In *Machine Learning: A Multistrategy Approach, Vol. IV* (R. S. Michalski and G. Tecuci, eds.), pp. 379-401, Morgan Kaufmann, San Francisco, CA, 1994

Macro and Micro Perspectives of Multistrategy Learning

Yoram Reich
(Carnegie Mellon University)

Abstract

Machine learning techniques are perceived to have a great potential as means for the acquisition of knowledge; nevertheless, their use in complex engineering domains is still rare. Most machine learning techniques have been studied in the context of knowledge acquisition for well defined tasks, such as classification. Learning for these tasks can be handled by relatively simple algorithms. Complex domains present difficulties that can be approached by combining the strengths of several complementing learning techniques, and overcoming their weaknesses by providing alternative learning strategies. This study presents two perspectives, the *macro* and the *micro*, for viewing the issue of multistrategy learning. The macro perspective deals with the decomposition of an overall complex learning task into relatively well-defined learning tasks, and the micro perspective deals with designing multistrategy learning techniques for supporting the acquisition of knowledge for each task. The two perspectives are discussed in the context of BRIDGER a system that learns to design bridges.

1 Introduction

Real world engineering problems, in particular design[1], are very complex; they involve the execution of many tasks in rich knowledge environments with limited resources. Knowledge-based systems are expected to alleviate the complexity of solving engineering tasks. An effective support for the construction of such systems is crucial to their dissemination in practice. Machine learning has the potential of assisting in this construction.

Several previous attempts at matching learning techniques to the acquisition of design knowledge have failed (Shalin et al, 1988; Witten and MacDonald, 1988). The main reason for this failure is the attempt to identify a single technique that will suffice to support knowledge acquisition for design *as a whole*. Design, however, is neither a single process nor a static one. The design of learning techniques for such a complex activity requires its decomposition into simpler tasks and the matching of learning techniques for the acquisition of knowledge for each task. Usually, even this matching difficult since most machine learning techniques use only one strategy that can partially match the requirements of real learning tasks (Tecuci and Michalski, 1991).

Since single strategy learning approaches are not sufficient for supporting learning in real domains, multistrategy approaches must be employed. Two perspectives of using multistrategy learning techniques emerge: the *macro* and the *micro*, elaborated below.

The macro perspective deals with the design of an architecture that learns and problem-solves in the context of a large engineering task. It assumes that it is not feasible to expect that a single

[1]The discussion in this paper is in the context of design, but it equally applies to other tasks and domains.

learning program will acquire all knowledge in a real engineering domain. Therefore, an architecture that supports this task is bound to include several learning programs and control mechanisms that direct learning tasks to the appropriate programs. Building such a learning system is a complex design task by itself. Its successful completion depends on following a systematic approach. The approach proposed in this study is called M^2LTD (Matching Machine Learning To Design). It is detailed in Section 2.1, including an example of its use to specify a learning system for a specific design domain.

Several recent studies incorporated macro-perspective issues in their system implementation. Stirling and Buntine (1988) investigated learning routings of products in a plant. They decomposed the learning task into two subtasks. In the first task, descriptions of routings of products in a mill are used to induce a grammar; which can then be modified by an expert. In the second task, examples are used by **ID3** (Quinlan, 1986) to induce routing decisions which are expressed as different branches in the grammar. The rules generated can then be evaluated by an expert. The grammar induction was necessary for the execution of the second learning task.

Bergadano, Giordana, and Saitta (1988) described a learning program for noisy domains that uses one module to extract object descriptions from data and makes these description available to a concept learner. Again, the two modules are necessary for performing the learning task.

Lu and Chen (1987) described a method for learning evaluations in a manufacturing domain. In the first stage the behavior of objects is clustered by CLUSTER/2 (Michalski and Stepp, 1983). Then, each class is treated as a concept to be learned by the concept learning program **AQ15** (Michalski et al, 1986). The rules generated by **AQ15** can assign a new object to the class of behaviors to which it belongs. The two learning programs are both required to perform the overall learning task.

In all these studies, the macro perspective deals with the use of distinct learning programs, each for a specific aspect of the learning problem. The control of the programs is determined by the problem requirements, and is fixed.

The micro perspective deals with the design of specific learning procedures for the acquisition of knowledge for well defined[2] engineering tasks. In real domains, these procedures are bound to employ several learning strategies, operating in an integrated manner. The crucial issue in this perspective is the control of and cooperation between the different strategies.

Several recent studies employ multistrategy learning. MTL (Tecuci and Michalski, 1991) uses deduction, analogy, abduction, and induction, to justify an input and potentially learn new knowledge from observations. Although currently the control of these strategies is fixed, future extensions will allow the dynamic execution of the strategies.

ML-SMART (Bergadano et al, 1989) views learning as a problem-solving activity. It uses knowledge and different strategies, and allows the user to specify which strategies are applicable for the particular application domain.

AIMS (Lu and Tcheng, 1991) supports the dynamic selection of learning techniques in the recursive-splitting learning paradigm. The system divides the domain based on some criteria, such as information gain, and selects a technique to further learn the subdomain based on the desired accuracy, comprehensibility, and efficiency of the learned knowledge and the characteristics of the available techniques.

LIMES (Herrmann, 1991) employs several learning techniques for learning estimation knowledge. The techniques used are generalizations, specializations, graph annotation (used in prediction), and class addition. The strategies are used in response to problem solving scenarios.

[2] Note that any engineering task is usually complex. However, at this level of granularity, we make the assumption that the problem can be formulated such that an algorithm can solve it.

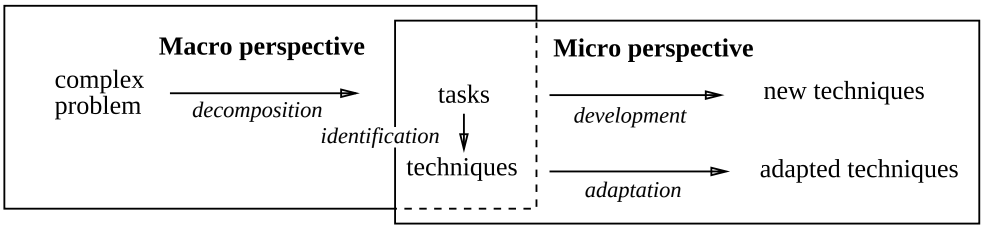

Figure 1: The macro and micro perspectives

Neither the macro nor the micro perspective are sufficient for supporting the knowledge acquisition for real engineering domains; both are equally important. Figure 1 illustrates the relationship between the two perspectives. The macro perspective decomposes the problem into simple tasks and identifies machine learning techniques that best, but probably not fully, match the specific engineering tasks. The micro perspective deals with the development of techniques to better suit the well-defined tasks or the adaptation of existing techniques identified by the macro perspective. Usually, the scope of each technique will be restricted, necessitating the use of several multistrategy programs for supporting a real domain.

Two phases in the development of domain models, an idea similar to the macro and micro perspectives, is discussed by Tecuci (1991): (1) the definition of a suitable framework and (2) the implementation of the model in the framework. These phases correspond to the macro and micro perspectives, respectively.

Plan of the paper. The remainder of the paper is organized as follows. Section 2 discusses the macro perspective of multistrategy learning. It reviews M^{2}LTD and illustrates how BRIDGER, a system that assists in the design of cable-stayed bridges, was developed following M^{2}LTD guidelines. Section 3 briefly reviews COBWEB and PROTOS. It discusses their shortcomings in relation to engineering domains and outlines some of the extensions that introduce additional learning strategies into these systems. These extensions are implemented in ECOBWEB, and EPROTOS respectively. Section 4 provides a common framework for the two perspectives by revisiting the concept of generic-learning tasks and showing that both perspectives operate within the generic task framework, but in different grain sizes. Section 5 summarizes the paper and discusses future research directions.

2 The Macro Perspective

This section describes M^{2}LTD, the systematic approach for identifying learning techniques for complex domains, and reviews BRIDGER, the system built following the principles of M^{2}LTD.

2.1 M^{2}LTD

M^{2}LTD (Matching Machine Learning To Design) is a *manual* systematic approach for designing systems that acquire knowledge in complex domains (Reich, 1991b; Reich, 1991c); it reflects the macro perspective of multistrategy learning. As any design activity, the use of M^{2}LTD does not guarantee a successful development of a learning system; rather, it helps identifying *potential* learning techniques that may perform the overall learning task. Feedback from implementation or other concerns emerging in the integration of the different techniques, may dictate the modification of the techniques identified or even their replacements. The collection of experiences using M^{2}LTD is important to its future use. In that sense, the learning in this perspective is done by the developers of large learning systems.

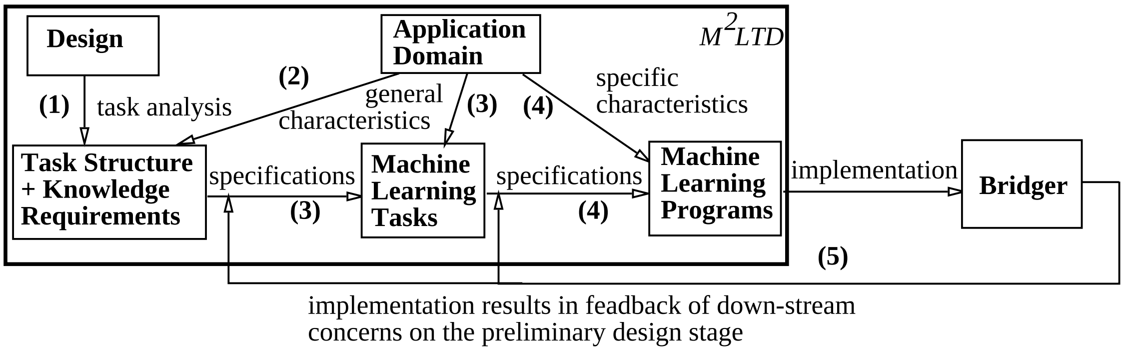

Figure 2: M^{2}LTD: Matching Machine Learning programs To Design tasks

M^{2}LTD is based on four steps (Figure 2):

1. decomposition of the design task into a collection of smaller tasks (e.g., synthesis or analysis);
2. identification of the representation of design objects used (e.g., lists of attribute-value pairs, trees or graphs) in each of the tasks described in Step (1), and the strategies each task uses (e.g., top-down refinement);
3. selection of machine learning paradigms that have the characteristics identified in Step (2) (e.g., concept formation or EBL); and
4. use of additional domain characteristics to select particular machine learning programs, from the collection available in each paradigm found in Step (3), that can acquire the knowledge in the right representation and support the strategies employed. These well-defined learning techniques are called *generic learning tasks* (Reich and Fenves (1989); See also Section 4).

Rarely will an existing machine learning program do the task as specified; but a close match eliminates the effort of building a new learning program, leaving the need for modifying an existing one. M^{2}LTD focuses the implementation effort on the important modifications required, which in many cases involve the addition of new learning strategies.

Figure missing

Figure 3: Matching learning programs to design tasks

Many concerns enter the decision about the appropriate machine learning task to use for each problem task. Some of the decisions are not conclusive and require revisions when the system is implemented. This is shown as Step (5) in Figure 2. Some of the concerns are mentioned in the next section which discusses Bridger, a system that demonstrates the application of M^{2}LTD in the design of a learning system.

2.2 Bridger

2.2.1 Uses of M²LTD

This section describes the design of Bridger, a system that assists in the preliminary design of cable-stayed bridges. In Step (1) of M²LTD, preliminary design was decomposed into several tasks executed sequentially: synthesis, analysis, redesign, and evaluation (see Figure 3a). In Step (2), the domain of cable-stayed bridge design was analyzed based on the tasks previously identified, and resulted in the following observations:

Synthesis. There is no explicit knowledge about synthesis of cable-stayed bridges. Rather, synthesis uses the collection of previously designed bridges and adapts them to new specifications. Cable-stayed bridges can be described by an elaborate list of property-value pairs. A list sufficient for representing preliminary designs, drawn from existing bridges, contains about 60 properties.

Analysis. There is extensive knowledge about analysis of bridges which is readily available in programs such as finite-element analysis.

Redesign. There is a collection of heuristics that can propose design modifications for particular deficiencies in candidate designs. These heuristics were generated from studies of bridge examples. The heuristics are not exact and may conflict; therefore, can be viewed as a weak domain knowledge.

Evaluation. Subjective judgment is usually used in selecting between candidate designs that satisfy the design requirements and the relevant design codes. In particular, aesthetic criteria have a major impact on evaluation.

The above observations direct the selection of learning tasks (Step (3) in Figure 2 and Figure 3b) and programs (Step (4) in Figure 2 and Figure 3c) for the acquisition of knowledge in the domain of cable-stayed bridge design. Both steps are discussed next.

Synthesis is a process that generates a description of an artifact given a list of specifications. In cable-stayed bridge design, both specifications and artifact descriptions can be represented by lists of property-value pairs (Reich, 1991b). A learning process that can capture a relation between two sets of properties is concept formation, which is believed to be fundamental for capturing synthesis knowledge (Reich and Fenves, 1991). To briefly illustrate this idea, Figure 4 shows the mappings possible when using concept learning: a mapping from several inputs to a single classification outcome, and concept formation: a mapping between several inputs to several outcomes; the latter is required for synthesis.

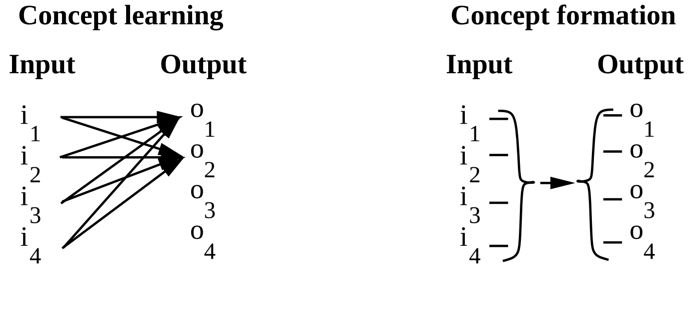

Figure 4: Concept learning versus formation in support of synthesis

The design domain characteristics determine the use of techniques that do not use knowledge since it does not exist[3]. Step (4) suggests that a program such as Cobweb (Fisher, 1987), can perform

[3] Later we show how, in fact, if knowledge exist, it can be used to enhance learning performance.

the learning activity. The reimplemention and testing of COBWEB on several design domains show some deficiencies leading to significant enhancements that are implemented in ECOBWEB (Enhanced COBWEB; see Section 3.1).

Analysis is coded directly, no learning is used in this task. Nevertheless, an experiment was performed to show that ECOBWEB can learn heuristic analysis (Reich and Fenves, 1991).

Redesign is a diagnosis task which can be supported by concept learning. The characteristics of cable-stayed bridge design allow the use of weak domain knowledge. A learning program that supports such learning activity is PROTOS (Bareiss, 1989).

Evaluation is not explicitly captured as knowledge; nevertheless, candidate designs generated by BRIDGER that are chosen by the designer can be used as training examples for enhancing synthesis knowledge. Consequently, the user evaluation becomes an implicit part of synthesis.

2.2.2 BRIDGER's architecture

BRIDGER's architecture is based on the task analysis. BRIDGER contains two main subsystems: synthesis and redesign (see Figure 5). The synthesis system is responsible for synthesizing several candidates for a given specification. It is implemented by ECOBWEB (see Section 3.1). The redesign system is responsible for the analysis, redesign, and evaluation tasks of design. Candidate designs are analyzed by a finite-element program according to bridge design codes. The analysis results are compared with the code to generate constraint violations. These violations are input to the redesign module that retrieves the best design modification for the particular violations of the bridge. The user can override the redesign modifications and supply explanations that enhance redesign knowledge. The redesign module is implemented in EPROTOS (see Section 3.2). The results of the redesign system are acceptable designs.

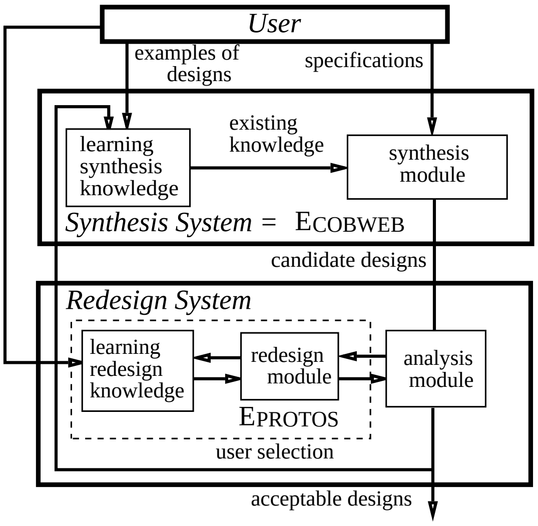

Figure 5: BRIDGER's architecture

One of the important issues in multistrategy learning is the interaction between the strategies or programs. In the context of BRIDGER, this question is translated into the nature of the interaction between synthesis (ECOBWEB) and redesign (EPROTOS).

In the macro perspective, the control, and therefore, the interaction is pre-specified and limited.

The different learning programs interact in the normal course of solving problems in the domain. In BRIDGER, the product of synthesis is delivered to analysis and redesign, therefore, the redesign knowledge acquired will be tuned to the candidates generated by the synthesis module. In addition, candidates modified by redesign may be learned by the synthesis module, therefore, enhancing its knowledge. As BRIDGER designs more bridges, the candidate generated by synthesis are better and closer to those that redesign can easily correct.

2.2.3 Additional uses of M^{2}LTD

Two other application of M^{2}LTD for specifying a system that learns to design ships and a system that learns to perform finite element modeling are discussed in (Reich, 1991b). Since the preliminary design of ships is similar to the preliminary design of bridges, the use of M^{2}LTD in this domain results in a specification of a program that is similar to BRIDGER. Two modifications to BRIDGER's architecture allow the acquisition of analysis and evaluation knowledge of ships. Due to the similarity between the bridge and the ship design domains, the use of M^{2}LTD for the domain of ship design is almost guaranteed to be successful. The third domain, the design of finite-element models, is completely different than the first two domains. The use of M^{2}LTD for this domain results in a rough specification of a learning program. A successful implementation requires the modification of several machine learning programs and substantial testing and further programs adaptation.

3 The Micro Perspective

This section discusses the micro perspective of multistrategy learning in the context of the systems that implement the synthesis and redesign modules of BRIDGER: ECOBWEB and EPROTOS. ECOBWEB includes several extensions that make it a multistrategy learning system; ECOBWEB is the major focus of this section. EPROTOS enhances PROTOS, which is already a multistrategy learning system; its operation is described, including two new extensions. Due to space limitation, the description assumes a certain level of familiarity with the original COBWEB and PROTOS.

3.1 Ecobweb

COBWEB is a concept formation program for the creation of hierarchical classification trees (Fisher, 1987). A classification is 'good' if the description of an example can be guessed with high accuracy, given that it belongs to a specific class.

COBWEB evaluates a classification of a set of examples into mutually-exclusive classes $C_1, C_2, \ldots, C_n$ by a statistical function 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. The first term in the numerator measures the expected number of 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 category utility measures the *increase* of property-value pairs that can be guessed *above* the guess based on frequency alone. The measurement is normalized with respect to the number of classes.

When a new example is introduced, COBWEB tries to accommodate it into an existing hierarchy starting at the root. The system can perform one of the following operators (See (Fisher, 1987) for a detailed description):

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

COBWEB selects the operator that its application maximizes CU. If the example has been assimilated into an existing sub-class, the process recurses with this class as the top of a new hierarchy.

COBWEB predicts using a mechanism similar to the one used for augmenting the hierarchy by new examples but allowing only operator 3 to apply. COBWEB sorts a partial example through the hierarchy to find the best host for the new example. The best host is a leaf node (i.e., one of the training examples) that is used to complete the description of the new example.

COBWEB has a number of drawbacks for its use in an engineering design domain.

1. COBWEB can only manipulate nominal properties. CLASSIT is a descendant of COBWEB that handles continuous properties (Gennari et al, 1989). Elsewhere, we contrast its approach with the extension to continuous properties implemented in ECOBWEB (Reich, 1991b) and conclude that ECOBWEB's strategy is more flexible.
2. COBWEB has a stiff prediction scheme. It makes a strong commitment to continue prediction until a complete existing design is retrieved. No generation of new designs is possible. In addition, leaf prediction is not adequate since it produces only one candidate in each prediction. Leaf prediction is also susceptible to noise and may result in high error rates.
3. COBWEB uses only the syntactic measure of category utility to guide its learning and prediction. No domain dependent or independent knowledge is used, although if available, could enhance learning substantially.
4. COBWEB's performance depends substantially on the order of example presentation, even though it has two learning operators, split and merge, that are specifically designed to reduce order effects on learning.

Extensions that address each of the deficiencies are described next. They all involve some aspect of using background knowledge or additional learning strategies. Since ECOBWEB prediction is interleaved with its learning, extensions to prediction methods are also important and discussed next.

3.1.1 Extension to continuous properties

As a system that learns from bridge examples, ECOBWEB must handle the continuous, ordinal, or nominal property types that appear in bridge descriptions. ECOBWEB implements an extension of CU that can handle continuous properties (Reich and Fenves, 1991; Reich, 1991b). In its simplest variant the term $\sum_j P(A_i = V_{ij}|C_k)^2$ is calculated as:

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

where $p_i(x)$ is a distribution for the A_i, $2d_i$ is dependent on the range of values of 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. If no specific knowledge is available about the distribution p_i, the default is the normal distribution. This extension has been tested extensively in several domains and has proven to be effective and relatively insensitive to the choice of d_i (Reich, 1991b).

Knowledge can be used in two ways. The first is the use of domain knowledge about the distribution of continuous properties. The second way is the inspection of *a posteriori* distributions of continuous property values and the revision of their distributions based on this inspection. The first way is fully implemented and the second can be incorporated in Ecobweb.

3.1.2 Extension of the prediction methods

Cobweb's prediction method is too restrictive. Also, it is not clear how an inductive learner of the Cobweb type performs without a rich set of bridge examples. The new prediction methods are designed to operate between a case-based approach, when few examples are available, to a prototype-based approach when many examples are present.

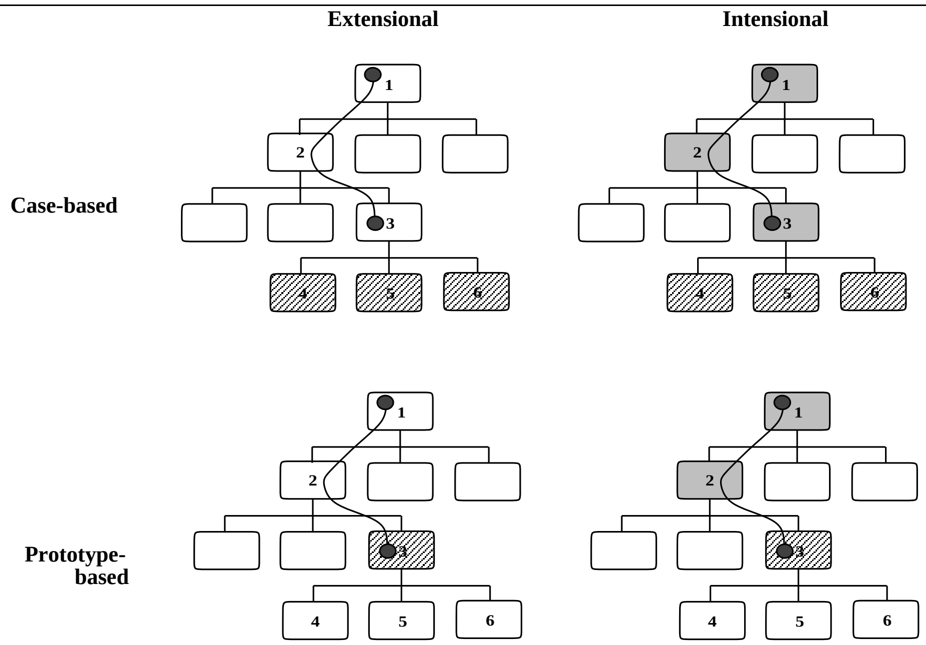

Figure 6: Synthesis methods

The basic procedure is similar to Cobweb's prediction: CU guides the sorting of a partial example down the hierarchy. The first modification is that at each node, the property values of the example are compared to the characteristic[4] values of the node. A similar value is called a match. When all the property values of the example are matched, the sorting process terminates, potentially at an intermediate node in the hierarchy. To illustrate, assume that the problem is specified by two requirements, and that one of the characteristic values of class 1 is the same as one requirement, and one of the characteristics of class 3, equals the other requirement. In this situation, the path from class 1 to 3 matches the two requirements and therefore, the synthesis process terminates.

[4]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.

This behavior generates general solutions to general problems, a good by-product of the design of the new prediction methods.

The second difference between the new and COBWEB's prediction method is the variety of techniques available in the new prediction. ECOBWEB's prediction methods can be described along two dimensions: the refinement (i.e., sorting) process which can be *extensional* or *intentional*; and the generation process which can be *case-based* or *prototype-based.* Figure 6 illustrates these dimensions. In the extensional approach, refinement classifies a new example with a new subclass starting from the top node (class 1 in Figure 6) until the process terminates (class 3). In this view, a class represents the extension of all its leaves. In the intentional approach, while classifying the new example, characteristic property-values of the classes traversed (classes 1, 2, and 3 in Figure 6) are assigned to the new design. In the case-based approach, a set of existing designs is retrieved as candidate designs. For example, designs 4, 5, and 6 are retrieved. In the prototype-based approach, the last class, i.e. class 3, is used to generate several candidates from the property value data it contains.

3.1.3 Use of knowledge in concept formation

Use of knowledge to override CU. An important characteristic of the knowledge generated by ECOBWEB is that it is declarative. This enables an external body of knowledge, domain dependent or independent, to inspect it. This inspection can make inferences that enhance the use of the classification hierarchy in learning and prediction. The ability to benefit from external knowledge relies on the flexible nature of the learning method which uses weak search methods directed by the category utility function. CU can be used as a default mechanism in the absence of knowledge; explicit knowledge, on the other hand, can prefer the execution of a specific operator or other learning strategies and override CU.

An example of constraining learning and prediction by Pareto-optimality and hard constraints is given in Figure 7. The top level of the classification hierarchy consists of four classes. Each class is partially described by three characteristic objectives; all the remaining properties are irrelevant to this discussion. In learning or prediction, instead of trying to find the best host from all the sub-classes, only sub-classes that are Pareto-optimal with respect to the desired objectives $(C_2, \ldots, C_k, \ldots, C_n)$ and that satisfy all the hard-constraints $(C_k, \ldots, C_n)$ are considered. Consequently, C_1 is excluded since it is not optimal and violates the constraints, and C_2 is excluded since it violates the constraint. Alternatively, given additional resources, exploratory search might take place by relaxing constraints (permitting the consideration of C_2), or by trading objectives with other considerations (permitting the consideration of C_1) .

Use of knowledge about modeling. Another kind of domain knowledge that is investigated is knowledge about modeling. For example, such knowledge can suggest important features of objects. Given an initial description of objects, the knowledge can augment it by additional properties, compress the description of several properties, or erase irrelevant properties.

A simple use of such knowledge is exercised in modifying Equation 1 to account for the *a priori* or *learned* differences in the relative importance of properties. Equation 3 shows how CU can accommodate knowledge about the relative importance of properties and the structure of the classification hierarchy:

$$\frac{\sum_{k=1}^{n} P(C_k) \sum_i \sum_j W_i \, P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j W_i \, P(A_i = V_{ij})^2}{n^\alpha} \quad (3)$$

where W_i is the weight of A_i, and $\alpha \geq 1.0$.

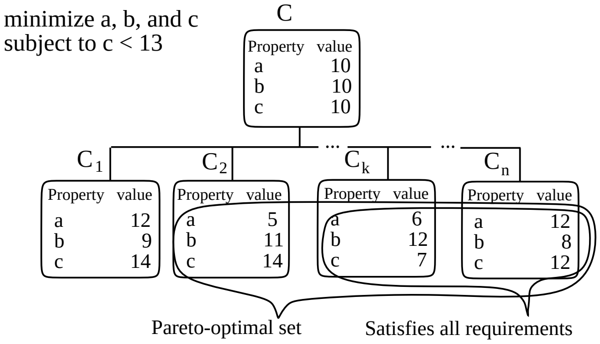

Figure 7: Constrained optimization and exploratory design

The parameters α makes sure that the classification hierarchy has a reasonable branching factor. For example, a flat hierarchy is computationally inefficient for the retrieval of design and a binary hierarchy looses information about similarity between designs. In the cable-stayed bridge design a flat hierarchy is generated when using the original category utility expression from Equation 1. The variation of the parameter α changes the hierarchy from binary, when α is large, to flat, when α equals 1.0. The value selected for the cable-stayed bridge domain is $\alpha = 1.1$. This value can be dynamically modified by observing the branching factor of the hierarchy. A branching factor of 3 to 4 is experienced to yield good results across domains. The knowledge about the weighting and the α parameter are required if data is sparse and not sufficient to construct a usable hierarchy. The need to use knowledge in empirical learning, when data is limited, is recognized in many learning systems. EPROTOS, the system used for the acquisition of redesign knowledge, is one example of such a system.

Constructive induction. COBWEB makes use of the original property-value pairs appearing in example descriptions, assuming that the description language is adequate. **ECOBWEB** has a constructive induction scheme that can generate higher-order features from existing property values (Reich, 1990). The extension to continuous properties can be viewed as grouping values into samples from an assumed distribution. For example, two 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.

The constructive induction capability can be used to handle hierarchical properties. In this case, the only features that will be allowed are those that represent nodes of the hierarchy. For example, $\{circle\}$, or $round = \{circle, ellipse\}$, shown in Figure 8, are possible features, whereas $\{square, rectangle\}$, or $\{triangle, circle\}$ are inappropriate features.

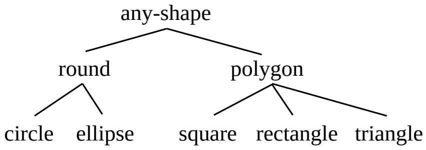

Figure 8: A hierarchical property

Experimentation. Another way of using knowledge is to construct examples that will enhance the performance of the system. This capability is very important since machine learning techniques are sometimes used for extracting knowledge from simulation of behaviors of designs which are examples that are generated on demand (Bratko et al, 1989; Buchanan et al, 1988; Reich, 1991b). The availability of good sources of examples is crucial for obtaining a satisfactory learning performance.

For example, in the context of BRIDGER, there are relatively few examples of cable-stayed bridges available and the generation of new examples is time consuming; it involves the selection of 'good' specification and the design of solutions for the specification. It is therefore important to devise a method that will allow the generation of the most useful examples, such that the improvement in knowledge due to learning is maximized with minimal training resources.

The problem of selecting good examples is especially dominant when using an incremental learning system (Fisher et al, 1991). In this case, even if all the examples are available, it is useful to impose a good ordering on the training examples (MacGregor, 1988). While using COBWEB (Fisher, 1987) in several design domains (Reich, 1991b) it has been observed that the order of training examples used to generate synthesis knowledge has a substantial effect on the synthesis ability. Deviation of 90% in synthesis performance were observed in statistical experiments with random orderings of training examples.

ECOBWEB's experimentation technique can be performed in two ways. The first method only makes use of the information stored at the root of the hierarchy, namely, the most frequent property-value pairs. The next training example that is selected is one that has property values that are the most distant from the most frequent values observed thus far.

The second experimentation method is more complex, it searches for an example that if learned, will maximize the category utility of the top-level classification. This method is more informative and makes use of ECOBWEB mechanisms. Preliminary experiments in several domains show some benefits from using the techniques but invariably this issue remains unsolved (Reich, 1991a).

Hierarchy-correction scheme. Another approach to mitigate the problem of order effects on learning is to use knowledge about properties to reorganize the hierarchical knowledge structure in a process called *hierarchy correction*. This procedure can detect problems in the classification hierarchy that were introduced in spite of the mechanisms that use knowledge to override CU, discussed before.

The hierarchy-correction scheme follows three steps. First, properties deemed most critical by domain knowledge are selected as *triggers*. Second, the hierarchy is traversed top-down. Each class with a characteristic value of a trigger property, that differs from a characteristic in one of the class' ancestors, is removed along with its subtree from the hierarchy. Third, the examples at the leaves of all the removed subtrees are re-learned. Since the process is heuristic, learning can reconstruct exactly the same tree after the subtrees are removed, in contrast the process can iterate several times, each leading to some improvement. The process stops when no change to the hierarchy is obtained or when the limit on the number of iterations is reached. The application of this procedure enhances the predictive accuracy of ECOBWEB.

To illustrate, assume that the top classification hierarchy is as appears in Figure 9. Spec$_1$ is a specification property and des$_1$ is a design description property that are considered triggers by domain knowledge. Only characteristic values are shown in the figure. The hierarchy correction traverses the hierarchy until reaching class C_1. By looking at its sub-classes, C_2, C_3, and C_4, the method detects that: the characteristic value of spec$_1$ in class C_2 is different than that in class C_1, the characteristic value of des$_1$ in class C_3 is different than that in class C_1, and both characteristic values in class C_4 are equal to those in class C_1. The differences suggest that the information stored in C_2 and C_3 contradict the information stored in the parent class C_1. It is better if this contradiction is removed. Therefore, classes C_2 and C_3 are erased from the hierarchy with all their sons, and the leaf nodes which are training examples are re-learned by **Ecobweb**.

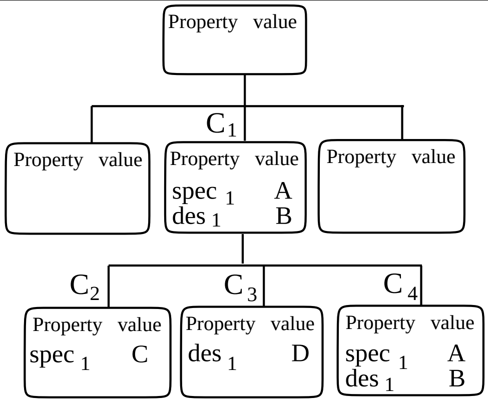

Figure 9: Classification hierarchy with characteristic contradictions

User role. Beside all the automatic techniques discussed above, that make use of some knowledge, there is the most trivial way of using knowledge in learning and prediction: the apprentice mode. In that mode, **Ecobweb** allows the user to determine the next learning operator or strategy to apply instead of using category utility or one of the additional mechanisms described. Similar ideas to the use of interactive induction, in which the user of the learning program observes the results of the program and modifies its input, are discussed in (Stirling and Buntine, 1988). It is expected that in complex domains, a synergistic approach of human-computer induction will be more beneficial than the fully automatic approach (Reich et al, 1991). Future extensions will allow the user incorporating knowledge while controlling **Ecobweb**'s operation.

Summary. This list of implemented and potential uses of knowledge or other learning strategies in the **Cobweb** framework is not exhaustive. Presently, the user of **Ecobweb** determines which learning strategies are used in a specific learning scenario. The automatic control of these strategies is complex and is the subject of future research.

3.2 Eprotos

Protos is an exemplar-based learning program that integrates empirical learning with weak domain knowledge in the form of explanation (Bareiss, 1989). Therefore, **Protos** is a multistrategy learning system. The task of **Protos** is heuristic classification: given a new case and knowledge in the form of category structure, find the best category for the new case. **Protos**' operation is detailed in the following paragraphs.

Protos represents knowledge in categories that contain representative cases called *exemplars*. Exemplars are described by lists of property-value pairs. The exemplars are augmented with domain knowledge, which explain why a certain exemplar is a member of its category. The explanation can also provide additional relations between other pieces of knowledge. The explanations are in the form of a network of predicates relating terms in the category structure. Each predicate is associated with several types of probability measures, such as strength or degree of belief. Every object in the category structure constitutes a term: the name of the category, the name of an exemplar, each of the properties or their values, or the predicates themselves. Such a representation allows the construction of an elaborate network from simple primitives.

The first process in Protos' algorithm is finding an exemplar that strongly matches the new case. The description of the new case serves as the indexing information. Protos can index the category structure using four types of indices: remindings, censors, prototypicality, and difference. Remindings associate properties with exemplars in the category structure. Remindings are combined to generate access to a particular exemplar. A censor leading from a property to an exemplar prevents access to that exemplar.

Remindings and censors are used in the hypothesis formation stage of classification. In this process the properties of a new case remind exemplars that are rated based on their combined remindings' strength. The category that contains the strongest exemplar is selected as the hypothesis of the classification. In the second stage of classification, hypothesis confirmation, prototypicality ratings of the exemplars are used to select the most prototypical exemplar from that category to further confirm the result.

In the confirmation process, the properties of the old exemplar are matched against the new case. Protos tries to obtain one-to-one matching between the properties. A property that is not matched triggers search through the category network to heuristically construct further matchings via the use of explanations. If Protos cannot find a suitable explanation between the old exemplar and the new case, it tries to fetch a better exemplar of the same category by using the difference links and then starts the matching process again.

Protos learns while interacting with an expert. Problems in either the hypothesis formation (classification) or the confirmation (explanation) trigger queries for the expert. The queries are focussed on the local problem that Protos has. The expert's answers are used to refine Protos' knowledge.

Indices are learned and strengthened in successful hypothesis formation. An exemplar is learned if it cannot be classified or if there is no good match between it and an existing exemplar. Explanations are solicited from the user about new terms or to explain a match between properties that Protos cannot pursue. Credit is assigned to exemplars that are used in a successful matching process.

In relation to design, Protos is used to acquire redesign knowledge and to redesign objects. In this case, exemplars and cases contain the description of bridges with some performance values and a redesign modification contains a list of properties that should be modified to correct the performance. Protos has several drawbacks for its use as a redesign system.

1. Protos can only manipulate symbolic properties. This problem was discussed before in relation to Cobweb. In engineering domains, numbers play a major role and must be handled effectively by Protos.

2. As a classification program, Protos outputs a single class as the confirmed hypothesis. In redesign, it may be necessary to output several redesign modifications that may be coupled as the redesign recommendation.

3. The explanation language cannot handle exact relationships between terms. For example, it cannot easily represent an equation.

The first two drawbacks are addressed in extensions implemented in Eprotos.

3.2.1 Extension to continuous properties

The original PROTOS cannot work with continuous or ordered properties. Both PROTOS' main mechanisms: (1) the retrieval of remindings to categories or exemplars; and (2) the comparison of a new case with an exemplar; must be modified to handle continuous properties.

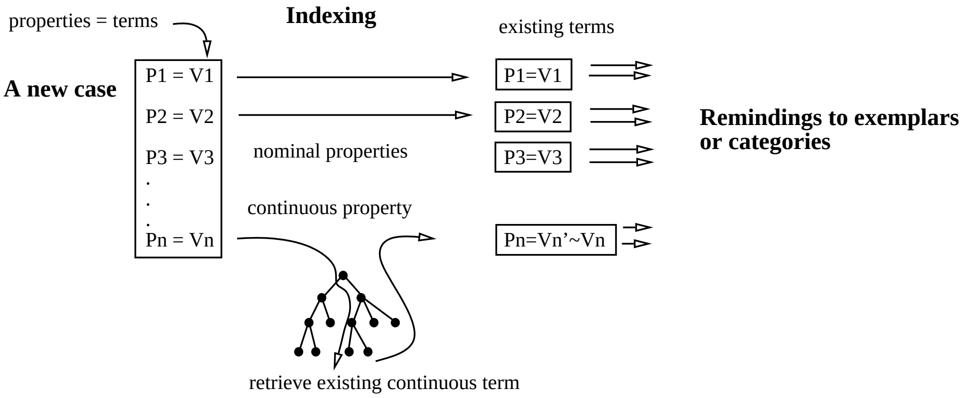

Figure 10: Indexing remindings with continuous properties

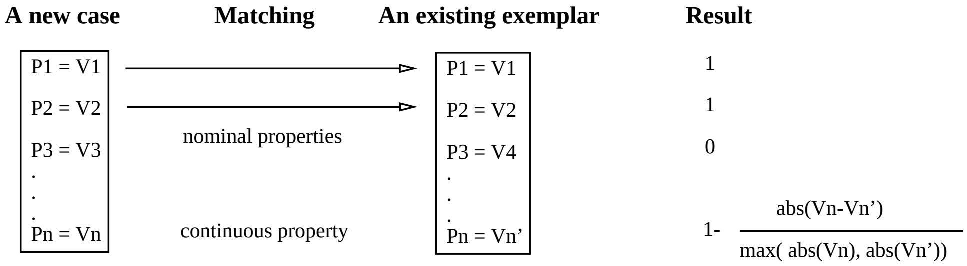

Figure 11: Comparison with continuous properties

Retrieval of remindings by continuous properties. When a new case is introduced, PROTOS tries to match the case's properties to existing terms in the category structure. A term is considered to be the property with its value. If a match exists, the remindings attached to it are triggered; otherwise, PROTOS does not recognize the property and it asks the user for relevant information. In the case of continuous properties, it is highly improbable that exactly the same value will be in memory. Consequently, PROTOS will not recognize the property-value pair.

To remedy this situation, an indexing mechanisms is created (see Figure 10). This mechanism takes a new property-value pair as input and outputs the most relevant property-value pair in memory. This functionality leads itself to the use of ECOBWEB as an implementation mechanism for the indexing. ECOBWEB is used to create a hierarchy from the cases in PROTOS' memory. This hierarchy is used to index existing continuous property-value pairs with one of the existing synthesis mechanisms. A single hierarchy is sufficient for all the continuous properties. A highly desirable consequence of this synergy between ECOBWEB and PROTOS is that gradually, the indexing hierarchy will assimilate the heuristics in PROTOS' category network, leading to better retrieval of relevant properties. In fact, the indexing hierarchy can be later used to predict redesign actions as well.

Comparison of cases with exemplars. After forming a hypothesis about the most probable category to which the new case belongs, PROTOS tries to match the case with an existing exemplar. A successful match verifies the hypothesis. This match is performed by a knowledge-based matcher that searches through the category structure. Before search is expanded, a test is performed to check whether there is a match between a case property-value pair and the term in the category network. This test faces the same problem as before: no match will be found between continuous properties. To remedy this problem, a match is calculated for continuous properties based on their values (see Figure 11). This match is easily calculated.

These modifications allow EPROTOS to handle continuous values. These modifications were not tested extensively as the extension to continuous properties in ECOBWEB; nevertheless, they were sufficient to support the processing required of EPROTOS.

3.2.2 Retrieval of several redesign modifications

PROTOS evaluates several hypotheses and ranks them. It only outputs the most favorable one based on the match between the new case and one of the exemplars in the category that represents the hypothesis.

A simple modification to the output of PROTOS allows the retrieval of all the hypotheses formed. The order of the hypotheses is based on the strengths of their remindings and confirmation. Although the user is responsible for determining the magnitude of continuous property modifications, the user can use the strength of the hypotheses as a recommendation to the relative variation in the redesign actions.

4 Generic Learning Tasks

Both the macro and the micro perspectives rely on similar principles. In both, it is important to know (1) what is the input and output of a learning program, (2) what is the representation of knowledge used and learned, and (3) how the program is executed. This information is used manually in the macro perspective for identifying learning programs for specific tasks; and could be used automatically in the micro perspective for selecting between learning strategies in a multistrategy learning program.

Similar information requirements emerge from Chandrasekaran's work on generic tasks as high level building blocks for expert systems design (Chandrasekaran, 1986). A generic task is a tuple $G = (I/O, KR, C)$. Its components are described as follows.

I/O is the Input/Output of the generic task. It represents the task functionality.

KR is the representation of knowledge required to achieve the task functionality.

C is the control structure of the task. It specifies the procedure for achieving the task functionality by operating on the knowledge representation.

This tuple is crafted such that the knowledge representation and control used in the task are directed toward providing the required Input/Output characteristics. The knowledge representation and control are to be designed such that their syntax matches their semantics. The syntax-semantics correspondence establishes a strong limitation on the task scope but makes it productive for a specific class of problems.

The concept of *generic learning tasks* is the application of the generic task idea to machine learning except that now the task is learning a specific type of knowledge that will be used by some generic task. To illustrate the concept, examples of generic learning tasks from the macro and the micro perspectives are discussed.

Macro generic task: ECOBWEB. ECOBWEB can be viewed as a generic learning task that acquires knowledge for a class of routine synthesis problems[5]. The information related to the task is:

I/O. The input to ECOBWEB is examples represented by lists of property-value pairs. The output is a classification hierarchy that can be used for synthesis.

KR. Learned knowledge is represented as a probabilistic declarative classification hierarchy. Background knowledge can be represented in a variety of forms.

C. ECOBWEB makes use of four learning operators and a heuristic control driven by CU. This approximates hill-climbing in the space of classification hierarchies. ECOBWEB exercises fixed control over the learning mechanisms that use domain knowledge. C also includes time and space complexity of all the learning procedures implemented in ECOBWEB.

Micro generic task: hierarchy-correction. The hierarchy-correction procedure can be viewed as a generic learning task that improves the quality of knowledge represented by a classification hierarchy. The information related to this task is:

I/O. The input to hierarchy-correction is a classification hierarchy and a list of constraints on the hierarchy. The output is a classification hierarchy that heuristically minimizes the constraint violations.

KR. This procedure represents the classification hierarchy as a probabilistic declarative structure. The constraints are represented by predicates.

C. Hierarchy-correction performs iterative constraint violation detection, forgetting and re-learning of the forgotten examples, until the hierarchy is not changed. This constitutes a hill-climbing in the space of classification hierarchies. The time and space complexity of this procedure is similar to ECOBWEB since it includes the execution of ECOBWEB for re-learning the forgotten examples and this is more difficult than detecting the nodes to be erased.

The hierarchy-correction procedure shows how easily can complex leaning behaviors be created. Although it is one of ECOBWEB's mechanisms, it can make use of all ECOBWEB's functionality, including itself, for executing one of its sub-processes.

In complex multistrategy learning programs, there will be several generic learning tasks that can provide the same functionality but differ in their accuracy, cost, etc. The control of the multistrategy system will have to consider the overall learning task requirements in determining which learning task will perform each function.

Preliminary experience with developing generic learning tasks, and their integration into larger systems, suggests that considerable knowledge about learning techniques is required to successfully employ multistrategy techniques in complex domains. Knowledge is not yet available that would allow the full automation of selecting appropriate learning tasks for specific problems. It is conjectured that a better use of multistrategy learning is facilitated through the use of the generic learning

[5]Of course, COBWEB can be viewed as a generic learning task that acquires knowledge for a class of classification problems.

task idea which reflects both the macro and the micro perspectives of multistrategy learning. This idea, in turn, requires the development of a better understanding of existing machine learning techniques. This understanding involves the ability to identify machine learning techniques that can support the knowledge acquisition for a particular domain (i.e., have the right I/O) and support the problem-solving strategies that manipulate knowledge in that domain (i.e., have the right KR). In general, this understanding involves the construction of a mapping as shown in Figure 12[6]. A step towards this end is the development of language to describe the input and output of various learning algorithms called CKRL (Morik et al., this volume).

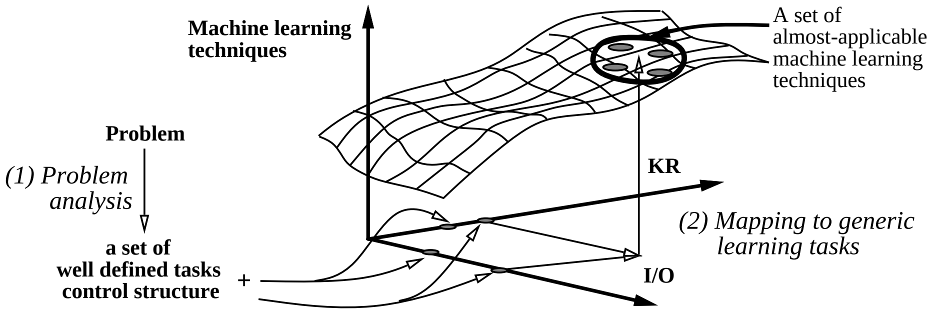

Figure 12: Mapping problems into machine learning techniques

5 Discussion and Summary

This paper discussed two perspectives of viewing multistrategy learning systems: the macro and the micro. The macro perspective deals with the decomposition of large learning problems into manageable pieces, and the selection of state-of-the-art learning techniques that can address these simpler learning problems. Since rarely will existing techniques satisfy the requirements of real world learning problems, the specification of new techniques are developed. M^2LTD is a manual procedure that guides the designer of a large learning program in the above decomposition, selection, and specification procedures. One example of the use of M^2LTD is the design of Bridger, a system that assists in the design of cable-stayed bridges.

The micro perspective deals with the design of multistrategy learning programs that must cope with the simpler learning tasks identified by M^2LTD. Ecobweb and Eprotos are two multistrategy systems that are incorporated in Bridger.

The problem of multistrategy learning is not building the mechanisms but their integration and control. The control aspect requires knowledge about the learning strategies: their performance, functionality, representation, accuracy, etc. For multistrategy learning to successfully work, a good understanding of machine learning techniques must be developed.

The automatic use of multiple learning strategies and programs for solving hard learning problems is an important problem that begins to attract new research effort. It is conjectured that the development of a better understanding of generic learning tasks will be beneficial in advancing the use of multistrategy learning.

Acknowledgments

This work has supported in part by the Engineering Design Research Center, a National Science

[6]Although the surface is shown to be continuous, it is in fact discrete.

Foundation Engineering Research Center.

References

Bareiss, R. (1989). *Exemplar-Based Knowledge Acquisition*, Academic Press, Boston, MA.

Bergadano, F. Giordana, A. and Saitta, L. (1988). "Automated concept acquisition in noisy environments." *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 10(4):555–578.

Bergadano, F., Gemello, R., Giordana, A., and Saitta, L. (1989). "ML-SMART: A problem solver for learning from examples." *Fundamenta Informaticae*, XII(1):29–50.

Bratko, I., Mozetič, I., and Lavrač, N. (1989). *KARDIO: A Study in Deep and Qualitative Knowledge for Expert Systems*, MIT Press, Boston, MA.

Buchanan, B. G., Sulivan, J., Cheng, T.-P., and Clearwater, S. H. (1988). "Simulation-assisted inductive learning." In *Proceedings of AAAI-88*, pages 552–557, St. Paul, Minnesota, Morgan Kaufmann.

Chandrasekaran, B. (1986). "Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design." *IEEE Expert*, 1(3):23–30.

Fisher, D., Ling, X., Carnes, R., Reich, Y., Fenves, S., Chen, J., Shiavi, R., Biswas, G., and Weinberg, J. (1991). "Selected applications of an AI clustering technique to engineering tasks." Technical Report CS-91-08, Vanderbilt University, Nashville, TN.

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.

Herrmann, J. (1991). "Learning analytical knowledge about VLSI-design from observations." In Birnbaum, L. A. and Collins, G., editors, *Proceedings of the Eight International Workshop on Machine Learning (Evanston, IL)*, pages 610–614, San Mateo, CA., Morgan Kaufmann.

Lu, S. C.-Y. and Chen, K. (1987). "A machine learning approach to the automatic synthesis of mechanistic knowledge for engineering decision-making." *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 1(2):109–118.

Lu, S. C.-Y. and Tcheng, D. K. (1991). "Building layered models to support engineering decision making: a machine learning approach." *Journal of Engineering for Industry, Transactions of the ASME*, 113(1):1–9.

MacGregor, J. N. (1988). "The effects of order on learning classifications by example: heuristics for finding the optimal order." *Artificial Intelligence*, 34(3):361–370, April.

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., Mozetič, I., Hong, J., and Lavrač, N. (1986). "The multi-purpose incremental learning system AQ15 and its testing application to three medical domains." In *Proceedings of AAAI-86 (Philadelphia, PA)*, pages 1041–1045, San Mateo, CA, Morgan Kaufmann.

Quinlan, J. R. (1986). "Induction of decision trees." *Machine Learning*, 1(1):81–106.

Reich, Y. and Fenves, S. J. (1989). "Integration of generic learning tasks." Technical Report EDRC 12-28-89, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

Reich, Y. and Fenves, S. J. (1991). "The formation and use of abstract concepts in design." In Fisher, D. H. J., Pazzani, M. J., and Langley, P., editors, *Concept Formation: Knowledge and Experience in Unsupervised Learning*, pages 323–353, Los Altos, CA, Morgan Kaufmann.

Reich, Y., Coyne, R., Modi, A., Steier, D., and Subrahmanian, E. (1991). "Learning in design: An EDRC (US) perspective." In Gero, J., editor, *Artificial Intelligence in Design'91, Proceedings of The First International Conference on Artificial Intelligence in Design, Edinburgh, UK*, pages 303–321, Oxford, UK, Butterworths.

Reich, Y. (1990). "Constructive induction by incremental concept formation." In Feldman, Y. A. and Bruckstein, A., editors, *Proceedings of the Seventh Israeli Symposium on Artificial Intelligence and Computer Vision (Ramat-Gan)*, pages 195–208, Amsterdam, Elsevier Science Publishers.

Reich, Y. (1991). "Automatic selection of examples for training a learning design system." Technical Report 12-42-91, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

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.

Shalin, V. L., Wisniewski, E. J., Levi, K. R., and Scott, P. D. (1988). "A formal analysis of machine learning systems for knowledge acquisition." *International Journal of Man-Machine Studies*, 29(4):429–446.

Stirling, D. and Buntine, W. (1988). "Process routings in a steel mill: a challenging induction problem." In Gero, J. S. and Stanton, R., editors, *Artificial Intelligence Developments and Applications*, pages 301–313, North-Holland, Amsterdam.

Tecuci, G. D. and Michalski, R. S. (1991). "A method for multistrategy task-adaptive learning based on plausible justifications." In Birnbaum, L. A. and Collins, G., editors, *Proceedings of the Eight International Workshop on Machine Learning (Evanston, IL)*, pages 549–553, San Mateo, CA., Morgan Kaufmann.

Tecuci, G. (1991). "A multistrategy learning approach to domain modeling and knowledge acquisition." In Kodratoff, Y., editor, *Machine Learning-EWSL91, Proceedings of the European Workshop on Machine Learning*, Berlin, Springer-Verlag.

Witten, I. H. and MacDonald, B. A. (1988). "Using concept learning for knowledge acquisition." *International Journal of Man-Machine Studies*, 29(2):171–196.