CONSTRAINED EXAMPLE GENERATION: A TESTBED FOR STUDYING ISSUES IN LEARNING

Edwina L. Rissland*
Elliot M. Soloway**

University of Massachusetts
Amherst, MA 01003

ABSTRACT

In this paper we explore issues in learning such as the role of agenda mechanisms, noticers, history keepers, initial seed sets of knowledge, and problem sequences in the context of our Constrained Exampled Generation (CEG) system working problems in the mini-domain of linear functions.

1. INTRODUCTION

In past work, we have investigated the structure of knowledge in complex domains like mathematics and computer science [3] CM][71 in particular the important role played by examples and the process of generating examples that meet specified properties or constraints [2][5][6]. We have built a computational model of the process of constrained example generation which was motivated in part by our observations of human subjects.

The basic Idea behind the CEG system is as follows: given the goal of producing an entity with certain properties, instead of generating the entity from first principles, find an example in the data base which "most closely" matches the desired constraints, and then apply difference*reducing operators in order to modify that example to fit the given constraints. The architecture of the CEG system is given in Fig. 1.

•This work supported, in part, by the National Science Foundation under grant IST-80-173^3.

••This work was also supported, in part, by the Army Research Institute for the Behavioral and Social Sciences, under ARI Grant No. MDA903-80-C-0508.

Any opinions, findings, conclusions or recommendations expressed in this report are those of the authors, and do not necessarily reflect the views of the U.S. Government.

We have explored the robustness of this model, and its LISP realization, in such domains as: generating specific atoms and lists in LISP, generating simple recursive programs in LISP, generating lines in algebra, generating scenes in a simple blocks world, and generating tactics in a game [5][11]. Effectively, the same system was used in each domain; only the domain specific knowledge was changed. The core system operates in a GPS-like fashion, applying a set of difference-detectors and a set of difference-reducers in order to modify examples to meet the desired constraints. Thus, in this work, the system could undertake remedial modifications since it "knew what to do".

Currently, we are investigating learning. To learn, a system must acquire experience and be able to review and summarize it C8 3C9 3. The design of the CEG system has the necessary modules to allow it to do so; in particular, the JUDGE, AGENDA-KEEPER, HISTORY-KEEPER, and NOTICER/GENERALIZER can be used to allow the system to monitor its own performance, and provide feedback, which as noted by Smith et al [10] is critical to learning.

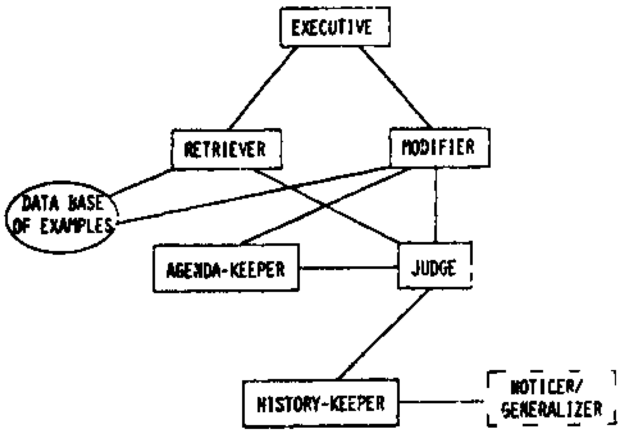

Fig. 1
Architecture of the CEG System.
(Dashes indicate that module is not yet fully perational.)

The specific task we have set for the learning-CEG system is to learn under which circumstances a particular reducer is appropriate. Neves [1] explored a similar problem. This task allows us to explore the space of factors involved in learning:

1. The role of agenda mechanisms
2. The role of history-taking and -summarization
3. The role of initial "seed" sets of examples
i*. The role of posed problems, their content and order.

The CEG system provides a vehicle in which to <u>empirically</u> explore the tradeoffs and interactions of these factors. The contribution of each of these components is a "parameter" to the current system. We are in the process of systematically testing various settings of these parameters.

For example, learning the appropriateness of an action implies learning something about the order in which to apply it. This is particularly important in the case of <u>interacting constraints</u> where remediation of one constraint deficiency might destroy satisfaction of another constraint.

The agenda mechanism also affects the exploration of sequences of actions and modified examples. For instance, if a particular modification routine has just made the candidate example "better", e.g., closer to meeting a constraint, should it be tried again, and if so, for how many times? On the other hand, should other routines be given an opportunity to demonstrate their effects? The latter technique would lead to discovering routines which are a mixed sequence of actions, whereas the former technique would lead to discovering routines composed of multiple copies of the same actions.

2. AN EXAMPLE FROM THE LINES DOMAIN

The domain chosen for our investigations is that of linear functions on the real numbers, i.e., lines like y = 2x «■ 1. A typical CEG problem might be to generate a line such that:
1. it is <u>steeper than</u> a given line, and
2. it has a negative <u>slope</u>.
For a general line, ay = bx + c, steepness is the absolute value of the slope, i.e., Ib/ai. The sign of the slope is the sign of (b/a).

The CEG system has an initial data base of example, i.e., an "Examples Space." An example is a specific line, ay = bx + c, represented as a frame, whose value slot contains the three-tuple (a b c). Other slots include "derived-from" pointers indicating from which other example the example is constructed, and scaring and history Information gathered during attempted modifications.

The "goodness" of a modification is measured by the JUDGE module and expressed in terms of two scores: (1) the "global esc" (constraint satisfaction count) which measures the example's satisfaction of all posed constraints; (2) the "local csc" which measures its satisfaction of the constraint currently being worked on. The scores are expressed discretely as "success", "better", "no-change", or "worse".

The system possesses five "primitive" routines that modify the x-coeffie lent:

1. make-steeperl (ms1) which adds 1 to the x-eoefficient;
2. make-steeper2 (ms2) which doubles the x-coefficient;
3. make-steeper3 (ms3) which squares the x-coefficient.
4. make-steeper[4] (ms4) which subtracts 1 from the x-coefficient;
5. make-negative (mn) which changes the sign of the x-coefficient.

Thus, there are four routines that modify the steepness and one that modifies the sign of the slope. The system starts out "knowing" that msl, ms2, ms3 and msl affect the steepness and thus, implicitly that steepness has something to do with the x-coeffiecient of a line. It does not know of the role of the y-coefficient.

Some specific "facts" we wish to have the system acquire are:

1. ms2 (doubling) always works;
2. ms1 (add!) works for slopes > 0;
3. ms3 (squaring) works where islope! > 1.
4. ms3 is "faster" than ms2 which is "faster" than ms1.
5. it is better to fix the steepness before the sign.

3. EXPERIMENTS AND RESULTS

Thus far, our experiments have involved variation of the following:

1. The agenda mechanism, in particular, the search of sequences of modifications;
2. the scoring of problems, in particular, whleh sequences count as "successes", "better", etc;
3. The initial seed set of examples possessed by the system;

4. The order of problems posed to the system;

5. The "remembering" and "forgetting" of examples created by the system, e.g., the addition of successful solution examples to the system's "Examples-space".

A sample of the observations made on our experiments are:

1. Selection of operators: Random selection (which allows the system to try the powerful ms3 routine more often) does in fact perform better than the regime that does not vary from the initial choice of operator. However, the sequences of operators obtained under random selection are effectively "impossible" to analyze, even by humans. Thus the style of the AGENDA-KEEPER influences the credit assignment task of the NOTICER/GENERALIZER.

2. Influence of Initial Knowledge Base. The initial seed set of examples should include a variety of examples, for instance, some likely-to-cause-trouble examples like y=2, some nicely-behaved examples like y=2x. The variety affects what is accessible to discovery; for instance, without lines with !slope! < 1, the system would *never* learn ms3's weakness, and without lines with negative slopes, ms4's strengths. There is a trade-off between the variety of the knowledge base and the complexity of the agenda mechanism: the system didn't need a highly tuned AGENDA-KEEPER if the KB contained variety. This is related to mesa and false peak problems in hill-climbing.

3. Tuning the data-base. The "goodness" of certain examples (leading to successes), like y=2x, and the "troublesome-ness" of others (leading to worse-es), like ys2, became readily apparent.

4. Remembering/forgetting. To force exercise and learning of abilities, don't save answers.

5. Persistence. Don't give up evaluating even if the situation looks locally worse. For Instance, one agenda mechanism stops the modification attempts at the first encountered "local worse*; it never led to evaluations that might have showed that ms1l and ms4 tried in sequence lead to "global no-change" which could lead to discovery of their relation as inverses.

4. CONCLUSIONS

In summary, we are using CEG to systematically explore alternative work-loads, influences, and contributions of modules in a learning system. We feel strongly that an empirical testbed is necessary for the understanding of AI systems which evolve in complex ways. There is a difference between making a system perform better and learning how it can learn to perform better. While expert system research often emphasizes the former, we feel it can benefit from study of the latter, which is our focus.

REFERENCES

[1] Neves, D. M., "A computer procedure that learns algebraic procedures by examining examples and by working test problems in a texbook". In Proc. Second Conference of Canadaln Society for Computational Studies of Intelligence. Toronto, May 1978.

[2] Rissland, E. L., "Example Generation". In Proc. Third National Conference of the Canadian Society for Computational Studies of Intelligence. Victoria, B.C., May 1980.

[3]__________. "The Structure of Knowledge in Complex Domains". In Proc. NIE-LRDC Conference on Thinking and Learning Skills. Pittsburgh, November 1980, in press.

[4]__________, "Understanding Understanding Mathematics". Cognitive Science, Vol. 2, No. 4, 1978.

[53 Rissland, E.L., and E. M. Soloway, "Generating Examples in LISP: Data and Programs". In Proc. International Workshop on Program Construction. Bonas, France, September 1980.

[6]__________, "Overview of an Example Generation System". In Proc. First National Conference on Artificial Intelligence. Stanford, August 1980.

[7]__________, "The Representation and Organization of a Knowledge Base About LISP Programming for an ICAI System". COINS Technical Report 80-08, Univ. of Mass, in preparation.

[8] Selfridge. O.G., "Learning to Count: How A Computer Might Do It". Bolt Beranek and Newman, Inc, 1979.

(93 Soloway, E. M., "Learning r Interpretation + Generalization: A Case Study in Knowledge-Directed Learning". COINS Technical Report 78-13, University of Massachussetts, 1978.

[103 Smith, R.G., T. M. Mitchell, R. A. Chestek and B. G. Buchanan, "A Model for Learning Systems". In Proc IJCAI-77.

[113 Wesley, L., "Learning Racketball by Constrained Example Generation". In Proc. IJCAI-81.