An Evolutionary Approach to the Induction of Process Model from Continuous Data

Javier Nicolás Sánchez
Computer Science Department
Stanford University, CA 94305
jsanchez@cs.stanford.edu
http://www-cs-students.stanford.edu/~jsanchez/

Abstract: Process model induction is a new research problem for machine learning that has been recently proposed. One of the biggest challenges of that problem is the huge search space that it defines, since it is a problem that requires simultaneous optimization of structure and parameters of models. Thus, algorithms for this problem need to address accuracy of the models as well as finding results with a reasonable computational effort. In this paper I propose a new algorithm for the induction of process models that applies genetic algorithm techniques to solve it. The proposed approach draws on the power of genetic algorithms for search in high dimensional spaces, resulting in an algorithm that shows competitive results and that seems to scale gracefully with the size of the problem. Experiments with two synthetic data sets are presented showing results obtained when inducing process models from population dynamics. In closing I discuss possible extensions and areas for future research on this technique.

1 Introduction and Motivation

In recent year we have seen an increasing availability of large amounts of data from scientific and engineering domains involving continuous variables that change over time. Many of those domains lack precise models that describe the interactions between those variables, thus the application of induction methods on that data holds obvious benefits. As a result, inducing models from data is a field of active research with a large literature body on computational methods for regression and time-series prediction.

However, those techniques emphasize the predictive accuracy, while paying little attention to the communicability of the learned knowledge. As Pazzani et al. (2001) have shown, experts in some domains will reject a learning system's output, even when it is accurate, unless it makes contact with their prior knowledge. To address that issue, the task of process model induction has been recently proposed in Langley et al. (2002). Its goal is to learn a model that accurately predicts the data using concepts that are commonly used in the domain where the induction is taking place. Consequently, the result of the induction task is a model that can be easily interpreted by domain scientists and engineers. Moreover, since those models are cast in terms of processes they have the expressive power to explain what is happening in the system, which as Cartier et al. (2001) notes, is a fundamental purpose of models in science.

Therefore, research on algorithms to induce process models from continuous data configures an interesting problem to be addressed by the machine learning community. However, since it is in its early stages, little research has been done on efficient algorithms to address this task. Currently the only existing method for induction of those models is the one proposed in Lagley et al., which performs an exhaustive search in the space of possible model structures. In order to make the search problem more tractable, the algorithm uses constraints on the size of allowable process model structures as well as other constraints drawn from background knowledge for the particular domain.

Nevertheless, the performance of the algorithm that has been proposed is very sensitive to a change in any of the parameters that define the size of the search space, such as number of variables being predicted, maximum size of the allowable model structure, or the number of generic processes being considered.

Furthermore, as noted in the previously mentioned work, the induction algorithm lacks heuristics for avoiding over-fitting of models and to guide the search to deal with increasingly bigger search spaces.

Genetic algorithms are an obvious choice to be considered for this problem since the search space where we need to work has high dimensionality and the different model structures are related in highly non-linear ways, two properties that usually characterize areas of successful application of evolutionary algorithms. Furthermore, genetic algorithms can provide an elegant approach to the task of discovering a model structure whose size is not known in advance without greatly affecting the performance of the search procedure, since the evolutionary process can be used to identify that size. Finally, genetic algorithms have shown better properties when the size of the search space grows, something that is highly desirable in the process model induction task.

Consequently, in this paper I explore an alternative approach for the task of process model induction based on the application of genetic algorithm techniques. In particular, I focus this work on proposing a new algorithm to search through the space of possible structural models, since that is the most novel problem that appears with the induction of models. Furthermore, focusing this work on the search through the space of possible models allows a more clear comparison between the previous approach and the one proposed here, since both approaches use the same type of numerical optimization techniques to fit the parameters of the candidate models.

The remainder of this paper is organized in six more sections. The following section gives an overview of the process modeling formalism used to encode the domain knowledge that is used to induce the process models. Section 3 presents the algorithm for model induction and how it was implemented. In Section 4 the results of different experiments with synthetic data sets are presented, and those results are discussed and interpreted in Section 5. Finally, Section 6 presents the conclusions of this research work, and Section 7 closes with directions for future research in this area.

2 The Process Modeling Formalism

In this section I will briefly introduce the process modeling language presented in Langley et al. (2002), which is the notation used to introduce background knowledge and to represent the results of the induction process in this work.

The process modeling language encodes models in terms of two concepts, processes and variables. The processes describe one or more causal relations between input and output variables. Those relations can be stated in terms of differential equations (for processes that involve change over time) or static equations (for processes that involve instantaneous effects). Furthermore, a process may also include conditions, stated as threshold tests on its input variables, that describe when it is active. Consequently, a process model consists of a set of processes that link observable input variables with observable output variables, possibly through unobserved theoretical terms.

Processes in this modeling language can be of two types, generic processes and specific processes. Generic processes do not commit to specific variables or parameter values, thus they serve as templates from which specific processes can be derived. Therefore, specific processes are identical in structure to generic processes but they have been assigned specific variables and values to their parameters, thus providing a process whose effect can be numerically evaluated.

Table 1 presents an example of the encoding in the modeling language of a Holling's type II predator-prey relationship from the domain of population dynamics. The language constructs are self-explanatory, being the "variables" clause used to define the variables that appear in the model, and the "observable" clause used to determine which of the variables can be actually observed in the data. Equations are expressed in a notation similar to that used in mathematical software, though I should clarify the notational convention `d[prey,t,1]`, which denotes the first derivative of prey with respect to `t` (time).

Examples of generic processes can be found in Table 2, which shows the library of generic processes used in the experiments described later in this paper. One of the differences that appear in declaring a generic process is that the variables involved in the process are associated with a type, indicated between curly braces, which specifies a constraint that restricts the applicability of that generic process to variables of that particular type. The other difference from the specific processes previously introduced is the presence of intervals such as `[0,50,100]`, which encode bounded constants that constitute the parameters to be fitted for that process. The leftmost number in the interval denotes the lower bound on the value that the constant can have, and the rightmost number is the upper bound. The middle value is the initial guess for the constant, which in this case is the middle of the interval.

Finally, to close this brief discussion on the modeling formalism, one more aspect of the language should be clarified, which is the assumed additive effect of processes. By convention the modeling language assumes that processes affecting a common variable have additive effect over it, that is, their equations are added, something that is extensively used in the models presented in this work.

3 A Genetic Algorithm for Inducing Process Models

The task of process model induction involves discovering a set of specific processes that accurately predicts the observed data. In order to accomplish that, we need to determine a set of generic processes, the unification of variables that need to be done on them, and finally the specific parameters of those processes.

As can be easily seen, the definition of the problem presents an enormous search space. However, as proposed in Langley et al., we can apply a divide and conquer approach and split the search in two levels. A first level, where we search for model structures, i.e. models composed of partially instantiated generic processes whose variables have been unified, but whose parameters still remain unspecified. And a second level, where we perform a constrained optimization of the parameters involved in each particular structure.

Representation Scheme

The goal of this research work is to propose an alternative algorithm for the first level of the search, which is the exploration of the space of possible model structures. Thus, the building blocks to be considered when encoding the problem are the partial instantiation of generic processes, i.e. those processes for which we have established the unification of their variables but we have not yet set a value of their parameters.

Since we know the number of variables that we are trying to predict, as well as the number of generic processes that we need to consider, we can perform an exhaustive enumeration of all the possible partial instantiations of the generic processes. Notice that the size of this enumeration is not prohibitively big, and in fact a search through the space of models should be able to consider a model that eventually includes all those processes. Having enumerated all the partial instantiations, we assign a bit position in the binary string to each of those partially instantiated generic processes, and interpret a bit position set to 1 as the presence of that process, and a 0 as its absence. Consequently, every possible structure in the entire search space can be encoded with this representation, since every possible partial process model that can appear in a model structure has a different bit position.

Figure 1 illustrates how the encoding mechanism works for two sample individuals when considering the generic processes presented in Table 2 and two variables to be predicted called predator and prey. As can be easily seen, individual 1 encodes a process model that has the structure of the concrete model shown in Table 1. Another example is presented with individual 2, which encodes a model composed of a logistic growth process for predators, an exponential decay of preys, consumption of preys by predators, and a constant inflow of preys.

Operators

The operators used in evolving the solutions are the traditional reproduction, mutation, and crossover on binary strings. However, the use of those operators can create invalid individuals, i.e. individuals that do not contain processes to predict all the variables on which the induction is taking place. This problem arises from not having in the representation any constraint that prevents the operators from producing an individual with zeros in all the positions corresponding to the processes that affect a given variable.

The solution of that problem requires the introduction of a repair operation that is applied when an invalid individual is produced as the result of the application of mutation or crossover. The repair technique creates a new valid individual by setting to 1 the bits of randomly chosen processes, which are selected from a pool of processes that determine the variables that where not predicted by the model encoded in the failed individual.

Figure 1. An example of the encoding of structural models for a two variable case using the generic processes from Table 2

Bit		Individual 1	Individual 2
0	exponential_growth_predator	0	0
1	exponential_growth_prey	1	0
2	logistic_growth_predator	0	1
3	logistic_growth_prey	0	0
4	exponential_decay_predator	1	0
5	exponential_decay_prey	0	1
6	consumption_predator_predator	0	0
7	consumption_predator_prey	1	1
8	consumption_prey_prey	0	0
9	consumption_prey_predator	0	0
10	constant_inflow_predator	0	0
11	constant_inflow_prey	0	1

Thus, the repair operation applies a criterion of minimal change to obtain a valid individual, something of great importance to preserve the properties of the genetic search.

Seeding Strategy

Another important aspect of the algorithm is the seeding strategy used to create the initial population. Here again we are faced with the problem of creating individuals that represent valid partial models. The seeding procedure implemented randomly chooses a bit string for the individual and then applies the repair operation on it if it is needed. Thus, all the individuals appearing in the initial population are valid and randomly chosen from among the valid models, since there is no bias on the repair operation.

Fitness Evaluation

The fitness evaluation is the most complex and expensive operation in the algorithm being proposed, since the type of models analyzed are expressed in terms of systems of differential equations. In particular, to evaluate the fitness of a given individual we need to follow two steps: first, fit the parameters of the encoded structural model, and then evaluate that concrete processes model to determine its fitness.

Since it is a goal of this project to be able to draw a comparison with the previous work done in model induction, we needed to use the same fitting mechanism previously used. However, this proved to be a challenging problem since LAGRAMGE (Todorovski & Dzeroski, 1997), the system previously used for parameter fitting, requires a costly initialization procedure. This was not a problem in the induction algorithm proposed in previous work because the search space to be explored could be encoded so as to require only one invocation of the discovery system. However, the nature of the genetic algorithm requires at least one invocation for every generation. Thus, an alternative procedure had to be used. Fortunately, the method used by LAGRAMGE to perform the constrained optimization of models is implemented using a freely available numerical software library. Thus, while in the implementation of the algorithm LAGRAMGE was no used to fit the parameters of the models, the results obtained are exactly the same, thus effectively providing a method for fair comparison between the two induction algorithms. The parameter optimization proceeds by decoding the model represented in the chromosome and then feeding it into an optimization function that determines the best values for the parameters of the processes involved.

Table 1. A process model of a two-species species predator-prey relation

```
model predprey_holling;

variables pred,prey;
observable pred,prey;

process pred_prey_consumption;
equations
  d[prey,t,1] = - pred * prey / (0.1 * prey + 1);
  d[pred,t,1] = 0.5 * pred * prey / (0.1 * prey + 1);

process prey_logistic_growth;
equations
  d[prey,t,1] = 2 * prey * (1 - prey / 1.4);

process pred_exponential_decay;
equations
  d[pred,t,1] = -0.1 * pred;
```

The second stage of the fitness evaluation involves determining how fit is the concrete model encoded in the individual. Since the goal of the induction process is a model that accurately predicts observations it follows that the fitness measure must evaluate the error in the predictions of the encoded model. The metric chosen to evaluate the error is the mean squared error, which is computed by solving the system of differential equations using numerical integration. Is important to mention here that the numerical integration procedure used to evaluate the error is the same that is applied during model fitting, thus no additional error is introduced by using a different method to solve the equations. Since

Finally, the fitness measure used also considers the number of processes that a model contains, penalizing models as they get more complex. The method consists of multiplying the fitness by a scaling function whose value depends on the number of processes present in the model. The more processes the model has, the bigger the scaling factor is, and thus models that have low error but over-fit the data with a complex structure are penalized, allowing simpler models with less accuracy to compete for survival. Since the model is not known in advance and the fitness measure introduces a scaling factor depending on the size of the model there is no optimal value known in advance, but the smaller the error the better. A fitness measure of 0 would mean perfect fit of the model to the data, but that is an unlikely case since we are dealing with time varying systems that are approximated with numerical integration.

Termination criteria

The criterion for terminating a run is maximum number of generations or plateau in fitness of best-of-generation individuals. The algorithm considers that it has reached a plateau when the best-of-generation individuals do not change their fit for a specified number of generations. In the experiments run a value of 10 seemed to be a good choice.

The schemata

Since the encoding used is based on binary strings the number of possible schemata in this representation is equal to 3^n where n is the number of possible partial instantiations of the generic processes using the variables being considered. To compute the value of n, we need to consider the types of the variables on which the induction takes place as well as the definition of the generic processes, computing the count of possible permutations of the variables that serve as parameters in each generic process.

To give a concrete example, we can consider the value of n in the situation where we have two variables of type population and the generic processes listed in Table 2. Processes with one variable as parameter, such

Table 2. Small library of generic processes for the population dynamics domain

```
generic process logistic_growth;
variables Pop{population};
equations
  d[Pop,t,1] = [0,50,100] * Pop * (1 - Pop / [0,50,100]);

generic process exponential_growth;
variables Pop{population};
equations
  d[Pop,t,1] = [0,50,100] * Pop;

generic process constant_inflow;
variables Pop{population};
equations
  d[Pop,t,1] = Pop;

generic process consumption;
variables Pop1{population}, Pop2{population};
equations
  d[Pop1,t,1] = - Pop1 * Pop2 / ([0,50,100] * Pop1 + 1);
  d[Pop2,t,1] = [0,50,100] * Pop1 * Pop2 / ([0,50,100] * Pop1 + 1);

generic process exponential_decay;
variables Pop{population};
equations
  d[Pop,t,1] = -[0,50,100] * Pop;
```

as logistic_growth, exponential_growth, and exponential_decay, can be partially instantiated with each of the variables in the problem, thus, each one of those processes results in two alternatives. The remaining consumption process has two variables as parameters, thus we can have as many partial instances as permutations can be considered for the number of variables and the number of placeholders for variables in the process. In this case there are two variables and the process has two parameter variables, thus we can obtain 2x2 = 4 possible partial instances. Finally, summing up all the possible partially instantiated processes we obtain 2+2+2+4 = 12 = *n*.

Since the operators applied in this algorithm are simple crossover and mutation over binary strings, the number of schemata than can be processed at each time step can be characterized with the bounds given in Golberg (1989). In particular, for crossover we have a minimum of 2^L schemata being processed simultaneously, and an upper bound of N x 2^L, for strings of size L and population of size N. Specific values depend on the particular problem since the algorithm proposed in this paper is not tailored to solve only one instance of the process model induction problem.

4 Experimental Results

To analyze the performance of the algorithm two experiments with varying degree of complexity where run. The complexity of the search space was increased by increasing the number of variables used for the induction task, since that is a parameter that greatly affects the size of the search space.

The first experiment had the goal of inducing a simple process model for a two species predator-prey relationship as the one presented in Table 1. The data set used was generated by simulating the system of differential equations for 200 time steps using one of the differential equation solvers included in the mathematical software MATLAB. The setup for the algorithm considered the generic processes presented in Table 2 and the variables “predator” and “prey”. Table 3 shows a Tableau summarizing the choices made for this experiment.

The complexity of the search space involved for this simple example is 2^{12}, since that is the number of possible combinations of partially instantiated processes. The size of the population was selected to be 30 so as to give the algorithm the opportunity of doing some search, since a bigger population number increases the odds of having a correct model in the initial population. To evaluate the performance of the algorithm it was run 20 times on the same data set using different randomly chosen initial populations. In all the runs the algorithm was able to successfully induce the correct model in a few generations. The number of runs needed to obtain the correct solution ranged between 0, for one case where the solution appeared in the initial population, to 12 in the case that took the longest.

Table 3: Tableau for the two-species predator prey induction problem.

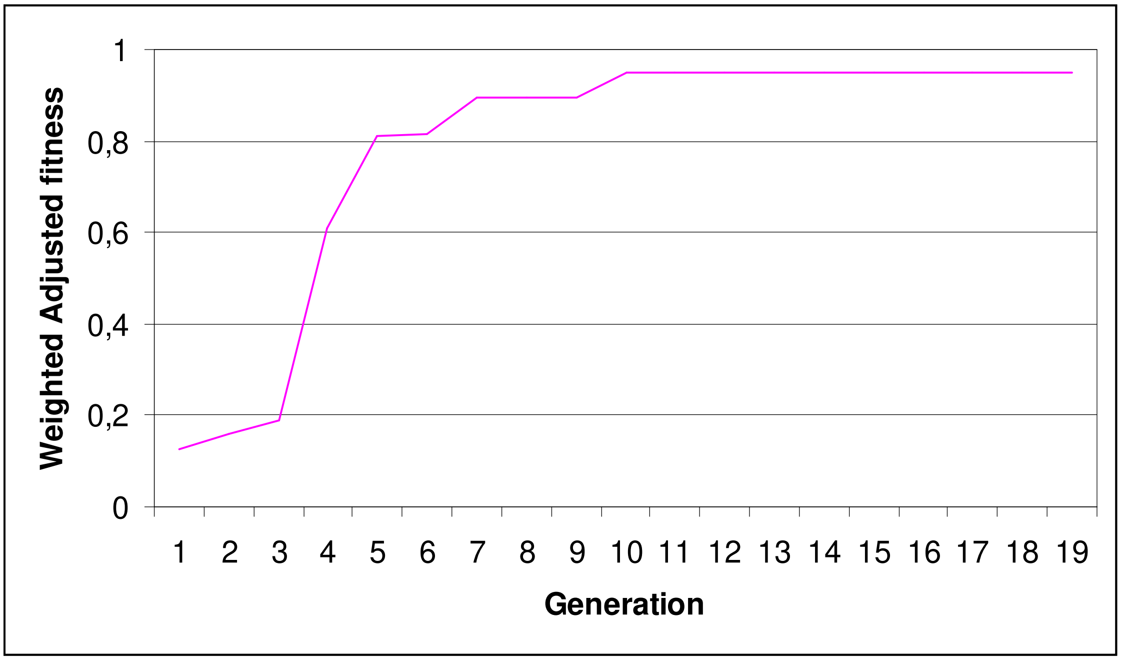
Objective:	Induce a process model for a two-species predator-prey relation with specific instances of the following generic processes: predator consumption of preys, exponential decay of the predator, and logistic growth of the prey that accurately predicts the data used as observations.
Representation scheme:	Structure = fixed length string of 12 bits Alphabet size K = 2 (binary) Mapping = Each bit in the chromosome encodes the presence or absence of the correspondent partially instantiated process as explained in Section 2.
Fitness cases:	Observations of the variables predator and prey for 200 time steps
Fitness:	Raw fitness = Standardized fitness = Mean squared error between the predictions generated by the model and the 200 fitness cases weighted by the scaling factor which is defined as a linear function that penalizes with 10%increase in error models containing the maximum number of processes and with 0% those models containing 1 process.
Parameters:	Population size M = 30 Maximum number of generations to be run G = 40
Termination criteria:	The GA has run for G generations or a plateau for the best of generation individual has been reached (I consider that the algorithm is in a plateau if the best of generation does not change for 10 generations).
Result designation:	The best-so-far individual in the population

The second experiment that was run had the purpose of evaluating the algorithm when faced with a considerable increase in the search space. Therefore, using the same set of generic processes the problem was increased in complexity by adding one more variable to be predicted. The addition of that variable increased the search space by an order of magnitude jumping from 2^{12} to 2^{21} possible structural models. While the previous experiment considered a search space that could be easily explored exhaustively, the one considered in this second experiment is big enough to make such approach infeasible.

In this case again 20 runs of the genetic algorithm where performed using randomly selected initial populations. In all the cases the algorithm was able to approximate or obtain the optimal solution in as few as 20 generations. Table 4 presents a Tableau that summarizes the main parameters used for this experiment, being the most noticeable change the increase in the population size. The increase in population size is introduced since as the theory dictates we need to provide room in the population to maintain a diversity of schemas that can be exploited by the genetic search.

The evolution of the algorithm as it searches though the space of possible model structures is illustrated in Figure 2 which shows the evolution of the weighted adjusted fitness for the best-of-generation individual in a particular run. Notice that the adjusted fitness varies between 0 and 1, and the best value is 1. As can be seen in the figure the algorithm quickly converged to an optimal solution, however the curve does not reach 1, which would be perfect fitness, because it is being weighted by the scaling factor that penalizes complex models.

Figure 2: Best-of-generation evolution in a run of the three species food chain model



5 Discussion of Results

The experimental results obtained show that applying genetic algorithms to the task of process model induction is a valid approach and that is an interesting alternative to the current algorithm for that task. In particular, the proposed algorithm attains similar results to the previous induction algorithm in terms of finding the optimal models, but it achieves that at a much lower cost in terms of the number of evaluations required to reach a solution.

In particular, for the case of the process model involving the food chain, the previous algorithm requires a search through a space of more than 100,000 possible models. In contrast, the algorithm proposed in this work required between 2000 and 3000 evaluations to easily find or closely approximate the solution of the problem. Moreover, the results show that the algorithm not only is capable of efficiently searching through the space of possible models, but it can also determine the appropriate size of it without having to artificially set that as a parameter of the algorithm.

Finally, analysis of the computational effort required to solve the two problems explored in the experiments seems to support the hypothesis that this evolutionary search through the space of model structures can scale more gracefully with the size of the problem. This statement is based on the observation that a sudden increase in the size of the search space from 2^{12} to 2^{21} possible structures did not degrade in the same proportion the performance of the algorithm to find the solution to the problem. In contrast, the previous algorithm for process model induction suffered a more noticeable degradation from the increase in complexity, requiring a computational effort that is several times bigger than what has been observed for the algorithm proposed in this paper.

Table 4: Tableau for the three-species food-chain induction problem.

Objective:	Induce a process model for a three-species predator-prey relation that describes a food chain involving grass, sheeps, and wolves. The target model contains the processes: logistic growth of grass, logistic growth of sheeps, sheeps consuming grass, wolves consuming sheeps, and finally exponential decay of wolves.
Representation scheme:	Structure = fixed length string of 21 bits Alphabet size K = 2 (binary) Mapping = Each bit in the chromosome encodes the presence or absence of the correspondent partially instantiated process as explained in Section 2.
Fitness cases:	Observations of the variables grass, sheeps, and wolves for 100 time steps.
Fitness:	Raw fitness = Standardized fitness = Mean squared error between the predictions generated by the model and the 100 fitness cases weighted by the scaling factor which is defined as a linear function that penalizes with 10%increase in error models containing the maximum number of processes and with 0% those models containing 1 process.
Parameters:	Population size M = 100 Maximum number of generations to be run G = 60
Termination criteria:	The GA has run for G generations or a plateau for the best of generation individual has been reached (I consider that the algorithm is in a plateau if the best of generation does not change for 10 generations).
Result designation:	The best-so-far individual in the population

6 Conclusions

This paper has presented a new algorithm for inducing process models that applies genetic algorithm techniques to search through the space of possible model structures. The results obtained in the experiments performed show that the algorithm can effectively induce process models and that it can achieve that goal with less computational effort that the established induction algorithm.

Empirical evidence seems to suggest that the algorithm can gracefully scale to problems of higher complexity suffering much less degradation in the performance than what has been observed in the algorithm that has been previously proposed for the task of process model induction.

Moreover, the use of genetic algorithms to perform the search of the space of possible model structures opens the possibility of approaching problems that are far more complex than the ones being analyzed today for process model induction. This claim is based on the fact that genetic algorithms can easily take advantage of distributed resources where parallel computations of the costly model fitting procedure can be performed.

Finally, a lesson learned from this work is that the choice of using optimization techniques not based on genetic algorithms to perform the fitting of the parameters of the model was appropriate but will not scale well to more complex problems since a severe degradation on the performance of the numerical optimization routines was observed when analyzing the runs of the algorithm.

Future Work

An immediate extension to the initial study carried out in this paper is extending the proposed approach to induce process models containing unobservable terms. Such models are the next step in complexity that should be tackled, and research on appropriate modifications to the representation need to be studied.

Another area where future research can take place is in incorporating genetic algorithms to also explore the space of parameters for the model structures. Moreover, it would be worthwhile to analyze if considering simultaneous evolution of the structure and parameters using evolutionary techniques can be more effective than the current approach that searches at two levels.

Finally, this work centers on the induction of process models as described in Langley et al. (2002), but recently the modeling language has started to incorporate higher level concepts such as mappings to physical systems. Inducing process models using those other levels of structure is also an interesting area of research.

References

Langley, P., Sánchez, J. N., Todorovski, L., & Dzeroski, S. (2002). Inducing Process Models from Continuous Data. In: *Proceedings of the Nineteenth International Conference on Machine Learning*. In press. Sydney: Morgan Kaufmann.

Pazzani, M. J., Mani, S., & Shankle, W. R. (2001). Acceptance of rules generated by machine learning among medical experts. In: *Methods of Information in Medicine*, 40, Pages 380–385.

Cartier, J., Rudolph, J., & Stewart, J. (2001). The nature and structure of scientific models. The National Center for Improving Student Learning and Achievement in Mathematics and Science, Working Paper.

Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. (1992). *Numerical Recipes*. Cambridge, MA: Cambridge University Press.

Todorovski, L., & and Dzeroski, S. (1997). Declarative bias in equation discovery. In: *Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco: Morgan Kaufmann. Pages 376–384.