Combining Nominal and Continuous Properties in an Incremental Learning System for Design

by

Yoram Reich

EDRC 12-33-89

Combining nominal and continuous properties in an incremental learning system for design

Yoram Reich
Department of Civil Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract: Research in machine learning has produced many learning algorithms mainly for classification tasks. This paper reports on an extension made to the learning program COBWEB to allow it to handle examples described by a more complex description language. The paper describes Bridger, a system that implements this, as well as other extensions. Bridger, combines leaning and performance in design tasks. The extension implemented has been tested succesfully in four design domains. This extension and others are necessary for allowing Bridger to master design rather than simple classification tasks.

This work has supported in part by the Engineering Design Research Center, a National Science Foundation Engineering Research Center, and the Sun Company Grant for Engineering Design Research.

1 Introduction

The process of knowledge acquisition for any type of expert system is time-consuming and tedious. This effort increases when dealing with design domains that are ill-structured in their nature. One approach to alleviate the difficulty of the knowledge acquisition process is the introduction of learning into system development and maintenance stages. Research in machine learning has produced a variety of algorithms for the *learning from examples* paradigm that have the potential of acquiring expert knowledge from examples of expert decisions . Most of these programs where developed and tested in classification or diagnosis tasks; their applicability has not been proved for design domains.

Design is considerably different than diagnosis. In design, a complex description of an artifact is required as a solution, whereas in classification or diagnosis, the result is a single classification assignment. Design problems are typically under-constrained, leaving a large space to explore and resulting in several possible alternative solutions. Although optimal designs are preferred, the complexity of the process admits satisficing designs as acceptable. In contrast, diagnosis should outputs one class as the result.

An inherent characteristic of design (and other domains as well) is that real designs are described by both nominal and continuous property types. Many design aids provide numeric data to be used for designing, and thus necessitate coupling symbolic with numeric computations (Kitzmiller and Kowalik, 1986). Any learning algorithm for design domain should support this coupling. Common approaches have used discretization that transforms continuous to symbolic symbolic, resulting in an initial *bias* in the learning process. Such an approach was proven to be poor for medical diagnosis (Sharma and Sleeman, 1988), a domain that is conceptualized as less complex than design.

Among the learning programs recently explored, COBWEB (Fisher, 1987) seems most promising. COBWEB is incremental; it creates a hierarchical structure of concepts that might support top-down refinement; and it supports *flexible prediction* that can output several properties based on the remaining properties. Until today, COBWEB has only been tested on classification tasks and has demonstrated very good performance.

We have explored BRIDGER, a system built on the foundations of COBWEB for learning in design domains (Reich and Fenves, 1989b; Reich, 1989). We have shown how COBWEB'S prediction scheme is similar to case-based design and how another new prediction scheme can be viewed as top-down refinement. We have also examined a mapping between Bridger and a general theory of design (Reich, 1989).

In this paper, we concentrate on the first requirement discussed above: supporting learning from

examples described by a more complex description language. We have extended the COBWEB approach to handle continuous properties in addition to nominal properties, and have tested it successfully in four design domains previously published in the literature (ordered in increasing complexity):

1. floor slab design (Mackenzie and Gero, 1987),
2. oil lubricant design (Kamal et al., 1989),
3. window arrangement design (Mackenzie and Gero, 1987), and
4. bridge design (Reich and Fenves, 1989b; Reich, 1989).

The contribution of the paper is the extension of an incremental learning program to deal with more complex domains. It allows a natural treatment of continuous and nominal property types and supports generalizing abstract concepts described by ranges of continuous values. Combined with previous studies, the present research supports our approach for the incorporation of learning in design systems. Our approach is introduced in Section 2. Section 3 describes Bridger, an instantiation of our model with an emphasis on the specific problem addressed in the paper: the extension that handles continuous properties. Section 4 provides four examples of using the extension in design domains and section 5 discusses the approach and elaborates on methodological aspects of using learning in design domains. Section 6 summarizes the results and points to additional work in this and other related areas.

2 Learning in design

Our approach to the incorporation of learning in the knowledge acquisition and performance improvement of design systems stems from three ideas.

1. *design examples contain knowledge.* The first idea is that heuristic knowledge or style about design is implicitly captured in previous designs. These design examples can be used to extract that knowledge by using *learning from examples* methods.
2. *domain theory is too complicated for use in synthesis.* The second idea realizes that knowledge mainly exists in theories. However, such knowledge is useful only in the evaluation stage of design rather than in the synthesis process. Learning should gradually transform theories into heuristic knowledge that is readily useful. Since evaluation is done by theories (for example, finite-elements programs for calculating stresses and deflections of structures) that are much easier to encode, the process of building design systems should be easier. In addition, if there

exists a critic or a redesign system for a domain (also easier to encode than synthesis system), learning can transform it into a synthesis knowledge that gradually does not require redesign.

3. *optimal solutions are the ultimate goal of design.* The third idea is that design is a complex process and optimal designs can only be obtained if the design system has enough resources. Alleviating the complexity of design frees resources for advancing toward optimal designs.

These ideas support the use of learning in design domains. One way to categorize learning methods is to classify them into non-incremental and incremental systems. In a non-incremental system, all design examples are provided and the system extracts knowledge that is later used by an expert system; whereas in an incremental system knowledge is gradually accumulated in response to an inability to design appropriately.

Any design domain is dynamic in nature. New technologies emerge, new materials are introduced, and domain theories are extended or replaced. The addition of one property for describing designs in a domain can result in major problems when using non-incremental learning after considerable knowledge has been accumulated. This dynamic nature necessitates the use of incremental learning in design systems.

Figure 1 provides a simple view of the approach. Examples of designs can be obtained from the literature or from the system designs evaluated by a human designer or a system critic. These examples are used to enhance the system knowledge that is used to design artifacts for given specifications. Designs are given to a critic that evaluates them and submits them to a redesign system if necessary (the redesign system is not necessary for the learning process, but it speeds up the process). The learning process makes use of the current state of knowledge and the new design example presented.

The incorporation of learning as an integral part of design systems must support two additional requirements. First, the learning system should handle complex description space that is appropriate for describing designs. Second, the learning process should produce knowledge that operates in an acceptable design paradigm (for example, top-down refinement strategy).

3 Bridger as a model of learning for design

Our model for the incorporation of learning in design is implemented in Bridger. Bridger is a domain-independent learning system for knowledge acquisition and performance improvement that is currently under development (Reich and Fenves, 1989a). Bridger is built on the foundations of the learning program COBWEB (Fisher, 1987). It extends COBWEB along several dimensions. In particular,

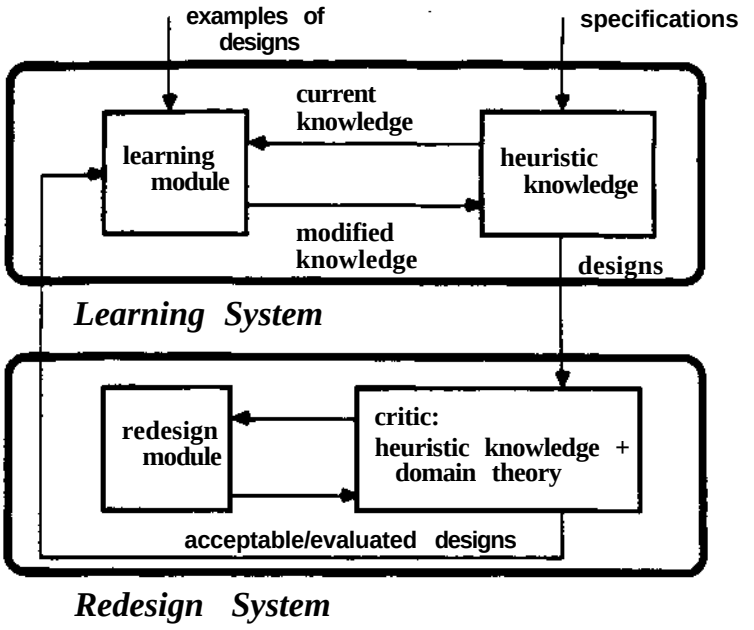

Figure 1: Incorporating learning in design

COBWEB handles examples described by a list of *nominal* property-value pairs. CLASSIT, a variant of COBWEB accepts descriptions of only continuous properties (Gennari et al., 1989), but does not handle a combination of the two property types. In contrast, Bridger can handle entities described by a combination of nominal or continuous property types. In addition, Bridger has a correcting-hierarchy module; it has a richer set of learning operators; it can forget undesired knowledge; and it can perform directed experimentation that increases the utility of its knowledge. Such extensions are necessary for the aoility to master design rather than simple diagnostic domains. The following discussion concentrates on the knowledge organization and a simplified description of the performance behavior of Bridger. The description uses design domain concepts although Bridger is not restricted to work in any specific domain.

Bridger uses an incremental learning scheme for the creation of hierarchical classification trees. Bridger accepts a stream of designs described by a list of property-value pairs. Designs need not be classified as feasible, optimal, or by any other classification scheme. The classification emerges from the structure of the domain. Any *a priori* classification can be assigned to a design and treated as any other property. For example, the cost of an object is a continuous evaluation property that can be used to classify objects into cheap and expensive groups.

A clustering[1] is of a good quality if the description of a design can be guessed with high accuracy.

[1]We use clustering or classification and *class* of a clustering or *node* in the classification tree interchangeably.

given that it belongs to a specific class. Such clustering promotes inferencing since it allows the prediction of property-value pairs based on class membership. Bridger makes use of a statistical function that produces a clustering of a design set into mutually exclusive classes, $C\backslash, C2, \ldots, C_n$. The function used by Bridger is:

$$\textit{classification utility} = \frac{\sum_{k} P(C_k) \left[\sum_{i} \sum_{j} P(A_i = V_{ij} \mid C_k)^2 - \sum_{i} \sum_{j} P(A_i = V_{ij})^2 \right]}{n} \qquad (1)$$

where :

Ck is a class,

Ai = V^ is a property - value pair,

P(x) is the probability of x, and

n is the number of classes.

The first term in the numerator measures the expected number of correct property-value pairs that can be guessed correctly by using the classification. The second term measures the same quantity *without* using the classes. Thus, the classification utility measures the *increase* of property-value pairs that can be guessed *above* the guess without the classification based on frequency alone. The measurement is normalized with respect to the number of classes.

The term *P(Ai = Vij)* is calculated by dividing counters that store the number of times an object in the example set had the property-value $Ai = V_{,j}$ and the number of times an object had property *Ai* in his description. The term $P\{A_L = V//|C^*)$ is calculated similarly by summing over objects in C* only. *P(Ck)* is the number of objects in class C* relative to the total number of objects. This method of calculation is not appropriate for continuous properties since the probability of a single event in a continuous distribution is zero.

Previous approaches for handling numeric data favored converting numbers into symbolic representation. Other approaches allowed simple learning with numeric data (creating ranges or threshold values) but only in non-incremental systems (for example: C4 (Quinlan et al., 1987) or CLUSTER/2 (Michalski and Stepp, 1983)). The first approach of discretizing numeric data suffers from the necessity to a priori recognize meaningful ranges - a prerequisite that defeats the purpose of learning. Furthermore, a slight error in the pre-assignment of range boundaries may result in errors in the learning process. The second approach is preferred, although it has not been demonstrated in incremental systems nor in noisy or fuzzy domains.

In order to handle continuous properties we assume that **the property-values observed are** instances generated from **a** normal distribution (compared to **the** equal **distribution for nominal** properties). This assumption allows us to estimate the function *p(x)* **that** provides **the probability of obtaining the value** *x.* The following formula allows us to accommodate continuous properties **in the** classification utility function:

$$P(Ai = V_{ij}) \equiv \int_{V_{ij}-d}^{V_{ij}+d} p(x)dx \qquad (2)$$

where :

$$Id = \frac{expected\ range\ of\ values}{expected\ number\ of\ distinct\ intervals}$$

The parameter *2d* is required for performing the integration. The sensitivity of the method to the choice of the parameter will determine the utility of this technique.

Bridger builds the clustering hierarchy in the following way. When a new design is introduced, Bridger tries to accommodate it into the existing tree. Bridger starts its process from the root of the tree. The sub-classes of the root form the classification at this level **of** the tree. Given the new design and the current classification, Bridger can perform one of **the following** operators:

1. expanding the root, if it does not have any sub-class, by creating a new class and attaching the root and the new design as its sub-classes;
2. adding the new design as a new sub-class of the root;
3. adding the design to one of the sub-classes of the root;
4. merging several sub-classes and putting the new design into the merged node; and
5. splitting a sub-class and considering again all the alternatives[2].

If the design has been accommodated into an existing sub-class, the process recurs with this class as the root of a new tree. Bridger uses the utility function to determine the next operator to apply. The best operator is the one that results in a new clustering that maximizes the utility function. This control constitutes a simple hill climbing strategy and results in the creation of a hierarchical structure of classes. Each class can be viewed as a generalization of all the classes below it.

Since the algorithm is incremental, there is no backtracking. However, the collection of operators

[2]Merging and splitting can be performed on combinations of nodes **richer than is possible in COBWEB.**

(e.g., split **and** merge) allows for simulating backtracking. The incremental nature of learning is achieved by storing in each node statistical information about the designs stored below that node. This information is updated each time a new design passes through that node.

Bridger designs using a mechanism similar to the one used for augmenting the tree by new designs but allowing only one operator - the *add-to-best-son* - to apply. Also, statistical information stored at tree nodes is not updated in design. Bridger sorts the new specification through the tree to find the best host for the new specification. The design progresses by assigning the new artifact characteristic property-value pairs describing the nodes traversed, when the specification is sorted through the tree. This strategy, which is different than the original method employed by COBWEB, can be viewed as a least-commitment process (Reich, 1989).

4 Examples

We tested the extension that deals with continuous properties in four design domains. Two experiments were conducted in each domain. In the first experiment, Bridger incrementally learned the example-set and designed the *complete* set given the specifications only. Converging to 100% indicates that the *knowledge* implicitly present in the examples has been assimilated into the system in the form of a classification hierarchy. The second experiment is a standard learning-performance test. Bridger used a subset of the examples as a training set and the remaining examples as the test set. This experiment was conducted with several splits to these subsets [3]. The first experiment is useful when only a small number of examples exist such that each example contains an additional meaningful piece of information not existing in the examples observed before. The second experiment is most useful in assessing the learning approach if the example set is large enough to cover the domain with redundancies, otherwise poor performance will result.

Each experiment was performed 10 times with random selection of the training set and random ordering of the examples. The results of the first experiment are described by figures 2a, 3a, 4a, 5a, and 6a; and those of the second in figures 2b, 3b, 4b, 5b, and 6b. These figures provide only a quantitative account of the performance. We do not provide a qualitative analysis of the results (e.g., evaluating the classification tree and the generalized concepts). Such analysis for the bridge domain can be found in Reich and Fcnves, 1989b.

In all the figures, the horizontal axis describes the number of examples learned. It is important to note that the significant aspect is not the absolute number of examples, but rather the number of the

[3] **Common practice in machine learning is to report such experiment for only one split**

examples learned, relative to the total number of examples in the data set. In the first set of figures, the vertical **axis shows** the accuracy of designing **the** complete set summarized over all the design properties. In **the second** set, the vertical axis shows the accuracy of designing the unseen examples based only on their specifications. These graphs show the accuracy for each property separately.

Floor slab design. This domain consists of 32 examples of floor slabs (Mackenzie and Gero, 1987). Each floor slab is described by three continuous specification properties: the span, the cost and the thickness of the slab; and by one nominal design decision: the type of construction. The original study generated rules by analyzing the shape of Pareto-optimal sets of the inverse design problem.

Figure 2 shows Bridger's performance in the floor slab domain. In the first experiment, Bridger rapidly assimilates the knowledge present in the examples and can replicate them at 100% accuracy.

Bridgets performance in the second experiment is also good but does not converge to 100%. The fact that the performance is enhanced as more examples are learned suggests that the examples share many relations but each adds additional information. This is probably a minimal training example set for this domain.

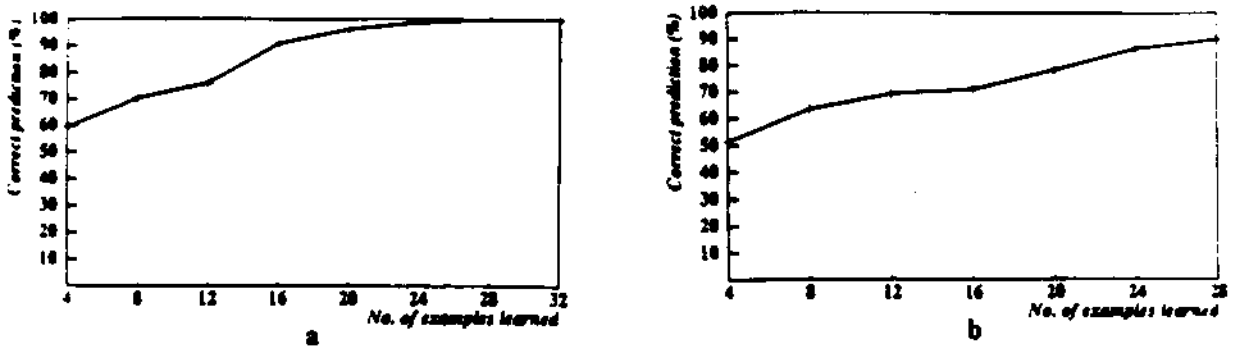

Figure 2: Design performance: the slab domain

Oil lubricants design. This domain consists of 37 examples of oil lubricant solutions (Kamal ct al . 1989). Each solution is described by five continuous specification properties: the kinematic viscosity at 40 and 100 c, viscosity at low and high temperatures, and loss of viscosity due to shear at high temperatures; and by one nominal design decision: the grade of oil required. The original study created rules 6y using a hybrid form of ID3 (Quinlan, 1986) after discrctizing the continuous properties.

Figure 3 shows Bridger's performance in the oil lubricant domain. The behavior of Bridger in this domain is similar to its behavior in the floor slab domain. In the second experiment, the performance

improves at a **slower** rate than in the previous domain. This suggests that a proper training set for this domain should **include** more examples. Consequently, the number of examples used is small relative to the required size of an appropriate training set.

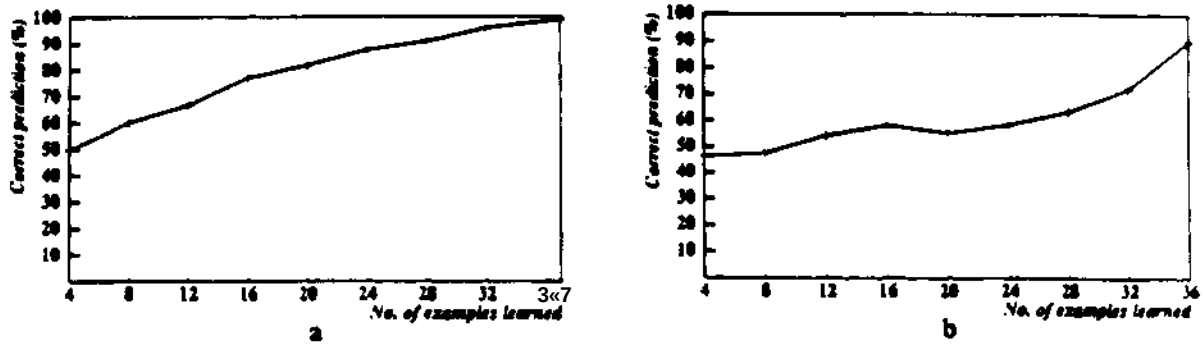

Figure 3: Design performance: the oil domain

Window arrangement design. This domain consists of 72 examples of window arrangement to obtain some environmental objectives (McLaughlin and Gero, 1987). Each design is described by three continuous specification properties: the daylight factor, the summer temperature and the winter temperature desired. In addition, each design is described by four nominal design decisions: the type of glass, the type of wall, the size of window and the size of the sunshade. The original study generated rules by using ID3 to discriminate between Pareto-optimal designs from inferior solutions.

Figure 4 shows Bridgets performance in the window arrangement domain. The first experiment didn't produce encouraging results. The performance level is around 70%, which suggests that many more examples are needed to form good clusters of the specifications. The performance in the second experiment is not different. The predictive accuracy obtained is similar to frequency prediction except for the glass property where the accuracy is doubled. These results strengthen the claim that this data set is two 'sparse' in the domain and appropriate performance requires additional examples.

Pittsburgh **bridge domain.** This domain includes 108 bridges constructed in Pittsburgh since 1818[4]. Each design is described by seven specification properties (three continuous and four nominal): the river and exact location of the bridge, the period it was constructed, the purpose of the bridge, the number of lanes and length of the bridge, and whether a vertical clearance requirement was enforced in the design. Five design properties are provided for each design: the material used, the span of the

[4]**The data-set is available from the author upon request**

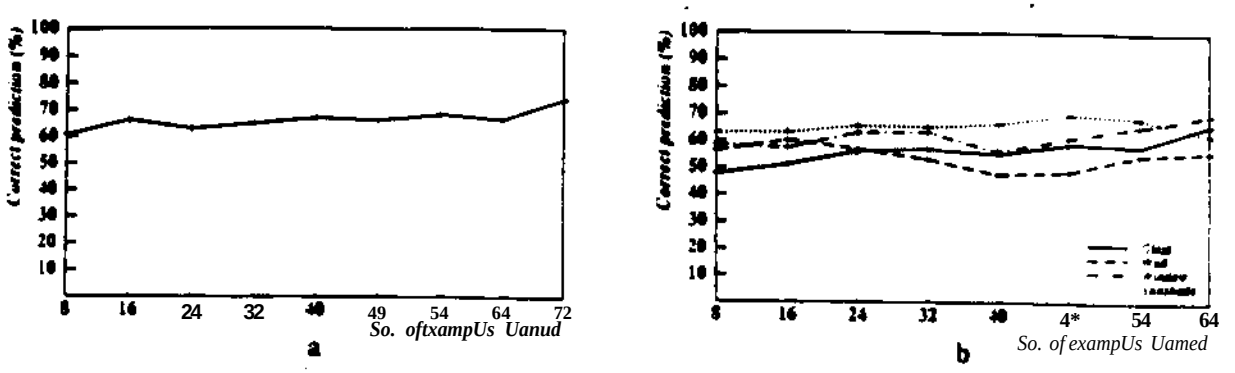

Figure 4: Design performance: the window **domain**

bridge, the type of bridge, the location of the road with respect to **the bridge, and the** relative size of the span to the channel width.

Extensive studies have been performed with COBWEB (implementation of **the** original algorithm with additional extensions) in this domain (Reich and Fenves, **1989a; Reich and** Fenves, 1989b). Figure 5 shows Bridger's performance in the bridge domain with **the original continuous** properties. Performance is enhanced as more examples are learned in both **experiments. The second** experiment shows that two properties are predicted well, whereas the remaining **three arc less accurately** predicted although they have major engineering importance. All the above **remarks suggest that the** data set is too small to capture the complexity of this domain. The first **experiment, however,** shows that the examples learned are assimilated appropriately into the system's **knowledge.**

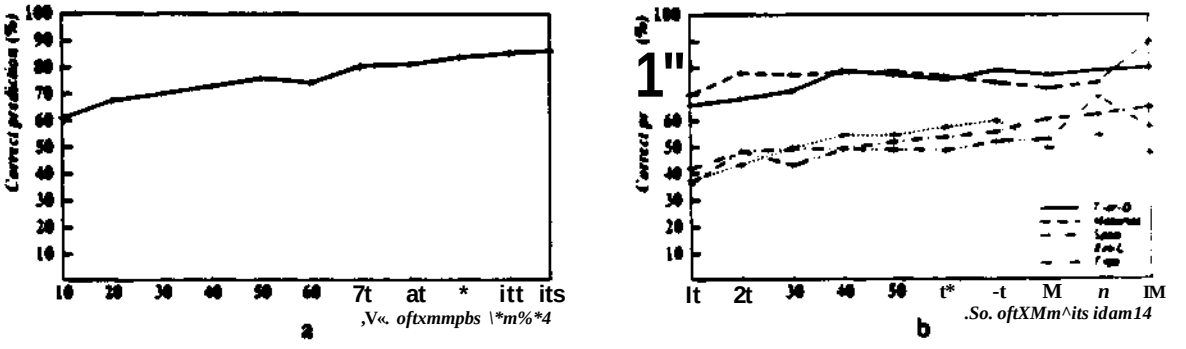

Figure 5: Design performance: the bridge domain with continuous properties

Figure 6 shows Bridger's performance in the bridge domain after discretizing the continuous properties. The same previous remarks hold for this figure. Compared to the learning with combined continuous and nominal properties, the performance after discretization is inferior in both experiments,

although not in a statistically significant manner. Bearing in mind that the discretization was suggested by an expert in this domain, the performance with the new extension is very encouraging.

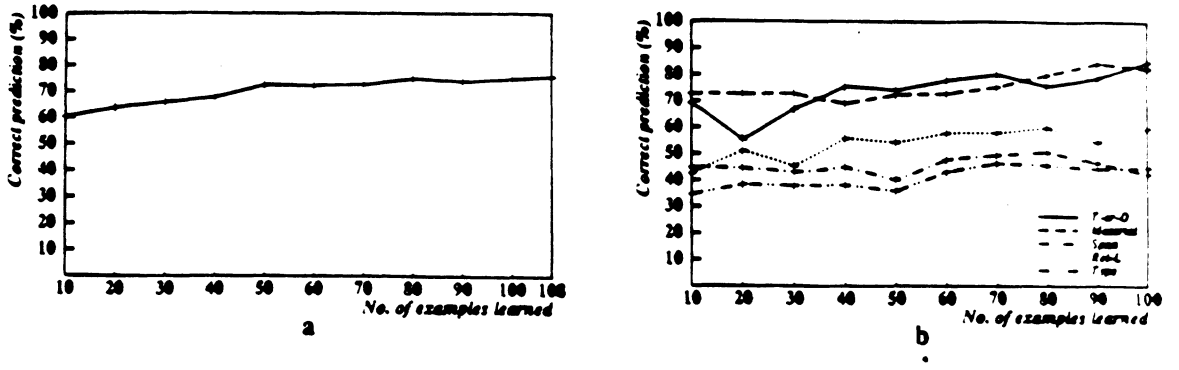

Figure 6: Design performance: the bridge domain with discretized properties

5 Discussion

The experiments show that the extension for handling continuous properties performs better than the original approach of discretizing numeric data by an expert. This was obtained in a rather simple domain. Complex domains preclude such perfect discretization, and thus become prone to errors. Similar performance was obtained when the parameter $2d$ was changed, indicating that the method is not sensitive to this choice. It has been observed that if a domain contains 'high' and sparse 'peaks' (see Rendell and Cho, 1988 for complete discussion on characterizing domains), the parameter $2d$ should be increased.

The results presented in the previous section point to some important issues of the appropriate methodology for testing machine learning in general, and in design domains specifically. We will concentrate on the general issue of designing an appropriate experiment for testing learning systems, and continue with the specific issue of learning in design.

The previous section provided Bridger's performance in two types of experiments. The first experiment measures how well the examples are assimilated into the hierarchy, and the second, the performance on a new test set. It is not clear that successful performance in one experiment will be demonstrated in the second. It is important to characterize classes of problems and domains where each of the experiments is appropriate[5]. Given a small data-set of examples, one cannot hope to obtain

[5]An example of a characterization of domain complexity in general, is given in (Rendell and Cho, 1988).

appropriate results in the second experiment. This experiment assumes **a** large example-set such that a random subset of reasonable size will 'cover' **the** domain. Since test examples are taken from the same population, this experiment is valid. If **a** small number of examples is available or the domain is 'large' relative to the number of examples available, the first experiment is more appropriate, since it shows the ability to handle the set learned or **a** similar set. Such an approach is best suited for large domain if a simple generator of examples exists. The generator can be used to generate a test set closely related to the training set, and thus simulate the second experiment. Such a setting combines the two tests. This experiment has been performed in the bridge domain where simple critic and redesign systems were added as described in Figure 1. The leaming process proved to converge fast to perfect performance. Such an approach is highly suited for design, since in most cases there are generators that can be used without evaluation. We can conclude that the first experiment is suitable as a measure of performance for large design domains.

The second issue relates to the aspiration one has for the incorporation of learning in design systems. Our approach favors the integration of learning into the performance system. In this case, knowledge learned should be readily available to the performance system without pre-processing. Furthermore, since the performance system operates based on some general design methodology, the leaming system should create a knowledge structure that facilitates this processing. As an illustration, we will compare the way Bridger and ID3 support this requirement. We have mentioned that Bridger's design process, using the hierarchy it generates, can be conceptualized as a top-down refinement strategy. Nodes in the hierarchy designate abstract concepts in the bridge domain. Thus, Bridger's knowledge representation and design process are meaningful in the design realm. In contrast, ID3 creates a *set* of decision trees, each for every design decision property. These trees can be translated into production rules that are more concise. ID3 'designs' by using each tree to select the corresponding design decision. There is no sense of ordering or dependencies between the various design decisions. There is no ability to generate higher level concepts in the design domain. Consequently, Bridger's approach is recommended over ID3 for the use in design domains.

6 Summary

In this paper, we have concentrated on a simple yet necessary extension to a learning program and tested it in several design domains. We showed that this extension results in good performance, depending on the domain used and the availability of appropriate number of examples. Furthermore, we compared it to the use of discretized properties in the bridge domain and found it superior although not in a statistically significant way. This extension enhances the ability of the learning system to

work in design domains in a natural manner.

The work presented leads to additional work that remains to be done:

- A formal analysis should be performed and experiments on artificial domains should be conducted to understand the behavior of the extension. An important aspect of the analysis is whether the approach presented introduces an improper bias that favors either nominal or continuous properties.
- Tests on additional complex design domains should be performed. Specifically, we are implementing a system that would learn to perform a detailed preliminary design for cable-stayed bridges.
- The extension should be compared with the performance of other programs, for example, CLUSTER/2 (although it is not incremental).

Acknowledgments

I would like to thank Prof. Steven J. Fenves for his support and discussions on this research and to the use of his expertise in Pittsburgh's bridges.

References

Fisher, D. H., 1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2(7),* 139-172.

Gennari, J. H., Langley, P., and Fisher, D., 1989. Models of incremental concept formation. *Artificial Intelligence, (in press).*

Kamal, S. Z., Mistree, F., Sorab, J., and VanArsdale, W. E., 1989. The development of an inductive learning system for design using experimental information. In *Artificial Intelligence in Design, Proceedings of the Fourth International Conference on the Applications of Artificial Intelligence in Engineering,* pages 521-538, Springer-Verlag, Cambridge, England.

Kitzmiller, C. T. and Kowalik, J. S., 1986. Symbolic and numerical computing in knowledge-based systens. In Kowalik, J. S., editor, *Coupling Symbolic and Numerical Computing in Expert Systems,* pages 3-17, North-Holland, Amsterdam.

Mackenzie, C. A. and Gero, J. S., 1987. Learning design rules from decisions and performances. *Artificial Intelligence in Engineering, 2(1),* 2-10.

McLaughlin, S. and Gero, J. S., 1987. Acquiring expert knowledge from characterized designs. *AI*

EDAM, 1(2), 73-87.

Michalski, R. S. and Stepp, R., 1983. Learning from observation: conceptual clustering. In *Machine Learning: An Artificial Intelligence Approach,* pages 331-363, Tioga Press, Palo Alto, CA.

Quinlan, J. R., 1986. Induction of decision trees. *Machine Learning, 1(1),* 81-106. ID3.

Quinlan, J. R., Compton, P., Horn, K. A., and Lazarus, L., 1987. Inductive knowledge acquisition: a case study. In Quinlan, J. R., editor, *Applications of Expert Systems,* Addison-Wesley, Reading, MA.

Reich, Y. 1989. Converging to "ideal knowledge" by learning. To be presented at The First International Workshop on Formal Methods in Engineering Design.

Reich, Y. and Fenves, S. J. 1989a. Bridger: an integrated, domain independent learning system for knowledge acquisition and performance improvement. In preparation.

Reich, Y. and Fenves, S. J. 1989b. Incremental learning for capturing design expertise. Submitted for publication.

Rendell, L. and Cho, H., 1988. *Empirical Concept Learning as a Function of Data Sampling and Concept Character.* Technical Report UIUCDCS-R-88-1410, Department of Computer Science, University of Illinois as Urbana-Champaign, Urbana, 111.

Sharma, S. and Slecman, D., 1988. Refiner: a case-based differential diagnosis aide for knowledge acquisition and knowledge refinement. In Sleeman, D., editor, *Procedings of the Third European Working Session on Learning,* pages 201-210, Pitman, Aberdeen.