Carnegie Mellon University
Research Showcase @ CMU

Department of Civil and Environmental Engineering

Carnegie Institute of Technology

1989

Integration of generic learning tasks

Yoram Reich
Carnegie Mellon University

Steven J.(Steven Joseph) Fenves

Carnegie Mellon University.Engineering Design Research Center.

Follow this and additional works at: http://repository.cmu.edu/cee

Recommended Citation
Reich, Yoram; Fenves, Steven J.(Steven Joseph); and Carnegie Mellon University.Engineering Design Research Center., "Integration of generic learning tasks" (1989). *Department of Civil and Environmental Engineering.* Paper 37.
http://repository.cmu.edu/cee/37

This Technical Report is brought to you for free and open access by the Carnegie Institute of Technology at Research Showcase @ CMU. It has been accepted for inclusion in Department of Civil and Environmental Engineering by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Integration of Generic Learning Tasks

by

Yoram Reich, Steven J. Fenves

EDRC 12-28-89
Carnegie Mellon University

Integration of Generic Learning Tasks

Yoram Reich and Steven J. Fenves

Department of Civil Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper presents a novel approach for the creation of intelligent machine learning programs. It introduces generic learning tasks as the basic learning processes and describes their representation and integration. The role of knowledge in an intelligent machine learning system is identified to a fine granularity. The new approach allows the integration of multiple learning processes into a system that can select the appropriate process based on its background knowledge and the task. Three ways in which the system improves its learning bias over similar classes of problems are identified. The approach is demonstrated in the domain of bridge design.

This work has been supported by the Engineering Design Research Center, an NSF Engineering Research Center, and in part by the Sun Company grant for Engineering Design Research.

University Libraries
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

1. Introduction

Concept induction from examples can be viewed as a search for a definition that is consistent and complete with respect to the data present in the examples. Since there is potentially an infinite number of possible hypotheses that satisfy these requirements, restrictions arc introduced to constrain the space of possible concepts. These restrictions are termed Way.

Recently, researchers have begun to explore the issue of creating intelligent learning systems that derive their initial bias [20,19]; improve their learning performance by modifying their bias [15,19,21,25,27]; or make use of hybrid representation schemes using several biases [21,26]. Russell & Grosof describe a logic formalism for initial bias creation and updating for the version space approach [20,19]. Rendell *et al* describe a system, VBMS, that modifies its bias by solving problems in some domain using biases from a given collection [15]. The system is able to use its performance evaluation to cluster classes of problems with their appropriate bias. These clusters provide predictive power for matching the appropriate bias to an unsolved problem. Utgoff describes a perception tree representation scheme that utilizes a dual formalism for a specific task [26]. The system chooses to apply each of the formalisms in the appropriate case. The same idea is used in Schlimmer's system STAGGER [21], which uses three formalisms to represent a concept and integrates and uses them to modify its initial bias.

All of these studies attempt to provide a system with flexible bias selection and updating mechanisms. They deal with parts of the three origins of bias [4][1]:

1. *formal definition of the concept* - the formalism chosen for the task, for example, propositions logic.
2. *description language* - the language that accepts the formalism as its syntax and constrains the search for possible concept descriptions.
3. *characteristics of output* - these characteristics constrain the search for possible concept descriptions, for example, by limiting the number of disjunctions in a concept description.

These studies point to the need for further exploring the integration and interaction of different methods in learning algorithms.

This paper extends the treatment of intelligent learning systems by describing a framework for representing, using and improving learning mechanisms. The paper introduces the concept of *generic learning tasks* and describes how generic tasks are integrated into an architecture to form a general intelligent learning system that improves its learning performance over time. The key issues are the representation of generic learning tasks and their integration. The specific architecture we suggest, namely Soar, provides the adaptiveness and intelligence characteristics of the system. Our proposal can also be viewed as an implementation of RendelTs general framework for induction [14].

The motivation for constructing a system that supports the integration of various learning methods and is able to improve their use arises from our work on building a self-improving expert system for bridge design. The task analysis has determined that such a system requires the integration of several learning mechanisms - a capability missing in the current approach for constructing learning systems. In

---

Similar ideas are expressed by Rendell as affecting the credibility function [14].

the current approach, a learning mechanism and a representation formalism is matched *a priori* to a specific task that is described by a predetermined description language. The three aspects of representation formalism, description language, and learning mechanism determine a fixed bias for the system. Such a bias is necessary to provide a realistic performance; however, it relies on the initial understanding of the domain which might prove to be incorrect or incomplete. Furthermore, in the current approach, it is difficult to integrate several mechanisms into one system. A better approach is to start with some initial assumptions about the domain, provide a collection of learning mechanisms, and let the system use the most appropriate mechanism at different stages of the problem solving and learning processes.

This approach promotes two other desirable features. The first is that of making the assumptions of the learning mechanisms explicit This allows the system to reason about its own performance, thus improving its learning behavior. The second feature is that of describing various learning mechanisms using the same language to maintain an equal base line bias that result from the implementation language. We choose to use the *focussing* algorithm terminology throughout this paper [1,27]. This does not constrain the scope of the approach as other formalisms can be incorporated as well.

In Section 2 we introduce the concept of generic learning tasks. We describe in Section 3 how they are integrated. The role of knowledge in the system is made explicit in Section 4. Finally, we provide an example from the domain of bridge design and a conclusion.

The system is not currently implemented; however, it constitutes our major research goal.

2. Generic learning tasks

The idea of *generic tasks* is rooted in Chandrasekaran's woik on generic tasks as high level building blocks for expert systems design [3]. A generic task is a tuple *G=(I/OJCR,Q,* where:

- *I/O* is the Input/Output of the generic task;
- *KR* is the representation of knowledge required by the generic task. It describes how concepts should be structured to achieve the task functionality; and
- C is the control structure for the task. It specifies the procedure for achieving the task functionality by operating on the knowledge.

This tuple is crafted such that the knowledge representation and control used in the task are directed to provide the required Input/Output characteristics, which may be termed the functionality of the task. The representation and the control need to be designed such that their structure matches their semantics. The syntax-semantics correspondence establishes a strong limitation on the task scope but makes it productive for a specific problem. Such representation of generic tasks rests on solid software engineering guidelines. However, since we expect a system that uses these generic tasks to behave intelligently, the accessible knowledge about the task should contain the consequences of its knowledge representation and control with respect to its functionality.

Generic tasks can be combined to result in a more powerful system. In such a system each generic task solves a partial problem. The exploitation of this idea for knowledge acquisition systems is suggested in [2] and is exemplified in McDermott's work on knowledge acquisition systems [9]. Our intent is to extend the scope of generic tasks into learning systems in general, but more specifically into concept learning programs.

Some examples of generic learning tasks at the level of inductive operators are: adding or splitting a disjunct, adding or modifying a property, or creating a range for some continuous attribute. Examples of generic learning tasks at the level of complete induction systems are: candidate elimination [12] or focussing [1]. These tasks differ along several dimensions: scope of method (domain, range, and specific functionality), time and memory complexity, deficiencies, etc.

Figures 2-1 to 2-3 provide three examples of generic tasks in the domain of concept learning which have the same functionality: *enriching the description space*. However, this functionality is achieved by different methods having different characteristics. These characteristics will determine the task chosen for satisfying a goal to enrich the description space.

Two items of comparison are worth mentioning. First, the tree-hacking task specifies a single task whereas the two other describe a small family of tasks, each having different storage requirements levels for previous examples. Observing the characteristics of these tasks, it is rare that the learning process will produce an *optimal* concept description for a real life domain. Furthermore, there is no guarantee that the concept learned will be correct, rather than just being a reflection of the system's knowledge about the goal of the learning process and its collection of generic tasks. Second, these tasks are not restricted to deal with contradictions (e.g., a syntactic trigger) that arise in the learning process [1, 27]; they can compete at every cycle of generalization/discrimination when a new example is presented.

Tree hacking

I/O:

- *functionality*: enriches the description space (fine grained restructuring of one attribute)
- *method*: modifies the hierarchy of a tree
- *characteristics*:
  - provides additional possibilities for generalization/discrimination
  - preserves consistency with previous examples
  - *complexity*: manipulating *one* tree

KR:

- representation in the form of a description space
- trees with markers
- marks of positive and negative values on nodes that were visited

C:

- algorithm *tree-hacking*

Figure 2-1: Enriching the description space: Tree-hacking

3. Integrating generic learning tasks

Two important requirements for using generic tasks are the provision of abilities to communicate and share knowledge. These abilities can be supported by combining generic tasks in an architecture that provides efficient and goal-oriented flow of information. The goal-oriented architecture is necessary because goals determine the knowledge that is applicable and knowledge provides the ground for the learning process (see next section). We propose Soar as an exemplar of a suitable architecture for this integration [6]. Rosenbloom's *integrated learning hypothesis* suggests that various learning processes

Addingadisjunct

I/O:

- *functionality:* enriches the description space (dealing with concept described by several disjunctions)
- *method:* creates a copy of the description space and adds it to the concept description as a disjunct[2]
- *characteristics:*
  - preserves consistency with previous examples *(if all example* are maintained)
  - possibly overgeneralizes *(jfsome negative examples* are missing)[3]
  - possibly overgeneralizes and overspecializes different disjuncts(*if some positive* examples are missing)
  - *complexity:* reconsidering *all* previous examples with *all* the current disjuncts

KR:

- representation in the form of a description space
- all examples
- trees with current markers (if not all examples presented)
- classification of the current example (positive or negative)

C:

- algorithm *odd-disjunct*

Figure 2-2: Enriching the description space: Adding a disjunct

can be cast in Soar as processes of reconstructing a copy of the new knowledge based on the existing knowledge and storing it in long term memory [16]. The storage process is realized by the chunking mechanism [7]. This hypothesis can address the integration of different methods if they are fine grained processes. We hypothesize that the level of generic tasks provides this level of granularity.

Soar has several characteristics that make it suitable for the integration task. Any goal-oriented problem solving behavior is represented in Soar as a search in problem space. A goal is represented as a data structure that contains slots for the goal, the problem space for achieving the goal, the current state in the problem space and an operator that is suitable for the current state. The Soar architecture provides deliberate mechanisms to select and fill the goal context slots by selecting, possibly from several alternatives, an appropriate candidate. The selection is based on long-term knowledge about preferences for the candidates for some slot. The selection process is itself a goal-oriented process that might give rise to meta-level reasoning if immediate selection is not possible (e.g., in an impasse situation) [18]. Such meta-level reasoning is performed in a subgoal and can be summarized with some implicit generalizations by chunking the problem solving behavior into production rules. Another support for choosing Soar results from the experimental point of view: Soar has proved to be an appropriate tool for integrating multiple sources of knowledge in an automatic algorithm designer [22].

[2]One issue here is concerned with whether the task should copy the current description space after possibly some tree-hacking or property additions, or just copy the initial description space.

[3]Missing examples, negative or positive, introduce order dependency into the learning process. Previous examples are assumed to relate to the previous disjuncts. Missing negative examples cannot constrain the new disjunct resulting in an overgeneralization of the new part of the concept definition (partial solution is copying the negative markers to the new disjunct). Missing positive examples cannot be analyzed to check whether they best fa the new disjunct Thus, they might prevent the generalization of the new disjunct and maintain the overgeneralization in the previous disjuncts.

Adding a property

I/O:

- *functionality:* enriches the description space (reconsider previous decisions)[4]
- *method:* adds a new property tree as a new conjunct to the concept description space
- *characteristics:*
  - preserves consistency with previous examples[5]
  - might overgeneralize or overspecialize with respect to the new property (if some examples are missing)
  - *complexity:* reconsidering new property values of *all* previous examples with the new property tree

KR:

- representation in the form of a description space
- all examples (default, other possibilities affect the functionality)
- trees with current markers (if not all examples are presented)
- classification of the current example (positive or negative)

C:

- algorithm *odd-property*

Figure 2-3: Enriching the description space: Adding a property

In mapping our task onto Soar, the goal of the system is to generalize concepts from examples. Examples might be described symbolically or by using continuous attributes; attributes can be structured, ordered or not; the examples might be noisy and might contain redundant or insufficient information. Such variety cannot be handled by any one of the existing learning algorithms[6]. Furthermore, there might be several algorithms that are potentially suitable for some partial solution.

Representing the algorithms as generic tasks enables each of them to propose itself when its ability is appropriate and compete with other applicable generic tasks according to its ability and deficiencies. Then, Soar can reason about the knowledge embedded within the tasks; select the most appropriate generic task for some goal context; and augment its knowledge about the generic tasks by chunking subgoals. Therefore, Soar can start with a collection of generic tasks (i.e., biases) and gradually transform its bias toward a problem oriented bias selection.

In some learning tasks the generic tasks might not have knowledge about their utility and deficiencies. Since there is no metric for *bias* other then evaluating its performance in the context of some problem solving task [4], different generic tasks might be tried in a look-ahead experimentation. The knowledge learned in the look-ahead should be augmented to the generic tasks as a new piece of bias to be available for future consideration.

[4]The purpose of using this task is to reconsider previous decisions in light of the new property. Thus, the new property should get higher preference when trying to select between properties for the purpose of discrimination, tree hacking, etc.

[5]This is true if all examples are presented with their values of the new property or if we assume that this property is immaterial for all previous examples. This assumption means that these examples are already generalizations, but more important, it defeats the purpose of the task. Reconsidering the examples with the new description space might introduce new decision points in the process and therefore produce different final results.

[6]This does not conflict with the statement about the generality of the chunking mechanism [7] since our discussion is at the *knowledge level* whereas chunking is a *symbol level* mechanism that can implement knowledge level learning [17].

The bias is learned at three levels:

1. Learning at the first level occurs when a generic task that modifies the description space (e.g., tree-hacking) is selected and executed.
2. The second level, that of choosing the appropriate generic task, is described in the previous paragraph.
3. The third level occurs if the system decides to deliberately learn higher level concepts. The system can use sequences of generic task applications to form new high level tasks in the following way: the *I/O* is the goal the sequence has achieved; the *KR* is the KR of the first task in the sequence; and the *C* is the description of the sequence. These generic tasks are amenable to further generalizations by other generic tasks.

4. The role of knowledge

Stepp *et al.* describe some key ingredients that should characterize an improving learning system [24]. Among them are the ability to use and update background knowledge and the ability to discover and compose generalizations of concepts for the various aspects of the system (goals, examples, background knowledge, biases, etc.). However, knowledge has a much more explicit role in the performance of an intelligent learning system.

Intelligent systems should be able to reason about their tasks and derive the learning bias from their tasks and knowledge [14, 20]. These systems should be flexible enough to detect a wrong bias and modify it using their background knowledge. More formally, the approach we present "breaks down" the borders of the version search space (e.g., learning conjunctive concepts) by introducing a much larger search space of possible concept definitions (e.g., disjunctive and conjunctive concepts). Such a search process can be tractable only if knowledge is used to guide it.

Knowledge exists in two forms in intelligent systems. The first form is as background knowledge about the domain and the task at hand. This form plays a major role in the problem solving and learning process by providing the following processes:

- Translation of tasks to goals the system has to achieve [20];
- Selection of relevant properties from the description of objects, and constructing a description space that constrains the possible use of these properties. This is the most complicated process and the only inductive part in the learning process [1, 20]. A good description should reflect the prior expectations about the domain (semantics) to allow proper syntactic procedures to generate plausible results [5]. Most of the information used in the learning task comes from the description space rather then from the example descriptions [14];
- Making hidden properties operational [23, 13]. This allows the debugging of the initial phase of description space construction by augmenting additional properties into the concept description;
- Refining properties to the attributes necessary for the generalization process. This process is useful if there are multiple layers of knowledge as it enables the system to overcome overgeneralizations and contradictions at a higher level by resolving them at lower levels;
- Deriving new properties in a constructive induction process [8]. This process can enrich the language by creating new properties; and
- Assessing the relevance or utility of generated concepts and defining preferences for the form of concept description [5, 10, 11].

The second form of knowledge is embedded within the generic tasks describing their functionality,

required information, cost (memory and time complexity), performance measurements, deficiencies etc. This form has been elaborated in Section 2.

Knowledge for the leamihg process is brought to bear in matching the goals fonnulated by the background knowledge to the generic tasks functionality. The control knowledge is implicit in the knowledge about the generic learning mechanisms and in the specific task. The architecture makes this information explicit in the selection of the appropriate generic task for some goaL Knowledge about cost, deficiencies, and needed information of some generic task is used when several generic tasks compete for the same goal.

5. An example: a learning system for bridge design

The target of our research is to create a design system that improves its performance over time. Initially, the system has domain knowledge for the design of bridges based on structural engineering theory and is equipped with an analysis program or heuristic evaluators that provide feedback on its performance.

A subtask of the system process is to create high level concepts that assist in reducing the cost of the design. Examples for this task can be generated by the system while it solves problems or provided by a mentor. Figure 5-1 illustrates some of the properties describing bridges. These properties have discrete, possibly structured or ordered, and continuous values. The description language of the task is very rich. There might be unnecessary properties in example descriptions, and unoperational or hidden properties that need to be augmented to the description. Since the theory of structural engineering might be modified (for example, new materials, better understanding of failure modes, or new construction methods may be introduced), the generalized concepts may change over time as well.

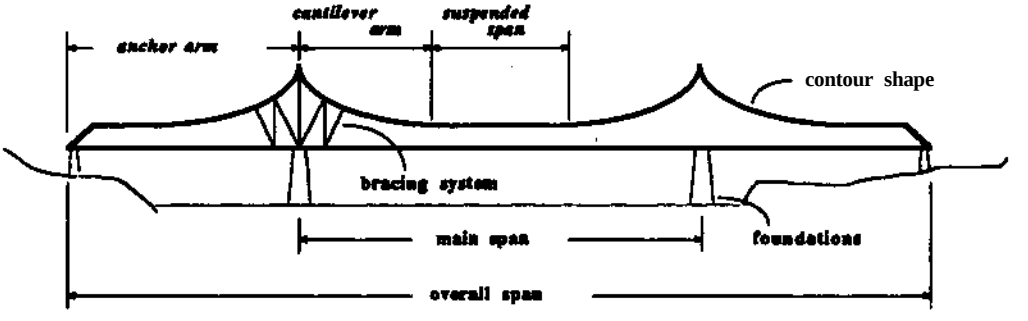

Figure 5-1: Bridge design domain

The system is given a goal to learn the concept of a good *cantilever arm* (see italized properties in Figure 5-1) for a *cantilever* type bridge from examples, mostly real, provided by an instructor[7]. The

[7]**The top goal of the system is to learn the concept of a cantilever bridge. In this process, the system should learn how to assemble cantilever and anchor arms with a suspended span in an appropriate way. Trying to learn first what is a good cantilever arm might introduce a premature bias that ignores the interaction between these concepts. However, for the sake of simplicity we use this simpler goal as an illustration.**

system assumes that most of the examples will be positive. Flat property trees[8] are inappropriate for such a set of examples, since after accepting two positive examples almost any negative example will lead to a contradiction. In this view, using the tree hacking algorithm after each example is fruitful since it can build a structured hierarchy. An initial hierarchy with some grouping of classes is created from knowledge about geometry and structural engineering (Figure 5-2). This is also the result of applying appropriate generic tasks. In this hierarchy, shapes 1 and 2 have different engineering interpretations: they differ in the type of support they have to offer to the suspended span. Note that the initial bias is toward grouping according to geometric knowledge.

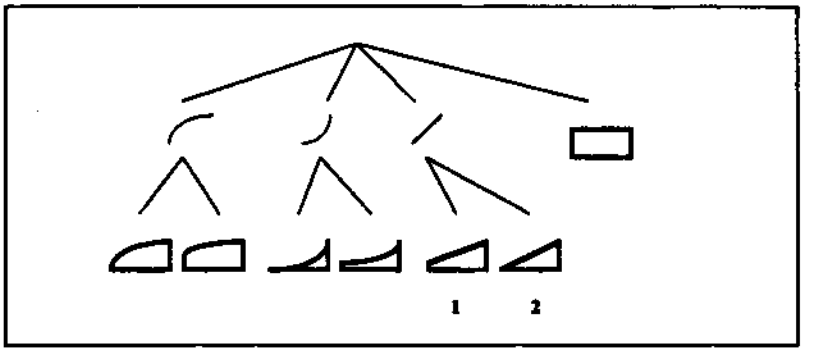

Figure 5-2: Initial shape property tree

Abstract descriptions of some bridges are provided (Figure 5-3) as initial examples. There are several possibilities to process the examples: tree-hacking to create a hierarchy, simple generalization, adding a disjunct, or adding a property. For the first generalization the last two processes do not help. Figure 5-4 shows the property tree at each stage of processing the examples. Two possible results after processing the first three examples are hierarchies 7 and 8. Hierarchy 7 is biased toward discriminating the positive examples from the rest of the domain whereas hierarchy 8 discriminates between negative examples and the rest of the domain. The results correlate with the application of the tree hacking algorithm. *Frequent application of tree hacking results in discriminating the positive examples from the rest of the domain.* This bias is not appropriate for our concept as the system assumes that most of the examples will be positive. This is a bias learned at the second level - that of generic task applications. If the tree hacking is expressed by a more fine grained description, the system will create two variations: TH_t and TH_2 that differ in the way they group the *negative* and the *undecided* subtrees. This difference can be viewed also as biased to discriminating either positive or negative examples from the rest of the domaia Note that some nodes (4 and 6) do not have other possibilities because the *U* and *L* markers are on top of the hierarchy and any addition of disjunct or property at this point will not advance the learning process.

The fourth example in Figure 5-3 marks a previously marked positive node as negative. Assuming that there are several properties describing the examples and that all negative markers of this example match positive markers on the property trees, this contradiction requires enriching the description language. There are several possibilities for achieving this goal depending on the following conditions:

[8]In a flat property tree, only the root-node has sons.

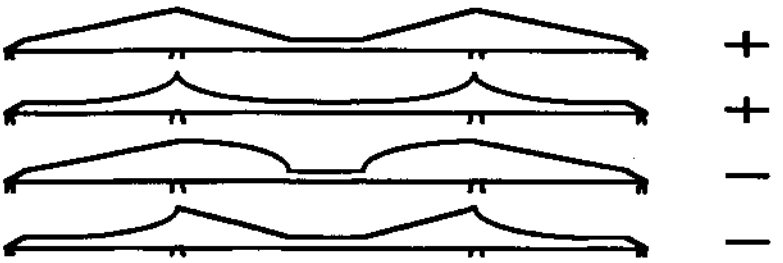

Figure 5-3: Examples of cantilever bridges

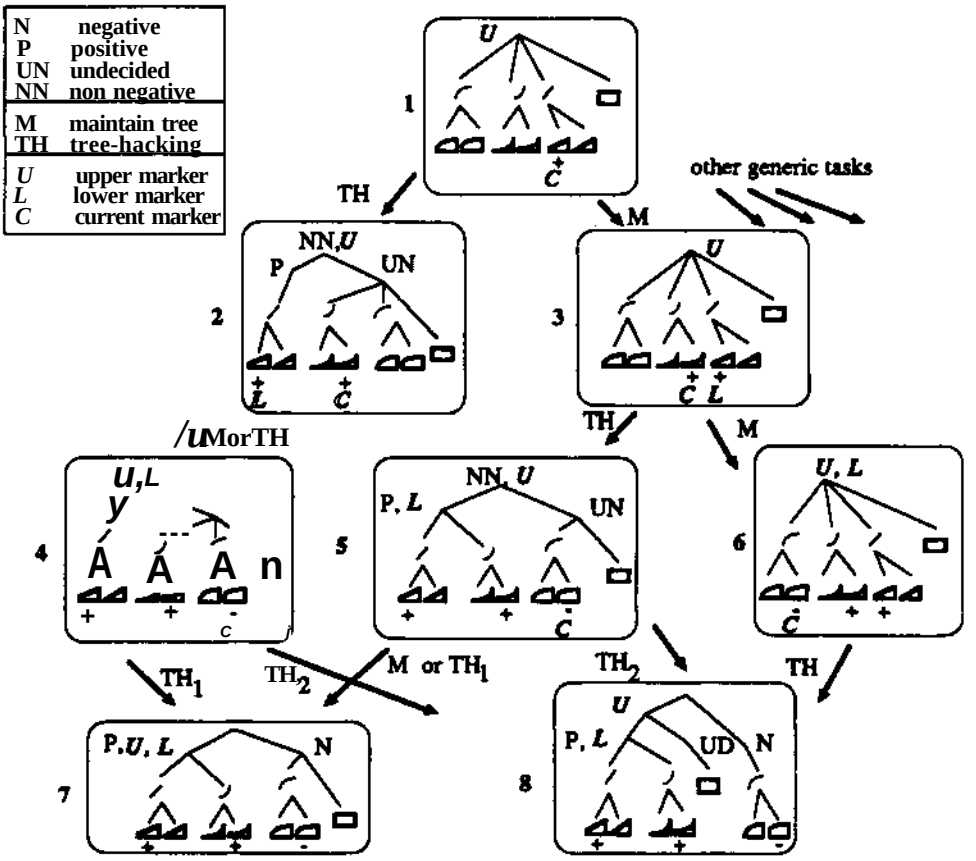

Figure 5-4: Learning the concept of a cantilever arm

- The example does not match a previously presented positive example. In this case the appropriate strategy is backtracking to previous decisions and considering other search paths for these decisions. Adding a new disjunct will not help because the current disjunct contradicts the negative example.
- The description of the new negative example matches the description of a previous positive example. This suggests that the description space is not sufficient and a new property should be introduced.

In our case, a plausible missing property is the span of the bridge. If span is very important, the shape

might be conditional upon the span. In this case the concept is best described by several disjunctions, each characterized by the appropriate span range and the corresponding shape.

Other tasks that are attempted by the system are learning the appropriate shape of an *anchor arm* and a *suspended span*. Based on structural engineering knowledge the system knows that an anchor arm serves similar purposes as the cantilever arm. Thus, it is cost effective to start learning the anchor arm concept with the final property hierarchy (7 or 8 in Figure 5-4). The same reasoning applies to the suspended arm except for the fact that its functionality is different than the cantilever arm. Transporting previously generated property hierarchy for a new concept is a bias learned at the first level.

An example of learning bias at the third level might be generating the original focussing generic task from applications of generalizations and discriminations on a perfectly structured description space.

This example and the type of bias that can be learned maps very well to other domains, as for example a *real* blocks world: learning what is a good block to put as a base for a tower, and later learning what can serve as a top block can be analyzed in the same way.

6. Summary and Conclusions

We have described a system for representing and integrating multiple learning mechanisms that can modify its learning performance. The system's learning performance changes by utilizing three mechanisms: changing the description space, learning appropriate sequences of generic task selections, and creating higher level generic tasks. The system is flexible and can accommodate additional learning mechanisms that are expressed as generic learning tasks. The system makes explicit the use of knowledge about the task and the generic mechanisms for choosing them for a given problem. This explicit knowledge is the base for transferring the learned bias to other tasks. We have described the use of knowledge in the various stages of the learning process. Specifically, we have demonstrated the system's scope in the context of learning concepts in bridge design. We have shown representative processes with the knowledge they use and the resulting bias at the three levels.

A contribution to the focussing algorithm has been made by better understanding of the semantics of the tree-hacking and other description space enhancement algorithms.

Much woric remains to be done. We intend to identify a complete collection of generic learning tasks for the bridge design domain and implement them in Soar. This implementation will lead to our top goal: the creation of an expert system for bridge design from the theory of structural engineering.

References

[1] Bundy, A., Silver, B., and Plumer, D.
An analytical comparison of some rule-learning programs.
Artificial Intelligence 27(2):137-181, 1985.

[2] Bylander, T. & Chandrasekaran, B.
Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition.
International Journal of Man-Machine Studies 26(2):231-243, 1987.

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

[4] Ganascia, J. G.
Improvement and refinement of the learning bias semantic.
In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 384-389. Pitman, Munich, W. Germany, 1988.

[5] Georgeff, M. P. and Wallace, C. S.
A general selection criterion for inductive inference.
In T. O'Shea (editor), *Advances in Artificial Intelligence*, pages 219-228. Elsevier Science Publishers, Amsterdam, 1985.

[6] Laird, J. E., Newell, A., and Rosenbloom, P. S.
Soar: an architecture for general intelligence.
Artificial Intelligence 33(1):1-64, 1987.

[7] Laird, J. E., Rosenbloom, P. S., and Newell, A.
Chunking in Soar: the anatomy of a general learning mechanism.
Machine Learning 1(1):11-44, 1986.
This paper presents the details of chunking in ***Soar***. It includes a demonstration of chunking based on Korf's Macro Problem Solver.

[8] Lenat, D. B.
AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search.
PhD thesis, Stanford University, 1976.

[9] McDermott, J.
Making expert systems explicit.
In H. J. Kugler (editor), *Information Processing 86*, pages 539-544. North-Holland, Amsterdam, 1986.

[10] Michalski, R. S.
A theory and methodology of inductive learning.
Artificial Intelligence 20(2):111-161, 1983.

[11] Minton,S.
Quantitative results concerning the utility of explanation-based learning.
In *Proceedings ofAAAI-88,* pages 564-569. Morgan Kaufmann, St. Paul, Minnesota, 1988.

[12] Mitchell, T.M.
Generalization as search.
Artificial Intelligence 18(2):206-226,1982.

[13] Purswani, K. and Rendell, L.
A reasoning-based approach to machine learning.
Computational Intelligence 3(4):351-366,1987.

[14] Rendell, L. A.
A general framework for induction and a study of selective induction.
Machine Learning 1(2): 177-226,1986.

[15] Rendell, L., Seshu, R., and Tcheng, D.
More robust concept learning using dynamically - variable bias.
In *Proceedings of the Fourth International Workshop on Machine Learning,* pages 66-78. Morgan Kaufinann, Irvine, CA, 1987.

[16] Rosenbloom, P. S.
Beyond generalization as search: towards a unified framewoik for the acquisition of new knowledge.
In G. F. DeJong (editor), *Proceedings of the AAAI Symposium on Explanation-Based Learning.* AAAI, Stanford, CA, 1988.

[17] Rosenbloom, P. S., Laird, J. E., and Newell, A.
Knowledge level learning in Soar.
In *Proceedings ofAAAI-87,* pages 499-504. Morgan Kaufinann, Seattle, WA, 1987.
Using chunking (symbol level learning mechanism) to learn in the knowledge level. Soar exhibits the ability to learn new symbols, recognize them, and recall them.

[18] Rosenbloom, P. S., Laird, J. E., and Newell, A.
Meta-levels in Soar.
In P. Maes and D. Nardi (editors), *Meta-Level Architectures and Reflection,* pages 227-240. Elsevier Science Publishers, Amsterdam, Holland, 1988.
Proceedings of a conference at Sardinia, 1986.

[19] Russell, S. J.
Tree-structured bias.
In *Proceedings ofAAAI-88,* pages 641-645. Morgan Kaufinann, St. Paul, Minnesota, 1988.

[20] Russell, S. J. and Grosof, B. N.
A declarative approach to bias in concept learning.
In *Proceedings ofAAAI-87,* pages 505-510. Morgan Kaufinann, Seattle, WA, 1987.

[21] Schlimmer, J. C.
Learning and representation change.
In *Proceddings ofAAAI-87*, pages 511-515. Morgan Kaufinann, Seattle, WA, 1987.

[22] Steier, D. and Newell, A.
Integrating multiple sources of knowledge into Designer-Soar, an automatic algorithm designer.
In *Proceedings ofAAAI-88*, pages 8-13. Morgan Kaufinann, St Paul, Minnesota, 1988.

[23] Stepp, R. E. and Michalski, R. S.
Conceptual clustering of structured objects: a goal-oriented approach.
Artificial Intelligence 28(l):43-69, 1986.

[24] Stepp, R. E., Whitehall, B. L. and Holder, L. B.
Towards intelligent machine learning algorithms.
In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 333-338. Pitman, Munich, W. Germany, 1988.

[25] Utgoff, P. E.
Machine Learning of Inductive Bias.
Kluwer Academic Publishers, Boston, MA, 1986.

[26] • Utgoff, P. E.
Perception trees: a case study in hybrid concept representation.
In *Proceedings ofAAAI-88*, pages 601-606. Morgan Kaufinann, St. Paul, Minnesota, 1988.

[27] Wielemaker, J. & Bundy, A.
Altering the description space for focussing.
In Merry, M. (editor), *Expert Systems 85*, pages 287-298. Cambridge University Press, Cambridge, December, 1985.