductive reasoning serves as a bridge for filling the gap between process simulations and expert systems. More importantly, the inductive engine is used as a computer-aided synthesis tool in this paradigm to acquire domain knowledge from process simulations which are mainly analysis-based. It is clear that, within the skeleton of this paradigm, inductive reasoning plays the most critical role.

Although the analytic knowledge of an engineer may enable him to build a useful simulation engine [Fu, DeVor, and Kapoor, 1984], the derivation of an "inverse" engine may be completely intractable. As an alternative approach to the analytic derivation of this inverse function, an inductive engine can be used to synthesize a set of decision rules that jointly represent an approximate inverse function of the simulation engine. In the previous approach [Lu, 1987b], generalization rules are discovered using an inductive engine from pre-classified training examples. This is called "learning from examples" [Michalski, 1985]. There, the result of simulation is pre-classified into several broad classes before the examples are given to the inductive engine in the following form:

\[(\text{control parameters}, \text{class-name})\]

The control parameters of an example can be viewed as the "attributes" that describes an instance of class-name. With a number of such examples belonging to different classes, an inductive engine can be used to produce generalized descriptions for each class in the form of a set of decision rules of the form:

\[\text{conditions} \rightarrow \text{class-i} \]

which indicates that if the control parameters of an instance satisfies the conditions at left, the instance is expected to have result as indicates by class-i. For the purpose of engineering decision making, these rules can be used in the inverse direction in order to specify the conditions that will lead to desired results. The knowledge for pre-classification (i.e., grouping events before the application of inductive engine) can come directly from the domain (e.g., when there is a single goal), or through the meta-knowledge obtained from human domain experts (such as the weights assigned to competing goals).

Difficulties arises when the human expert is unable to provide meaningful pre-classification knowledge due to either the complexity of the problem, or simply because the expert does not have sufficient knowledge about the domain. For example, in the machining domain each instance is measured by a set of interacting goals, such as the material removal rate, tool life, and surface roughness. A high material removal rate will often lead to a bad surface roughness and a shorter tool life. To achieve a good finished surface, the material removal rate needs to be reduced. In the previous approach, the competing nature of these multiple subgoals complicates the pre-classification that is required before an inductive engine can be used on the data generated by the simulation engine.
Another approach to solving this problem is through the use of a clustering algorithm that automatically creates classifications for the subsequent inductive engine. A clustering algorithm generates descriptions of clusters in the space of objective attributes (the output parts of the training instances). A cluster represents a collection of examples that are deemed "close" (according to the clustering criteria used) to each other. With the use of a clustering algorithm, the examples generated by the simulation engine can be clustered with respect to the result of the simulation. Note that only the parameters objective attributes are used for clustering, so that classes can be created in the objective space.

The clustering algorithm automates the process of defining a finite number of classes from a set of multiple competing goals that is represented in a continuous space. It can be shown that, as opposed to using a random or subjective classifier (as is the case when a human expert does not have sufficient knowledge about the problem), our approach is likely to create decision rules of higher precision.

3 METHODOLOGY

Learning from examples and learning from observations are the two main paradigms in current machine learning research. In the paradigm of learning from examples (also called supervised learning), a learning program produces generalized descriptions of examples and counter-examples of a given class. Learning from examples is often used as a tool for data compression where large amount of pre-classified instances can be condensed to a small number of decision rules. Since each example given to such a learning program must bear a label indicating the specific class it belongs to, a "teacher" is needed for this pre-classification (deciding if the example in question is an "positive" or "negative" instance to the class). In learning from observations, also called unsupervised learning, the learning program is not given information such as the class which an example belongs to. Classification or taxonomization are generated by the learning program according to the clustering criteria (e.g., a "compact" cluster is preferred over a "sparse" cluster) provided by the user. Conceptual clustering, a form of learning from observation, creates a hierarchy of classes in which each class is described by a single conjunctive concept [Stepp, 1986].

The combination of these two types of learning paradigms permits us to objectively synthesize engineering knowledge from information for a domain with multiple competing goals. This can be achieved in two steps. First, a classification are created with the use of a clustering algorithm on the objective attributes of the examples generated from a simulation engine. Second, an inductive engine that learns from classified examples are used to discover generalized descriptions (decision rules) for each class. The combined effect of these two steps is to create a qualitative mapping from a multi-dimensional control space to a multi-dimensional objective space. Note that the mapping between the control space and the objective space is created by an inductive learning program such as AQ15. Applying clustering on all attributes (both control attributes and objective attributes) alone does not create such mapping.

The process automates the synthesis of engineering knowledge embodied in the simulation engine, and bridges the knowledge transfer bottleneck between model-based simulation and knowledge-based expert system in the engineering domain.

4 A CASE STUDY

To understand the above methodology in synthesizing engineering knowledge, a case study of face milling processes is presented. We start by describing some basics of this process and a simulation program built for this process. We then explain how clustering and induction are used to generate knowledge for operation planning of that process.

4.1 Face Milling Process and Its Mechanistic Simulations

Face milling is a machining operation used to remove layers of material from a part surface. In performing operations planning for a cutting operation, a planner needs to decide on what would be the proper specifications of process parameters (control attributes) such as feed rate, depth of cut, number of inserts, cutter offset, etc. that would result in a "good" cut which meets performance criteria (objective attributes) such as surface roughness, production rate, tool life, etc. Some part design parameters, such as the structural stiffness of the part/fixture combination, also play an important role in this decision. Information found from machining data handbooks is often not sufficient for deciding these operational parameters. Engineers often have to rely on their experience to make their own decisions and/or to modify what is suggested by handbooks. Furthermore, operational parameters change greatly when part geometries, materials, and cutter types change. It is very difficult, if not impossible, for engineers to develop different heuristics to cover all the possible cases of the process. It is clear that the traditional way of building expert systems, which relies on manual interviews of domain expert and heuristic knowledge, is not appropriate for this operation planning task.

A simulation program which is based on mechanic models of the process was developed for face milling operations [Fu, et al, 1984]. In our case study, training examples are generated using this simulation engine. Each training example is defined in terms of five control attributes and four objective attributes. Given a large set of training examples obtained from the simulation engine, the goal here is to generate a relatively small number of generalized decision rules that collectively represent approximately an "inverse function" of the simulation engine. With these decision rules, one can decide the proper control parameters to be used in order to generate desired results.
set of objective attributes).

This problem has been partially solved with the use of an inductive program, AQ15 [Lu, 1987b]. In the previous approach, the objective attributes of each training example are combined using a set of weighting factors \( w_i \) to form an objective function \( f \). Each training example is defined by a tuple of five control attributes and four objective attributes

\[
< c_1, c_2, c_3, c_4, c_5, o_1, o_2, o_3, o_4 >
\]

This can be transformed into the following form using an objective function \( f \)

\[
< c_1, c_2, c_3, c_4, c_5, class >
\]

where \( class = f(o_1, o_2, o_3, o_4) \). The new attribute "class" has a finite number of values, and is used as a class designator for a training example. With training examples transformed this way, AQ15 [Michalski and Larson, 1978] was then used to find generalized discriminant descriptions (rules) for each class. The use of an objective function transform a multi-dimensional objective space into a one dimensional space, thus permitting

\[
\text{AQl5} \quad \text{to find generalized discriminant descriptions (rules) for each class.}
\]

4.2 A Different Classification Scheme

In this domain, the objective attributes represent a set of competing and interacting goals where optimum tradeoff needs to be made. In the previous approach, the objective function was constructed based on an expert's general knowledge of the domain to specify weighing factors for each competing goal. Since these weighting factors could be viewed as expert's meta-knowledge about the process, they are biased toward individual experience and preference. There was no guidance as to how "better" rules with greater precision might be generated with the use of alternative objective function. Here we define the "precision" of a rule as the variance of the predicted result. The precision of \( R \) can be described as

\[
\sum_{i=1}^{N} \frac{(y_i - \bar{y})^2}{N - 1}
\]

where \( \bar{y} \) is the center of the right-hand side of a rule, and \( y_i \) is the right-hand side of a test example (total \( N \) test examples) whose left-hand side satisfies the conditions of the rule.

In the earlier approach, an objective function of the following form was used:

\[
f(O_1, O_2) \equiv w_1 + w_1 + w_2 + O_3 \\
\text{if } f(O_1, O_2) \leq g_1, \text{ then class } = A \\
\text{if } g_1 < f(O_1, O_2) \leq g_2, \text{ then class } = B \\
\text{if } g_2 < f(O_1, O_2) \leq g_3, \text{ then class } = C \\
\text{if } g_3 < f(O_1, O_2), \text{ then class } = D
\]

Combining objective attributes this way is in effect classifying the examples in the objective space based on a set of parallel bands. \( W \) decide the slope of these bands, and \( g_i \) (human assigned values) decide on the size of these bands. It is obvious that this is a rather artificial way of classifying the examples in the objective space, since it is likely to be effective only on special circumstances when the training examples actually cluster into parallel bands in the objective space.

How does the use of clustering help in improving the precision of induced rules? Consider an example where each training example is described with two control attributes \( C_1 \) and \( C_2 \) and two objective attributes \( O_1 \) and \( O_2 \). An arbitrary classifier \( f \) may decide all examples that fall into an area \( K \) in the objective space are to be classified as class \( A \). A generalized description \( H \) can be found for class \( A \) in the control space using an inductive algorithm. This can be given as the following rule:

\[
H \rightarrow K
\]

Given \( N \) training examples, each example \( E_i \) described with two control attributes \( C_1 \) and \( C_2 \) that fall within \( H \), the precision of a rule is thus decided by the "compactness" of the events within \( K \) in the objective space. That is, the precision of a rule tends to be greater if events within \( K \) tend to concentrate around the center of \( K \). On the other hand, the precision of a rule would be lower if the events are scattered around the boundary of \( K \). This leads to the conclusion that in order to discover generalized rules with higher precision, the training events should be classified in term of their natural clustering in the objective attribute space (O space).

The problem is solved in two steps. First, discover clusters of training examples in the O space, then use these clusters as a classifier to assign a class label to each training instance. A program CLUSTER/2 developed by [Stepp, 1984, 1988] is used for this purpose. Given the examples classified using the clusters generated by CLUSTER/2, the second step uses an inductive program AQ15 to find generalized discriminant descriptions for each class.

4.3 Pre-Classification

Each example is described in terms of five control attributes: feed rate (FR), depth of cut (DC), number of inserts (NI), cutter offset (CO), and part/fixture stiffness (PS); and four objective attributes: average Y-direction cutting force (AVEFY), maximum Y-direction cutting force (MAXFY), maximum Z-direction displacement (MAXDZ), and material removal rate (MRR). These four objective attributes are chosen because they are all related to some measurements of the goodness of an milling operations. For example, the Y-direction cutting forces are related to the tool life, Z-direction displacement is related to the finish surface roughness, and the material removal rate is related to the production efficiency.
To generate a simple classification from the four objective attributes, each training example is given to CLUSTER/2 with only its four objective attributes: AVEFY, MAXFY, MAXDZ, and MRR. CLUSTER/2 is instructed to generate between 2 to 10 clusters based on a "sparseness" measurement. It returns the best clustering it can find in the categories of 2 classes, 3 classes, \ldots, 10 classes. A cluster, to be used as the definition of a "class" for subsequent processing, takes the form of a conjunctive expression. Although the experiment shows that two to five classes categories yield slightly more compact class, we have chosen the six-class categories because the clusters are also relatively compact, and the number of classes is more likely to produce rules of higher precision.

4.4 GENERATING AND USING DECISION RULES

The six classes generated by CLUSTER/2 are used as a class designator for each of the 243 training examples. For example, if a training example whose four objective attributes fall within the boundaries defined by class 2, then it is classified as an instance of class 2. The 243 examples are thus classified into six classes with the classifications generated by CLUSTER/2. AQ15 is then used to generate generalized description for each of these six class. The decision rules generated by the inductive engine represent a qualitative version of the simulation engine. The generalization offers several advantages:

- Only a relatively small set of rules (compared to the number of entries in an engineering table) is needed. This not only greatly reduces the amount of memory needed to store the knowledge, but also speeds up the retrieval process.
- Solutions may still be found from the decision rules even for situations that are not covered in the training examples.

Given a certain goals in mind, an engineer looks for those decision rules whose right-hand side covers his goals. The left-hand side of these rules are the solutions that will achieve the goals.

4.5 Evaluation of Results

There is a tradeoff between the number of classes (the number of clusters in the objective space) and the generality of the result. The precision of the resulting rules improves as more classes (i.e., clusters) are used. In the extreme case, each individual example can be considered as a class in itself, and the rules degenerate to the form of a table that catalogues unprocessed experimental results. When fewer classes are used, the precision of the rules generally degrades. This is due to the fact that instead of mapping a point in the control space to another point in the objective space, the rules now represent a less precise mapping of two hyper-rectangles between the two spaces. This degradation in precision is compensated by improvements in other ways. First, a smaller number of rules is produced, so that the memory and computing time required to store and use these rules are reduced. Second, using fewer classes leads to more general rules (i.e., the size of the hyper-rectangle is bigger), which permit us to predict the behavior of the system in part of the control/objective space where no training example is present. The balance between these competing factors is something that must be decided by user's requirements.

5 CONCLUSIONS

This paper illustrated a way of integrating AI techniques, namely machine learning and expert systems, with traditional engineering methods, namely model-based mechanistic simulations. The contributions of our approach are of two-fold. First, we demonstrated that inductive inference developed from AI research can be used as a computer-assisted knowledge synthesis tool. This opens a new dimension for present research in computer-aided engineering which mainly focuses on computer-based analysis. With this synthesis component added to the new decision-making framework we proposed, computers can be used as knowledge generators rather than just information collectors.

Second, we employed machine learning approaches as automatic knowledge acquisition tools for building engineering expert systems. This eliminates a major bottleneck of the present approach in which much engineering knowledge included in mechanistic models can not be directly used by expert systems. Our approach allows engineers to build expert systems on top of, rather than instead of, what they are currently doing best - simulations.

Our knowledge of an engineering domain is often manifested in the form of a simulation engine. By taking advantage of the analytic knowledge embodied in a simulation engine, we have shown that the gap between this important kind of knowledge source and an expert system can be bridged using inductive inference.

The use of model-based simulation engine as sample generators for learning programs has several advantages. First, the burden in off-line data collection is greatly alleviated. As opposed to collecting data manually, using a simulation engine speeds up the process of data collection. Second, with the use of a simulation engine, errors that originated in the process of data measuring and data entry may be avoided. Third, a simulation engine may be used as an interactive data source for inductive engines that are capable of demanding a certain type of training examples that may be most beneficial to the learning process. This inclusion of an automatic sample generator into an induction engine opens many new opportunities for researchers in machine learning to develop more robust and effective techniques.
6 ACKNOWLEDGEMENTS

Fundings of this research are provided by Bendix Corporation at Kansas City, Digital Equipment Corporation, and the National Science Foundation. The AQ15 algorithm was developed by Professor R.S. Michalski, the CLUSTER/2 program was developed by Professor R.E. Stepp, and the face-milling simulator was developed by Professor R.E. DeVor, all of the University of Illinois at Urbana-Champaign. Their efforts, which made this study possible, are greatly appreciated.

References


