

AUTOMATIC DISCOVERY OF HEURISTICS FOR NONDETERMINISTIC PROGRAMS *

Salvatore J. Stolfo
Computer Science Department
Courant Institute
New York University
251 Mercer Street
New York, New York 10012

Malcolm C. Harrison
Computer Science Department
Courant Institute
New York University
251 Mercer Street
New York, New York 10012

This paper discusses one way to limit the cost of executing declaratively specified nondeterministic programs. The approach we take is to develop control heuristics for a family of problems from traces of sample solutions generated during a training session with a human expert. Algorithms have been developed which recognize a set of patterns in the sequence of 'knowledge applications' and which compile descriptions of these patterns in a control language, called CRAPS. The CRAPS descriptions generated are then used for guidance in solving subsequent problems. We describe an implementation of this approach and give the results of several experiments.

1. INTRODUCTION

During the last few years, a number of relatively effective Artificial Intelligence programs have been written incorporating considerable amounts of knowledge, and the problem of encoding such knowledge in a useful form has emerged as one of the central problems of AI. Recently, attention has turned towards mechanisms which facilitate incorporating limited procedural or heuristic information into a primarily declarative framework (for example, Rychener's approach [4] is to build a rational "goal" structure into declarative rule-based systems, while Davis [2] favors a separate set of "meta-rules", specifying control information). The problem of acquiring, debugging and extending control information has become increasingly important. We believe that there are a number of important areas in which it might be possible to deduce control information automatically from a declarative program. These include declaratively specified problems: (1) for which there exists a relatively simple algorithmic procedure; or (2) whose performance can be improved in frequently occurring or particularly important special cases; or (3) in which particular subproblems can be solved by simple

algorithmic procedures. This problem has been discussed by a number of previous workers in various forms (for example [1,3,7]); space does not permit a complete review here (see [5] for such a review and comparisons with the present work).

2. APPROACH

In our experiments, we have been working on a (slightly idealized) jigsaw puzzle, in which each piece has an average color, and four sides described by unique integers (with side i fitting side $-i$). A nondeterministic algorithm written in production language form for this problem can be found in [6].

The representation contains 33 productions, and is highly nondeterministic, and if executed would be hopelessly inefficient. It does not even "know" that it will find a solution (i.e. terminate) if it can repeat the production which puts a piece in the puzzle as frequently as possible, so even the crudest goal-subgoal structure is absent. (However, there is no production to remove a piece from the puzzle, which serves as a clue to an intelligent observer). It is even possible for the program to pick up a piece and then put it down immediately without doing anything with it.

*This material is based upon work supported by the Office of Naval Research under Contract No. N00014-75-C-0571, (NRO49-347).

However, sufficient productions are present to permit simple sequencing heuristics to reflect the usual strategies used by experi-

enced puzzle-solvers:

- ° work on the outside edges first (build a pile of outside edges); repeat the put-piece-in-puzzle sequence until this pile is empty;
- ° separate the pieces in the heap into piles of the same average color, and search the appropriate pile first;
- ° search first for pieces which have more than one neighbor of the same average color (e.g. work on the sky first);

It may appear that automatic detection of such heuristics is a task of great difficulty. Below we outline an approach to this problem which appears to be quite successful, describe a system which has been embedded in a production system language and give some experimental results.

Our procedure is as follows: we select a 'typical' input to the program and run the program repeatedly on this input, recording the sequence of rules selected. This is repeated for other typical inputs. We then attempt to describe the better (i.e. shorter) successful sequences in a language, CRAPS, designed for this purpose and described in the next section. We then use this description to guide the program's subsequent decisions. The analysis of the solution sequences also produces a set of meta-rules which are used to dynamically alter the sequencing in the event the description does not behave properly.

Inherent in this approach is the assumption that good decision-making procedures or heuristics can be inferred from the performance of the program on only selected inputs. In general, as suggested by work on inductive inference and information theory, we give preference to short CRAPS descriptions which will generate a high proportion of short successful solutions and few long or unsuccessful solutions. We anticipate that the selection of inputs will be critical and that eventually we will want to be able to handle new inputs incrementally. Initially, however, we have concentrated on the simpler problem of getting a good solution for the non-incremental case.

As suggested by sample experiments, the execution time of a completely declarative program will usually be too long to permit a solution except in the simpler cases. Accordingly, we run the program in "training mode" in which its decisions can be observed and if necessary corrected. In initial experiments this was done

by a trainer who was aware of the structure of the program; subsequently we anticipate the necessity of using techniques similar to those of Davis [2], which will enable the trainer to deal only with the external behavior of the program.

It should be noted that we have deliberately chosen to exclude information about sequences which end in failure. It is clear that, as found by Winston [8], counterexamples will be extremely valuable. However, as the reader will note below, even this simpler problem poses considerable technical difficulties, and it was our feeling that a clearer picture would emerge from the simpler approach. Furthermore, the experiments suggest that useful results can be obtained without counterexamples.

3. THE CRAPS LANGUAGE

The CRAPS language provides a semantic framework with which to specify or describe sequences of rule applications in the execution of the non-deterministic program. The operators of CRAPS are Concatenation, Repetition, Alternation and Permutation of Sequences, with enabling conditions which are DNF expressions of rule applicability. Concatenation corresponds to sequential execution of statements, and repetition corresponds to iteration statements with 'while' and 'until' termination conditions. The alternation operator specifies alternative sequences of actions much like an ALGOL-68 'case' statement or LISP 'cond' expression. The permutation primitive represents a form of concurrent execution similar to the specification of collateral expressions in ALGOL-68.

The algorithms which compile CRAPS descriptions are based on standard optimization techniques and divide and conquer pattern recognition algorithms. The input to the program consists of a sequence composed of the rules which were fired on each cycle together with the associated input/output information and the list of rules which were applicable on each cycle. The analysis also produces a set of meta-rules whose objective is to suggest productions to fire in the event the CRAPS description is inappropriate. Complete details of these algorithms can be found in [5,6].

4. EXPERIMENTS

The experiments were performed on 5 puzzles with 25-30 pieces. Over 400 productions were fired to solve a puzzle. The CRAPS description generated from a training session using one puzzle consisted of 75 concatenations and 15

repetitions. A total of 192 meta-rules were generated. The original puzzle used in the training session was easily solved using the CRAPS description; no meta-rules were needed. The second puzzle was solved in 10 1/2 minutes but required 5 meta-rule calls. The third puzzle failed to be solved because of an incorrect choice specified by the meta-rules after 193 cycles. A fourth puzzle was solved in 11 1/2 minutes with one meta-rule call. A fifth puzzle was run with the meta-rules alone and it failed to make correct decisions after 80 successful cycles were executed. Complete details of these experiments can be found in [5].

5. CONCLUSION

It is believed by many researchers that an important quality of intelligent behavior is the ability to improve performance with experience, and that generalizing a concept is a critical aspect of learning. In CRAPS, a form of generalization occurs when a repeating subsequence is collapsed into a repetition. Although quite restricted in scope, CRAPS shows how a system might learn procedures, a form of learning which we believe is very important. Although there are a number of very difficult technical problems, it seems to us that with more powerful pattern recognition techniques and more powerful generalizations of control statements, this approach could be very fruitful.

With less ambitious designs, CRAPS can be viewed as a programming aid for the designer and implementer of a large AI problem-solving knowledge base. The CRAPS approach might be useful in fine-tuning a declarative knowledge base as opposed to 'hand-compiling' control elements to effect competent performance in such a system.

The power of the descriptions produced is limited by both the expressive power of the CRAPS primitives, and the level of sophistication of the pattern recognition algorithms that have been developed. For example, during repetition detection no notion of similar subsequence is used; furthermore, it is not possible for alternation to appear within repetitions. Despite this, the algorithms are powerful enough to detect interesting patterns and subsequently interesting heuristics.

Our experiments suggest that this approach will have the best chance of success when the encoding of the knowledge of the problem domain is such that on any execution cycle a small number of productions and only one instantiation of each production is applicable. This suggests

one way in which CRAPS should evolve: the execution traces should include not only productions but also the instantiations of productions (i.e. contextual or even goal information should be included). However, recognizing patterns in such sequences is much more difficult. We view our approach as an initial step in the understanding of this more general problem.

REFERENCES

- [1] Biermann, A., "On the Inference of Turing Machines from Sample Computations, Artif. Intell. 3 (1972).
- [2] Davis, R., "Applications of Meta-Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases" Ph.D. Thesis, Computer Science Dept. Memo CS-76-552, Stanford University, July 1976.
- [3] Fikes, R., Hart, P., and Nilsson, N., "Learning and Executing Generalized Robot Plans," Artif. Intell. 3 (1972).
- [4] Rychener, M., "Control Requirements for the Design of Production System Architectures," Proc. Symp. on Artif. Intell. and Program. Lang., 1977.
- [5] Stolfo, S., "Automatic Discovery of Heuristics from Sample Execution Traces for Nondeterministic Programs," Ph.D. thesis, Courant Institute, New York Univ., 1979, (in preparation).
- [6] Stolfo, S., and Harrison, M., "Automatic Discovery of Heuristics for Nondeterministic Programs," (full version), Courant Institute, New York Univ., CS TR-7, 1979.
- [7] Waterman, D., "Generalization Learning Techniques for Automating the Learning of Heuristics," Artif. Intell. 1 (1970).
- [8] Winston, P., "Learning Structural Descriptions from Examples," MAC TR-76, Mass. Inst. of Tech., 1976.