FLEXIBLE LEARNING OF PROBLEM SOLVING HEURISTICS THROUGH ADAPTIVE SEARCH

Stephen F. Smith

The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213 USA

ABSTRACT

Noting that the methods employed by existing learning systems are often bound to the intended task domain and have little applicability outside that domain, this paper considers an alternative learning system design that offers greater flexibility without sacrificing performance. An operational prototype, constructed around a powerful adaptive search technique, is presented and applied to the problem of acquiring problem solving heuristics through experience. Some performance results obtained with the system in a poker betting domain are reported and compared with those of a previously investigated learning system in the same domain. It is seen that comparable levels of performance are achieved by the two systems, despite the latter's dependence on a considerable amount of domain specific knowledge for effective operation.

I INTRODUCTION

The design of problem solving systems capable of improving their performance autonomously through the accumulation of experience in a specific task domain has received much attention within Al [1, 2, 6, 11, 13, 14). Such learning systems have appeared in many problem solving contexts and several have been quite successful. Yet, the underlying methods by which problem solving heuristics are acquired in these systems are often bound to the intended domain, relying on considerable amounts of information specific to the domain for effective operation. As such, these methods are inflexible and have little applicability outs.de the intended domain. This paper considers an alternative learning system design that offers greater flexibility without sacrificing performance. The design is operationalized in a program called LS-1 that learns a set of heuristics, represented as a production system (PS) program, to govern the application of a set of operators in performing a particular problem solving task.

II LEARNING AS ADAPTIVE SEARCH

As in [10], learning is viewed as a searching problem in the system's representation space. The core of the system is an adaptive search technique called a genetic algorithm (GA). GAs are powerful procedures, motivated by standard models of heredity and evolution but applicable for searching any suitably represented domain of structures. Empirical studies have demonstrated their capabilities in the areas of function optimization [3], model-fitting [4], learning [8], and discovery [9]. Despite these results, GAs have been largely Ignored by the Al community."

This may be duo, in part, to a mistaken association with early evolutionary search models (e.g. [5]) that embodied a form of random search.

Briefly, a GA maintains a knowledge base of structures (e.g. alter native sets of problem solving heuristics lor a given domain) and proceeds by repeatedly 1) selecting structures on the basis of observed performance, and 2) applying idealized genetic operators to the structures selected to construct new structures. This results in a search wherein subsets of structure components found to contribute to good performance in the domain are propagated through the structures in the knowledge base, forming the basis for subsequent exploitation of larger and larger such subsets of components. Intuitively, selection according to performance focuses attention on the most promising subsets and application of the operators serves to explore the utility of these subsets in new contexts.

Structures are represented as sequences of their constituent components and manipulated as such by the genetic operators. The variants employed in LS I operate at various levels of granularity, manipulating sequences of productions at the highest level; sequences of symbols at the lowest level. The first operator, *crossover,* takes two structures, selects a breakpoint on each structure at a particular level of granularity, and exchanges the sequences of components to the right of the breakpoints. For example, if two structures consisting of the component sequences $c_1,c_{.2}|c_3.c_4.c_5$ and $c_1,c_{.2}|c_3.c_4.c_5$' are crossed at the designated breakpoints, the structures generated are $c_1,c_{.2}|c_3.c_4.c_5$ ' and $c_1,c_{.2}|c_3.c_4.c_5$. This operator tends to preserve subsets of components in relatively close physical proximity to one another in the input structures. A complementary operator called *inversion* alters the sequence of components representing a structure, thereby reducing the susceptibility of particular subsets of components to disruption by subsequent crossovers. In this case, two breakpoints are selected on a single structure, and the sequence of components delineated by the breakpoints is inverted. For example, an inversion of the structure $c_1|c_2,c_3,c_4|c_5$ at the designated breakpoints yields $c_1,c_{4l}c_3x_2.c_5$. Application of this operator is constrained to levels where interpretation of the components is independent of their positions in the sequence (e.g. at the production level). A final operator is *mutation,* which arbitrarily alters individual components of a structure. It functions as a background operator (i.e. its/probability of application is very low), its presence insuring the reachability of all points in the search space.

As alluded to above, the power of the algorithm lies not in the testing of individual structures but in the efficient exploitation of the wealth of information concerning the components comprising the structures. Consider the sequence of compononts $c_1,c_2....c_n$ that comprises a single structure S in the knowledge base. An evaluation of S (e.g. an assessment of the relative *worth* of S as a solution structure) actually provides information as to the *worth* of each of the different subsets of components that are present in S,

since it is these components that interact to produce the observed performance of S. If S has *n* components, then information is provided about 2^n-1 distinct subset?. The theory underlying the algorithms, established in [/J and extended to encompass LS-1's operator set in|12], states that the number of structures in the knowledge base possessing a given subset of components can be expected to increase or decrease over time at a rate directly proportional to the observed performance of the subset. Thus, all subsets of components appearing in the structures in the knowledge base are *simultaneously* pursued in a near-optimal fashion, a phenomenon referred to as *implicit parallelism.* The reader is referred to [3, 7| for a more detailed discussion of the algorithms and their properties.

The remainder of the paper is organized as follows. A brief overview of the LS-1 system is presented in Section III (a complete description may be found in [12]). In Section IV, some experimental results obtained with LS-1 in the poker betting task domain investigated in [14] are reported and a comparison is drawn. In Section V, some concluding remarks are made.

III SYSTEM OVERVIEW

The LS-1 learning paradigm is depicted schematically in Figure 1. The system maintains a knowledge base of *M* structures, each a candidate set of productions (or PS program) for solving the task at hand. On a given cycle through the learning loop, each PS program is applied by the *problem solving component* to *k* instances of the task. The *critic* analyzes the /< operator sequences generated by the problem solver in this proficiency test as well as characteristics of the PS program under evaluation, and assigns a performance measure indicative of the relative worth of the PS program as a potential solution to the task at hand. Once all structures in the knowledge base have been evaluated in this manner, the GA is invoked to construct a new knowledge base of structures for testing and the cycle is repeated. The knowledge base of PS programs, together with the associated performance measures, is viewed as LS-Vs internal memory, representing the sum of the system's experience in the task domain at any point in time. Externally, LS-1's current hypothesis as to a solution to the task is the PS program that has been highest rated by the critic thus far in the search. LS-1's progress is monitored by considering the sequence of hypotheses generated over time.

A. THE PROBLEM SOLVING COMPONENT

The PS architecture serving as LS-1's problem solving component is organized as a domain independent framework into which task specific primitives must be injected for operation within a given task domain. More specifically, LS-1 is instantiated in a particular problem solving domain by supplying an appropriate set of state variables and operators. The state variables provide the problem solver with a characterization of the domain and the operators form the problem solver's repertoire of alternatives in reacting to the current state. When enabled, the productions invoke operators and enter signals into working memory (WM) in response to perceived state variable and WM configurations.

The productions,which collectively constitute a single structure in LS-1's internal memory, are homogeneous and simplistic in nature. Each antecedent contains a fixed number of elementary patterns, one sensitive to each of the domain's state variables and a given number attending to the signals resident in WM. Each consequent contains a signal, to be placed in WM if the production is activated, and, optionally, the designation of an operator to be applied to the current state. If no operator is designated, the production is intended for internal communication

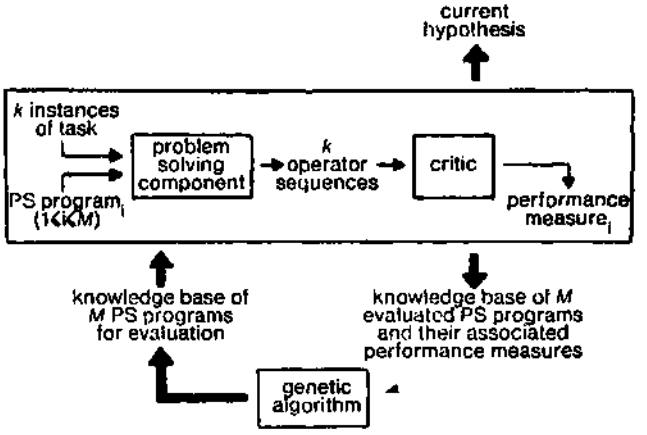

Figure 1: The LS-1 learning paradigm

purposes only.

This simple structure, necessary in providing a knowledge representation amenable to manipulation by a GA, imposes limits on the recognizing capabilities of individual productions. These deficiencies are balanced, however, by a PS control scheme that activates all instantiations found during the recognition phase of a given cycle. This allows distinct productions to cooperate in the recognition of more complex aspects of the current state. A default resolution mechanism coordinates external activity if more than one operator is designated by instantiated productions on the same cycle.

B. THE CRITIC

The LS-1 critic, in assessing the performance of the PS programs constructed by the GA, provides the GA with a global focus in the subsequent construction of PS programs. Accordingly, the critic is equipped with a means for analyzing the correctness of the *k* operator sequences produced by a given PS program during its proficiency test, and measures based on this analysis play a major role in the derivation of the overall performance rating. However, the critic also considers general characteristics of the PS program under evaluation that appear relevant to good performance regardless of the specific task domain. Structural properties of the productions (e.g. potential for communication in the WM patterns and signals, level of generality in the productions' conditions), dynamic properties of the execution history (e.g. percentage of productions activated, amount of dependence on the default resolution mechanism), and efficiency (e.g. number of productions) are all assessed to provide a finer level of discrimination between candidate structures.

IV SOME EXPERIMENTAL RESULTS

To provide a basis for comparison with previous work, LS-1 was tested in the poker betting domain first considered by Waterman [14]. Briefly, the system engages in a game of draw poker with the goal of making bet decisions that maximize profits. Each time it is the system's turn to bet, it must therefore infer the most appropriate bet decision to make, given the current state of the poker game. For purposes of evaluation in the experiments described below, the objective was to generate the correct bet decisions in each of 10 consecutive rounds of play.

Remaining faithful to Waterman's paradigm, the LS-1 problem solving component was presented with 7 state variables upon

which to base its bet decisions: *the value of the problem solver's hand, the amount of money in the pot. the amount of money last bet, the ratio of the amount of money in the pot to the amount of money last bet, the number of cards replaced by the opponent, a measure of the probability that the opponent can be bluffed,* and a *measure of the conservativcncss of the opponent.* 4 bet decisions were provided as operators: *call, drop, bet low,* and *bet high.* Within the critic, Waterman's deductive procedure for determining the correctness or incorrectness of a given bet decision, based on an *axiomatization* of the game of poker, was implemented and a measure indicating *the percentage of agreement with the axioms in the bet decisions generated* was incorporated in the evaluation function. A second measure relating to performance, *the number of successfully completed rounds of play,* was also utilized to insure that premature *drop* decisions (i.e. drops issued before cards have been replaced) would not be assessed as appropriate by the critic.

LS-1 was pitted against P[built-in], a poker betting program hand crafted by Waterman and judged to perform at roughly the same level as an experienced human poker player [14|. An internal knowledge base of 50 PS programs was maintained by the system, generated at the outset by randomly selecting conditions and operators for the individual productions of each structure. Running for 2 DEC10 cpu hours. LS-Vs operation verses, P[built in] was observed over an interval of 4,200 PS evaluations (approximately 85 cycles through the learning loop). As indicated by the performance results depicted in table 1. LS-1 achieved the performance objective within this time frame. A PS program was constructed that successfully completed 10 consecutive rounds of play in which there was total agreement with the poker axioms as to the bet decisions generated.

number of evaluations performed	500	1,000	2,000	3,000	4,200
performance rating of current hypothesis (scale: 0.0 · 1.0)	0.461	0.490	0.562	0.748	0.893
percentage of agreement with the poker axioms	71.4	80	76.1	85.7	100
number of successfully completed rounds	3	4	7	8	10

Table 1: Performance results in experiment 1

In attempting to understand these performance results it was discovered that the heuristics used by P[builtin], while appropriate within Waterman's paradigm, were not designed for such an extended game of poker (i.e. 42,000 rounds of play). Generally speaking. P[built-in] incorrectly judged LS 1 to be an extremely conservative player over time, and, with this misconception, was easily bluffed into dropping. LS-1 responded to its opponent in precisely the right manner, constructing PS programs that exhibited an overriding tendency to bet.

The decision to start LS-1 from *scratch* in these experiments was entirely motivated by a desire to provide a direct comparison with Waterman's results and is not due in any way to properties of the model. We could have initialized LS-1 at any level of expertise.

•••

Since a GA is inherently a stochastic process, successful performance in a single run might be misleading (e.g. a consequence of the random number seed). The results presented here actually represent averages taken over several distinct runs of LS-1.

To test LS-1 against a more formidable opponent, adjustments were made to P[built-in] to eliminate the above misconception and a second experiment was conducted. The results obtained in this case are presented in Table 2. We once again observe a steady and significant improvement in the hypotheses generated by LS-1 as the poker game progressed. Within 4,000 evaluations, a PS program was constructed that successfully completed 9 of 10 rounds of play while generating bet decisions that agreed with those deduced by the critic 82% of the time.

number of evaluations performed	500	1,000	2,000	3,000	4,000
performance rating of current hypothesis (scale: 0.0 · 1.0)	0.414	0.483	0.619	0.689	0.740
percentage of agreement with the poker axioms	58.75	73.03	89.9	75.1	81.93
number of successfully completed rounds	3	4	6	8	9

Table 2: Performance results in experiment 2

The relative success of LS 1 in the poker betting domain can be gauged by comparing these results to those obtained with Waterman's system. Waterman subjected his *trained* learning system to a proficiency test consisting of 50 rounds of poker against a human opponent, also measuring performance in terms of the percentage of agreement with the axioms in the bet decisions generated. 86% agreement was achieved by his system in this test [14]. Thus, while LS-1 PS programs were subjected to a shorter proficiency test (i.e. 10 rounds), the levels of performance achieved by the two systems are comparable. This becomes particularly significant when we examine the methods employed by Waterman's system in manipulating its PS program. Specifically, the creation of new productions was guided by an *a priori* supplied *decision matrix,* specifying for each possible bet decision, the reason why each state variable is relevant to that decision if, in fact, the variables are relevant. This information was used directly in building the predicates of new productions' antecedents.

V CONCLUDING REMARKS

The intent of the LS-1 design was to provide a flexible yet effective means of acquiring problem solving knowledge in a learning system. The experimental results presented above clearly illustrate the viability of the approach. Moreover, the method of learning employed by LS-1 embodies a general technique whose effectiveness in a given problem solving domain depends primarily on the quality of the evaluation function realized by the critic (LS-1 has, in fact, yielded similar performance results in another, unrelated domain [12], although space limitations prohibit their discussion here). Clearly, domain knowledge must enter into the derivation of the performance measure. However, It is felt that this shift in perspective with respect to the role of domain knowledge fundamentally enhances the general applicability of a learning system. The specification of domain specific measures to judge the external performance of hypotheses appears a far less imposing problem than the construction of learning methods based on domain specific assumptions.

ACKNOWLEDGMENTS

This work was conducted while the author was in the Computer Science Department at the University of Pittsburgh. The author would like to thank Ken DeJong for his guidance and encouragement throughout the course of the project.

REFERENCES

[1] Buchanon, B. and T.M Mitchell, "Model-Directed Learning of Production Rules", in Pattern Directed Inference Systems, Waterman and Hayes-Roth, Ed., Academic Press, 1978.

[2) Carbonell, J.G., "Experiential learning in Analogical Problem Solving", Proc. AAAI-82, August, 1902.

[31 DeJong,K.A., "Adaptive System Design: A Genetic Approach", IEEE Trans, on Man, Systems, and Cybernetics 10, 9 (September, 1980).

[4] DeJong, K.A. and T. Smith. "Genetic Algorithms Applied to Information Driven Models of US Migration Patterns", 12th Annual Pittsburgh Conf. on Modeling and Simulation, April, 1981.

[5] Fogel, L.J., A.J. Owens and M.J. Walsh, Artificial Intelligence Through Simulated Evolution, Wiley, New York, 1966.

[6] Hednck, C.L., "Learning Production Systems from Examples". Artificial Intelligence 7 (1976).

[7] Holland, J.H., Adaptation in Natural and Artificial Systems, Univ. of Michigan Press, 1975.

[8] Holland, J.H. and J. Reitman, "Cognitive Systems Based on Adaptive Algorithms", in Pattern Directed Inference Systems, Waterman and Hayes-Roth, Ed., Academic Press, 1978.

[9] Holland, J.H., "Adaptive Algorithms for Discovering and Using General Patterns in Growing Knowledge Bases", Int. Journal on Policy Analysis and Information Systems 4, 2 (1980).

[10] Mitchell, T.M., "Generalization as Search", Artificial Intelligence 18 (1982).

[11] Mitchell, T.M., P.E. Utgoff and R.B. Banerji, "Learning Problem Solving Heuristics by Experimentation", in Machine Learning, Michalski, Carbonell, and Mitchell, Ed., Tioga Press, 1983.

[12] Smith, S. F., A Learning System Based on Genetic Adaptive Algorithms, Ph.D. Th., Univ. of Pittsburgh, December, 1980.

[13] Stolfo, S.J. and M.C. Harrison, "Automatic Discovery of Heinictics for Nondeterministic Programs", Proc. IJCAI-79, 1979.

[14] Waterman, D.A., "Generalization Learning Techniques for Automating the Learning of Heuristics", Artificial Intelligence 1 (1970).