Constructive Induction by Incremental Concept Formation

Yoram Reich

Department of Civil Engineering and The Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

In Y. A. Feldman & A. Bruckstein (eds.), Artificial Intelligence and Computer Vision, 1991, pp. 191-204, Elsevier Science Publishers, Amsterdam.

Abstract

This paper describes a framework that generates constructive induction schemes for the concept formation system COBWEB. The basis of this framework—context-dependent bias of multi-valued properties—provides a way for allowing COBWEB to deal with continuous and hierarchical property types as a special case of constructive induction. The constructive induction scheme does not introduce learning bias and does not require major modification to the original concept-formation mechanisms. Bridger, a system that partially implements the constructive induction, as well as others extensions, is one of the first incremental concept formation programs with a general constructive induction ability.

1. INTRODUCTION TO CONCEPT FORMATION AND CONSTRUCTIVE INDUCTION

Concept formation is a fundamental activity. It structures observations (also called examples) into a concise form of knowledge that can be efficiently used in the future. Two such uses include the classification of unseen observations based on a subset of their properties and the qualitative understanding of those observations based on the knowledge generated from them. The importance of concept formation has resulted in the development of many concept formation programs that automatically generate clusterings or classifications of observations.

A concept formation program that has been shown to be powerful in several diagnostic and classification domains is COBWEB (Fisher, 1987). One of its descendants, Bridger, has been used in design domains for synthesizing artifacts from given specifications (Reich, 1990; Reich and Fenves, 1992b; Reich and Fenves, 1992a).

COBWEB was designed to use observations described by a language restricted to lists of nominal property-value pairs. Another recent descendant, CLASSIT (Gennari et al, 1989), also uses a restricted language consisting of lists of continuous property-values assumed to be drawn from a normal distribution. Such restricted languages are not appropriate for representing many real world domains such as design or medical diagnosis. Representing examples in real domains requires a language with a rich collection of types such as: ordered, continuous, or partially ordered (i.e., structured).

An implicit assumption is always made when a description language is fixed to certain features and property types. The mapping between the real objects to their representations in the language is assumed to be *natural*. By natural, we mean that the relationships between objects and their features in the real world are mapped and maintained in the language. Constructing such a natural description language requires the articulation of substantial domain knowledge. Even then, the resulting language may still prove to be too restricted. A simple example is the use of the shape, color and size properties to describe building-block objects. One can use these properties to differentiate between objects. However, if two objects have the same description, the language is not sufficient and consequently is not natural to the domain.

The elimination of an overly-fixed language requires the ability to enrich it. Enriching the description language, while keeping its nature to lists of properties, involves two activities. First, the language can be enhanced manually by allowing the use of new property types. Referring to the above example, an additional property such as the location of the object can enhance the language. Second, the system can be augmented by an automatic construction mechanism that can generate new features from the original property values.

Although these activities seem remote, they are in fact closely related. Both the manual addition of property types and the automatic construction of features, should be done *naturally* with respect to the system's mechanisms. In this paper, the term naturally is more specific than before. It relates to a different term, called learning bias, that is often used in machine learning to describe preferences of learning systems. The meaning of naturally here is that the addition of new property types or newly constructed features does not introduce a learning bias into the learning process. The creation of a bias-free enriching mechanism requires a good understanding of the original behavior of the system.

The manual enriching approach is usually pursued with the addition of property types and their ad-hoc generalization mechanisms. For example, the conceptual clustering program CLUSTER (Michalski and Stepp, 1983) uses several property types, each with its own generalizing operator, without analyzing whether an inherent bias is introduced by this combination. Usually, no such analysis is performed. One exception is Quinlan's (1985) analysis for the concept learning program ID3. The extent to which Quinlan has articulated his analysis was the demonstration that the criterion for choosing decision nodes in ID3 favors properties that have more values than the others; the suggestion of a remedy; and the comparison of this remedy with other approaches.

A constructive induction scheme that is both general and automatic has never been proposed for any concept formation program. Simple forms of term creation have been proposed for other concept formation programs or for closely related discovery programs. Examples include the creation of ranges of ordered values in CLUSTER, the creation of new terms in discovery systems such as BACON (Langley et al, 1987), or the creation and use of intermediate concepts such as in RINCON (Wogulis and Langley, 1989). In general, these schemes were not supported by any analysis and were restricted to few types of term creation. In addition, only few of these schemes were related to incremental systems.

This paper consolidates the two enriching activities into one framework. First, it provides a better understanding of the nature of the category utility (CU) function (Gluck and Corter, 1985) which is the evaluation function used by Bridger, as well as by COBWEB, for selecting among learning operators. Section 2 briefly reviews CU. Section 3 describes a simple analysis that approximately quantifies the bias of using properties having different numbers of nominal values. Instead of trying to prevent this bias, we use it in Section 4 to guide the design of a natural constructive induction method that is based on CU. Section 5 illustrates the application of the approach to continuous properties and Section 6 describes how the constructive ability with an additional shallow knowledge about properties provides the basis for a natural treatment of continuous, hierarchical, and possibly other property types.

Section 7 describes Bridger—the system that implements the constructive induction scheme. Section 8 provides results from using the technique in several domains. Preliminary results suggest that the method has potential, but requires enhancements that guide its behavior. Section 9 describes several possible extensions as well as a broader viewpoint of constructive induction for concept formation.

2. THE UTILITY OF CATEGORIES

A categorization of observations is measured based on its ability to support accurate prediction of missing properties of observations. These observations can include unseen cases as well as ones that are already included in the categorization. When predicting missing properties of an observation, a class that best matches the partial description of the observation is selected and its description is used to complete the properties missing in the observation. A good categorization is such that the ability to predict is maximized.

Gluck and Corter (1985) have suggested a measure of categorization called category utility (CU):

$$CU = \frac{\sum_{k=1}^{n} P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2}{n}; \quad (1)$$

where: C_k is a class, $A_i = V_{ij}$ is a property-value pair, $P(x)$ is the probability of x, and n is the number of classes. CU provides a balance between high inter-class differences and high intra-class similarities. In addition, CU provides a measure of the expected number of property values ($A_i = V_{ij}$) that can be predicted based on the classification. An optimal clustering can be generated by selecting the classes that maximize CU.

3. CONTEXT DEPENDENT BIAS OF MULTI-VALUED PROPERTIES

This section provides the basis for understanding and extending CU. It describes an analysis that assesses whether a minor change in the description language influences CU. The analysis is directed with the aid of a simple exercise. We imagine a domain of objects, each described by a set of properties ($A_i, i = 1, \ldots, m$). Each property has a value taken from a set of discrete values ($V_{ij}, j = 1, \ldots, n_i$). Given a set of objects, one can use CU to calculate a score for each candidate clustering. During the search for the best clustering, we wish to assess the effect of a "minor" change in the description language: we would like to collapse the first two values of A_1, i.e., V_1 and V_2, into a single value V_1^*. We denote the new property by A_1^*. The rest of the language remains intact. Such a modification is beneficial if it captures some intrinsic characteristics of the domain and if it results in an increase in CU. Calculating CU for the new language, CU for the old description language and subtracting them gives the change in the category utility that is obtained by the language modification. This change is denoted by δU and it equals (after rearranging and canceling terms):

$$\delta U = \frac{\sum_{k=1}^{n} P(C_k)\ [\,P(A_1^* = V_{11}^*|C_k)^2 - P(A_1 = V_{11}|C_k)^2 - P(A_1 = V_{12}|C_k)^2\,]}{n} - \frac{P(A_1^* = V_{11}^*)^2 - P(A_1 = V_{11})^2 - P(A_1 = V_{12})^2}{n}. \tag{2}$$

We assume now that for a particular clustering the number of times that $A_1 = V_{11}$ appears in example descriptions of objects in class C_k is a_k and the number of times that $A_1 = V_{12}$ appears is b_k. Clearly, the number of times that $A_1^* = V_{11}^*$ appears in example descriptions is $a_k + b_k$. We denote the number of times that A_i appears in example descriptions in class C_k by c_k. Finally, we denote the total number of examples in C_k by e_k. The same quantities for the complete set are a, b, c, and e, respectively. We use these values to estimate the probabilities in Equation 2 and obtain

$$\delta U = \frac{\sum_{k=1}^{n} \frac{e_k}{e} \left[\left(\frac{a_k+b_k}{c_k}\right)^2 - \left(\frac{a_k}{c_k}\right)^2 - \left(\frac{b_k}{c_k}\right)^2 \right] - \left[\left(\frac{a+b}{c}\right)^2 - \left(\frac{a}{c}\right)^2 - \left(\frac{b}{c}\right)^2 \right]}{n}. \tag{3}$$

After rearranging terms and assuming that there are no missing properties (i.e., $e_k = c_k$) we get:

$$\delta U = \frac{2}{n} \left(\sum_{k=1}^{n} \frac{c_k}{c} \frac{a_k\, b_k}{c_k^2} - \frac{a\, b}{c^2} \right). \tag{4}$$

There are three possible extreme cases for the relationship between the values V_{11} and V_{12} and the classification: (1) the values V_{11} and V_{12} are irrelevant to the classification; (2) the values V_{11} and V_{12} highly correlate with different classes; and (3) both values V_{11} and V_{12} correlate with the same classes.

Irrelevant property-values

If values V_{11} and V_{12} are distributed evenly in all the classes then $\frac{a_k}{c_k}$ and $\frac{b_k}{c_k}$ are similar to $\frac{a}{c}$ and $\frac{b}{c}$. Substituting these relations into Equation 4 causes δU to vanish. In this case, the change in the property does not influence CU.

Exclusive property-values

If the values V_{11} and V_{12} correlate with different classes in the classification, $\frac{a_k}{c_k} \approx 1 \;\Rightarrow\; \frac{b_k}{c_k} \approx 0$. Hence, we get:

$$\delta U \;=\; -\frac{2}{n}\,\frac{a\,b}{c^2}. \tag{5}$$

In this case collapsing the two values leads to loosing information necessary for the classification; thus, CU decreases.

Complementing property-values

Finally, if both property-values correlate with similar classes and the remaining values of A_1 correlate with different classes we get:

$$\delta U \;=\; \frac{2}{n\,c}\left(\sum_{k=1}^{n}\frac{a_k\,b_k}{a_k+b_k} - \frac{a\,b}{a+b+x}\right), \tag{6}$$

where x is the number of values other than V_{11} and V_{12}. Not always does Equation 6 produce a positive value. However, In most practical cases it does. For example, if $k = 1$ and $x > 0$ the value is positive. Consequently, in most instances where property values complement each other the value of CU increases.

We have shown that CU favors properties with more values depending on the contribution of these values to the classification. This constitutes a context dependent bias which is slightly different than the bias observed for ID3 (Quinlan, 1985). The reason we are not concerned with this bias is that in our scheme decision nodes are polythetic, and a decision about which learning operator to apply is made on a relative basis. Consequently, if a property introduces a bias into CU, it will do so for all the candidate operators. In contrast, decision nodes in ID3 are monothetic, thus it is important to divide the training set based on the best unbiased property.

4. CONSTRUCTIVE INDUCTION

Constructive induction (CI) can be defined as the creation of *useful* features not existing in the original property-value description of examples (Rendell, 1988). The original properties are called *ground properties*. The term useful means that the use of the new features should result in an improved leaning performance. This is a possible operationalization of the definition of the term *naturally* that was introduced before. Rendell's discussion takes place in the context of concept learning by selective induction that is non-incremental. This allows him to use constructive induction in response to learning performance since all the data is available in advance. Rendell's proposed framework consists of several mechanisms. Initially, a selective induction program uses an example set to learn generalized concepts. These concepts are evaluated qualitatively (e.g., by using a simplicity criterion) or quantitatively (e.g., by measuring their utility in a performance task). Poor learning performance that is detected by the above analysis triggers constructive induction that constructs higher order features from the ground properties. The original example descriptions are converted to accommodate the new features and another learning phase takes place. This process iterates until acceptable performance is achieved or no improvement is obtained.

We are concerned with similar mechanisms that create new features in the context of an incremental concept formation task. Such task places harder constraints on a prospective constructive method. There is no complete data set to analyze or learn in advance. One cannot iterate in a process of learning, evaluation, constructive induction, and re-learning. There is only one continuous learning phase without extensive feedback. Consequently, the constructive scheme must be *natural* to the system's behavior and must maintain the incremental nature of the overall learning process.

In designing the constructive induction we assume that the ground properties are sufficient to differentiate between examples, i.e., the case of combined exclusive property values does not exist in the original language. According to CU's bias, it is useful to combine complementary property values. For example two complementing values V_{11} and V_{12} can be combined into a new feature $G_1 = \{V_{11}, V_{12}\}$. This can be viewed as adding an internal disjunct into the description language. Additional values can be accommodated into or deleted from G_1 based on additional information. We can modify Equation 1 to work with such features in the following way:

$$CU = \frac{\sum_{k=1}^{n} P(C_k) \sum_i \sum_j P(A_i = G_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = G_{ij})^2}{n}; \quad (7)$$

where $G_{ij} = \{V_{ijl}|V_{ijl} \in Dom(A_i)\}$ are new features generated by the constructive induction such that: $\bigcup_j G_{ij} = Dom(A_i)$, $\bigcap_j G_{ij} = \phi$, and $Dom(A_i)$ is the domain of values of A_i. In addition, $P(A_i = G_{ij}) = \sum_l P(A_i = V_{ijl})$, $V_{ijl} \in G_{ij}$.

Initially features are initialized to ground properties. The constructive induction operates as an integral part of calculating CU, hence it tries to modify the features each time CU is calculated. The combination of two values is performed only if it increases CU. In principle, the best approach is to try all the combination of values each time CU is calculated. However, this process is exponential in the number of values. Another approach uses the new example that causes CU to be calculated as a trigger for the constructive induction scheme. Given a collection of features of a property, the constructive induction can only manipulate the features with respect to the property value of the new example. For each property in the description of the new example, the only property value that is allowed to change its feature is the one that describes the new example. To illustrate, if the property is the shape of an object, the current features of shape are $G_1 = \{square, rectangle, triangle\}$ and $G_2 = \{circle, ellipse\}$, and the shape of the new object is *square*, only *square* can change its feature. The remaining values are kept intact. There are three optional actions that the property value of the new example can perform:

1. maintain its current place in a feature;
2. leave its current feature and join another feature (for example, *square* can leave G_1 and join G_2, although it is unlikely given the values used in the example); and
3. leave its current feature and form a single-value feature (for example, *square* can leave G_1 and form $G_3 = \{square\}$.

In addition, features can be split or merged by using operators that are similar to the original split and merge learning operators[1]. For example, G_1 can be split into $G_3 = \{square, rectangle\}$ and $G_4 = \{triangle\}$ and then *square* can use one of the three above actions.

The new features are maintained in addition to the original ground properties. This allows to undo any feature construction and continue with the remaining constructed features in addition to the ground properties. Note that since Bridger constructs a hierarchical classification tree (see Section 7), constructed features will vary for different levels of the hierarchy.

The above scheme can be viewed as a *greedy* approach to CI. Each local opportunity for creating new features is exploited. Later, we illustrate some limitations and extensions of such an approach.

5. APPLICATION TO CONTINUOUS PROPERTIES

The method of calculation the probabilities in Equation 1 is not adequate for continuous properties since the probability of a single event in a continuous distribution is zero. We can use the constructive induction scheme to design an extension to CU that can handle continuous properties. Features can be naturally constructed for numerical properties by describing the values of the property as a distribution with a mean and a standard deviation.

The type of distribution must be initially assumed. Ideally, the assumption should be based on some domain knowledge; alternatively, common probability distributions can be assumed. Our approach does not commit to a certain distribution. However, one should be selected before starting a learning process. The type of probability distribution with the mean and standard deviation allows the estimation of a function $p(x)$, which provide the probability of obtaining the value x using the following formula:

$$\sum_j P(A_i = V_{ij}|C_k)^2 \Leftrightarrow P(A_i = \overline{V_i}|C_k)^2 \Leftrightarrow \left(\int_{-\frac{d_i}{\sigma_i}}^{+\frac{d_i}{\sigma_i}} p_i(x)dx\right)^2 \quad (8)$$

where: $p_i(x) = \frac{1}{\sigma_i\sqrt{2\pi}}e^{-(x-\overline{V_i})^2/2\sigma_i^2}$, $2d_i = \frac{\textit{expected range of property values of } A_i}{\textit{expected number of distinct intervals of property } A_i}$, $\overline{V_i}$ is the mean of the values of A_i in C_k, and σ_i is the standard deviation of the values of A_i. Equation 8 can be used to substitute the corresponding terms in Equation 1 (or 7), thus allowing CU to handle continuous properties.

[1]See (Fisher, 1987) for a detailed description of COBWEB's learning operators.

The approach presented is very flexible, no assumption needs to be fixed when information implicit in examples suggests otherwise[2]. If the distribution, initially assumed, contradicts the values in the sample, a better distribution can be used. Similarly, the expected range of property values or the expected number of distinct intervals can be modified during learning to accommodate changes in the domain without re-processing previous examples. Any change that needs to be done to the knowledge structure will be performed by the split or merge operators if warranted by CU. Another source of flexibility is the ability to accommodate continuous properties from different distributions in the same example description.

6. DOMAIN KNOWLEDGE AND CONSTRUCTIVE INDUCTION

In the constructive method presented, the construction of features is solely directed by the criterion of maximizing CU. When no domain knowledge exists, this is an acceptable and the default strategy. In other cases however, there exists knowledge about the structure of properties that establishes constraints on the features possible in the domain. For example, a constraint on the possible features for the shape property values described before might be: no feature will contain values from both $\{circle, ellipse\}$ and $\{square, rectangle, triangle\}$. Such knowledge can be easily incorporated into the method.

A simple example of the use of knowledge is the extension to continuous properties where the original values are grouped using the ordering and the distribution hypothesis (see Figure 1). Another example is the use of explicit knowledge about the structure (i.e., partial ordering) of a property to allow Bridger to accommodate this property type into its mechanisms. In this case, the *hierarchy-climbing* operator of structured properties (e.g., see (Michalski, 1983)) can be implemented by allowing features to be generated only if their values correspond to the leaf nodes of a single intermediate node in the property hierarchy. For example, for the structured property shown in Figure 2, the possible features are: $G_1 = \{circle\}$, $G_2 = \{ellipse\}$, $G_3 = \{square\}$, $G_4 = \{rectangle\}$, $G_5 = \{triangle\}$, $G_6 = \{circle, ellipse\}$ (which corresponds to *round* objects), $G_7 = \{square, rectangle, triangle\}$ (which corresponds to *polygon* objects), and $G_8 = \{circle, ellipse, square, rectangle, triangle\}$.

7. BRIDGER

Bridger is a domain-independent learning system for knowledge acquisition and performance improvement that is currently under development (Reich, 1991a; Reich, 1991b). Bridger is built on the foundations of the concept formation program COBWEB (Fisher, 1987), but extends it along several dimensions. Bridger can handle entities described by a combination of nominal and continuous property types. It has a correcting-hierarchy module; it has a richer

[2] In contrast, CLASSIT (Gennari et al, 1989), a descendant of COBWEB, can use only continuous property-values that are drown from a normal distribution.

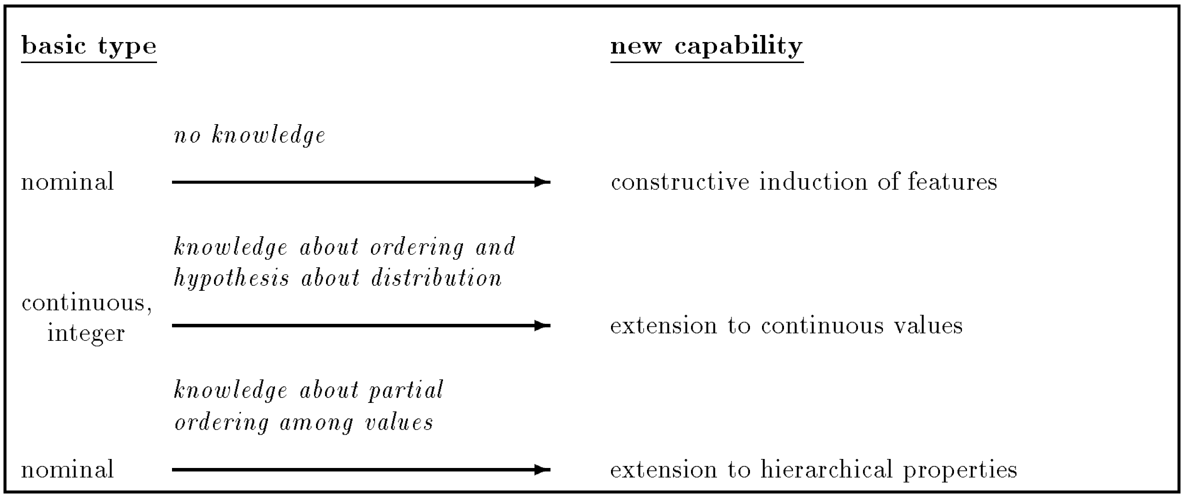

Figure 1: Knowledge and constructive induction

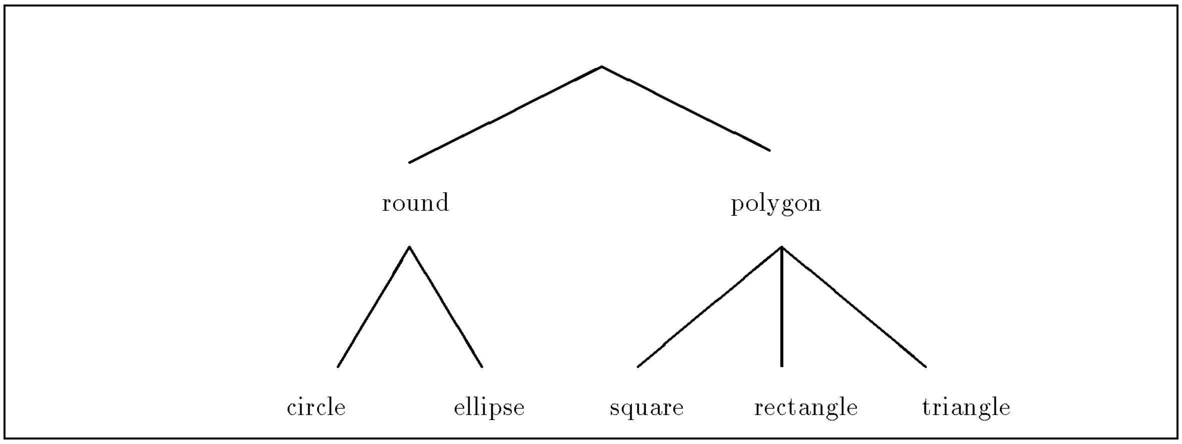

Figure 2: A structured property

set of learning operators than COBWEB has; it can forget undesired knowledge; and it can perform directed experimentation that increases the utility of its knowledge.

Bridger uses an incremental learning scheme for the creation of hierarchical classification trees. Bridger accepts a stream of examples described by lists of property-value pairs and builds a classification hierarchy in the following way. When a new example is introduced, Bridger tries to accommodate it into the existing classification starting from its root. The sub-classes of the root form the classification at this level of the tree. Bridger can perform one of several available operators to accommodate the new example into the hierarchy or/and modify the hierarchy. If the example has been accommodated into an existing sub-class (i.e., by using one of the operators), the process recurses with this class as the root of a new sub-tree. Bridger uses CU to select between the operators. The best operator is the one that results in a new classification that maximizes CU.

Bridger predicts missing properties of a new example using a mechanism similar to learning but allowing only one operator to apply. Bridger sorts the partial description of the example through the hierarchical classification to find the best host node for the example. Prediction progresses by assigning the partial description characteristic property-value pairs describing the nodes traversed[3]. This strategy, which differs from the original method employed by COBWEB, can be viewed as a least-commitment method (Reich, 1990).

8. EXPERIMENTAL STUDIES

The constructive induction scheme was tested in two domains: The domain of Pittsburgh bridges (Reich and Fenves, 1992b) and the mushrooms domain[4].

Pittsburgh bridge domain.
This domain is a difficult design domain with 108 examples. Each example is described by twelve property-value pairs. The number of examples is too few to cover the domain considering that for each example, five properties need to be predicted. This is in contrast to classification domains where only one property class is being predicted. We tested the system in a performance test using 10 runs of the *10-fold cross-validation* experiment (Breiman et al, 1984). The results were as follows. With all the properties as nominal values (continuous and ordinal values were discretized by an expert in the domain) the predictive accuracy summarized over the 5 properties was 64.6%. The performance with the original continuous properties, using the extension to continuous properties was 67.6%. When adding CI over all the nominal properties, the performance dropped to 61.9%. The statistical differences between the three measures is significant beyond the 0.01 level.

Compared to the learning performance with combined continuous and nominal properties, the performance after discretization is inferior to the original results. In this case the constructive induction enhances the performance. Similar good performance of the extension to continuous properties has been obtained in other domains (Reich, 1989; Reich, 1991c).

An analysis of the features generated in the last experiment reveals that many features do not have, and even violate, technical meaning. Since the data is sparse, the greedy scheme overfits it by generating every feature that locally improves CU [5]. Using knowledge to restrict the possible features, as illustrated in Section 6, would prevent the construction of these arbitrary features and would generate appropriate features as seen in the case of the continuous

[3]Characteristics are property values that satisfy: $P(A_i = V_{ij}|C_k) \geq threshold$ and $P(C_k|A_i = V_{ij}) \geq threshold$, where *threshold* is a pre-determined fixed value. Potentially, this value can be learned for each domain, but usually it is fixed to 0.75.

[4]Available from the repository of learning databases at the University of California, Irvine.

[5]This behavior is similar to the overfitting of noisy data by concept learning programs (Quinlan, 1986).

properties.

Mushrooms domain.
This domain is relatively easy to learn. The original Bridger achieves better than 99% predictive accuracy over various samples of the database. Essentially, there is no reason to use CI for this task. However, since the 99% accuracy is the converged value, we can compare the learning curves of the programs with and without CI. Bridger with CI converges more rapidly to the upper performance level than the original Bridger. For example, the number of examples to obtain 95% accuracy for a sample of 500 examples is 60 for the original Bridger and 30 for Bridger with the constructive induction scheme. The number of examples to achieve 98% is around 200 for the original Bridger and only 90 for Bridger with the constructive induction scheme. These results show that for a simple and well behaved domain, the greedy approach improves the convergence rate considerably. The ability to create new features allows the attainment of good performance with less data.

The experiments with the two domains show that the success of the greedy CI depends on the nature of the domain and on the knowledge provided to the scheme. This leads to a more general framework for constructive induction that is illustrated in the next section.

9. A GENERAL FRAMEWORK OF CONSTRUCTIVE INDUCTION

The previous sections illustrated a basic behavior of CU and the design of a greedy constructive induction algorithm based on this behavior. This algorithm can serve as the foundation for a family of constructive strategies that are illustrated below. Three dimensions characterize this family.

Greedy versus lazy constructive induction.
The greedy approach integrates CI into the learning process. Every opportunity for generating features is used and possibly results in overfitting the data. A *lazy* approach would constrain this nature. For example, a new feature will be generated only if it increases CU above a certain threshold. The trigger to use CI would be an evidence that the current performance over several examples is poor. Another method that constrains the use of CI is the generation of features based on evidence accumulated from several examples. This strategy, for the concept learning program UNIMEM, has been termed deferred learning (Lebowitz, 1988).

Local versus global constructive induction.
The current *local* scheme results in the creation of different features for different levels of the concept hierarchy. The traditional *global* view of CI is that new features are shared between all example descriptions. Mapping it to our representation, features should be shared between different hierarchy levels. One can implement this by collecting statistics on the quality of different groupings of values to features while learning or predicting a new example. The proposal of a new feature is then based on the accumulated evidence from all the levels of the

hierarchy. The new grouping will be the one that maximizes CU across all the levels of the hierarchy.

Constrained versus complete constructive induction.
The current *constrained* scheme tries to create features from values of the same property. This scheme can be extended to a *complete* scheme that allows the combination of properties as well. This mechanism will treat values from different properties similarly. The size of the set that would serve as ground values is the summation of the number of values of all the properties. A simple instance of this will allow the performance of boolean feature discovery (Pagallo and Haussler, 1990). With additional knowledge this scheme can be used to naturally learn from hierarchical description of objects.

10. SUMMARY

We have described a general framework for designing constructive induction mechanisms for an incremental concept formation system. The basis of the framework is a specific type of learning bias of the category utility function. We described in detail the development, implementation and testing of the most basic instance of this framework. We have illustrated the benefits from this method and pointed to some issues involved in the use of CI in concept formation. In our particular case, CI allows to extend the applicability and scope of Bridger to many domains and tasks. To this end, Bridger is the only incremental concept formation program who has a general scheme of constructive induction.

Future work includes the completion of the implementation and testing it in additional domains. An important consequence of these tests is the identification of conditions for appropriate use of CI depending on the nature of a particular domain.

Acknowledgments

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. I would like to thank Steven J. Fenves for his support and discussions on this research.

References

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*, Belmont, Waldsworth, CA.

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*, 40(1-3):11–61.

Gluck, M. and Corter, J. (1985). "Information, uncertainty, and the utility of categories." In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society, Irvine, CA*, pages 283–287, San Mateo, CA, Academic Press.

Langley, P., Simon, H. A., Bradshaw, G. L., and Zytkow, J. M. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes*, MIT Press, Cambridge, MA.

Lebowitz, M. (1988). "Deferred commitment in unimem: waiting to learn." In *Proceedings of The International Conference on Machine Learning, Ann Arbor, MI*, pages 80–86, San Mateo, CA, Morgan Kaufmann.

Michalski, R. S. and Stepp, R. (1983). "Learning from observation: Conceptual clustering." In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 331–363, Palo Alto, CA, Tioga Press.

Michalski, R. S. (1983). "A theory and methodology of inductive learning." *Artificial Intelligence*, 20(2):111–161.

Pagallo, G. and Haussler, D. (1990). "Boolean feature discovery in empirical learning." *Machine Learning*, 5(1):71–99.

Quinlan, J. R. (1985). "Decision trees and multi-valued attributes." Technical Report 85.4, New South Wales Institute of Technology, New South Wales.

Quinlan, J. R. (1986). "The effect of noise on concept learning." In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach, Vol. II*, pages 149–166, Morgan Kaufmann, Los Altos, CA.

Reich, Y. and Fenves, S. J. (1992). "Inductive learning of bridge design knowledge." In Arciszewski, T. and Rossman, L. A., editors, *Knowledge Acquisition in Civil Engineering*, pages 169–189, American Society of Civil Engineers, New York, NY.

Reich, Y. and Fenves, S. J. (1992). "Inductive learning of synthesis knowledge." *International Journal of Expert Systems: Research and Applications*, 5(4):275–297.

Reich, Y. (1989). "Combining nominal and continuous properties in an incremental learning system for design." Technical Report EDRC-12-33-89, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

Reich, Y. (1990). "Converging to "Ideal" design knowledge by learning." In Fitzhorn, P. A., editor, *Proceedings of The First International Workshop on Formal Methods in Engineering Design*, pages 330–349, Colorado Springs, Colorado.

Reich, Y. (1991). "Building and improving design systems: A machine learning approach." PhD thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA. (Available as Technical Report EDRC 02-16-91).

Reich, Y. (1991). "Design knowledge acquisition: Task analysis and a partial implementation." *Knowledge Acquisition*, 3(3):237–254.

Reich, Y. (1991). "Incremental clustering with mixed property types." Technical Report EDRC-12-??-91, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

Rendell, L. (1988). "Learning hard concepts through constructive induction: Framework and rationale." Technical Report UIUUCDCS-R-88-1426, Department of Computer Science, University of Illinois as Urbana-Champaign, Urbana, Ill.

Wogulis, J. and Langley, P. (1989). "Improving efficiency by learning intermediate concepts." In *Proceedings of The Eleventh International Joint Conference on Artificial Intelligence*, pages 657–662, Detroit, MI, Morgan Kaufmann.