POPULATION-BASED LEARNING: A NEW METHOD FOR LEARNING FROM EXAMPLES UNDER RESOURCE CONSTRAINTS*

Benjamin W. Wah
Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801
(217) 333-3516
b-wah@uiuc.edu

ABSTRACT

In this paper, we study a new learning model for designing heuristics automatically under resource constraints. We focus on improving performance-related heuristic methods (HMs) in knowledge-lean application domains. We assume that learning is episodic, that the performance measures of an episode are dependent only on the final state reached in evaluating the corresponding test case and are independent of the intermediate decisions and internal states, and that the aggregate performance measures of the HMs involved are independent of the order of evaluation of test cases. The learning model is based on testing a population of competing HMs for an application problem and switches from one to another dynamically, depending on the outcome of previous tests. Its goal is to find a good HM within the resource constraints, with proper tradeoff between cost and quality. It complements existing point-based machine learning models that maintain only one incumbent HM, and that tests the HM extensively before switching to alternative ones. It extends existing work on classifier systems by addressing issues related to delays in feedback, scheduling of tests of HMs under limited resources, anomalies in performance evaluation, and scalability of HMs. Finally, we describe our experience in applying the learning method on five application problems.

KEYWORDS AND PHRASES

Heuristic method, knowledge-lean application, learning from examples, machine learning, performance evaluation, point-based learning, population-based learning, resource scheduling, test case.

* Research supported partially by National Aeronautics and Space Administration Grant NCC 2-481 and by National Science Foundation grant MIP 88-10584.

IEEE Trans. on Knowledge and Data Engineering, Oct. 1992, vol. 4, no. 5, pp. 454-474.

1. INTRODUCTION

Many application problems in decision, control, and optimization are sequential problems that involve making decisions one after another. The problem solver for such applications initiates multiple decisions at decision points in order to reach the final state. A solution in this context is taken as a sequence of decisions made for a given problem instance in order to reach a desired state. Often, such decisions are carried out by heuristics.

Heuristics, in general terms, are ‘‘rules of thumb’’ or ‘‘common-sense knowledge’’ [41] used in attempting the solution of a problem. Newell, *et al*., defined heuristics as ‘‘a process that may solve a given problem, but offers no guarantees of doing so.’’ [38] Pearl defined heuristics as ‘‘strategies using readily accessible though loosely applicable information to control problem-solving processes in human beings and machine’’ [41]. In this paper, we do not distinguish between heuristics and strategies; both can be considered as a sequence of decision rules applied at decision points that select alternatives [11]. Note that decision rules are heuristics themselves. To avoid confusion between heuristics and elements of heuristics, we use the term heuristic method (HM) in this paper to denote a collection of heuristic decision elements (HDE) or heuristic decision rules applied to solve a target problem.

HMs often work well but offer no guarantee on performance. In problems whose solution is evaluated by the cost of getting the solution and the quality of the solution obtained, HMs perform a tradeoff between cost and quality. One would expect the quality of the solution to be better if a HM takes more time. The level of tradeoff, however, is not precisely defined, independent of how well specified is the objective of the target problem that the HM solves. (The objective of the target problem may be well-defined, or ill-defined but measurable.) We define the performance of a HM as a collection, but an undefined numeric function, of performance measures, each of which is measurable at instants or over an interval. Note that performance measures do not necessarily correspond to cost and quality — elements of cost or quality may be measurable but not cost or quality itself. When performance measures are numeric values, the collection of values of one measure or a subset of measures may be condensed into a more concise form called an aggregate performance measure or aggregate performance function. Examples include the sample mean of values of the quality measure, and the sample mean of speedups, each of which is computed as the ratio of two values obtained from two completion-time measures.

In applications whose solutions are evaluated by either quality or cost, but not both, there is no tradeoff involved. For instance, a heuristic selection HM may be applied in a search algorithm for finding feasible or optimal solutions to a target problem. In this case, the quality of the method is 1 if an optimal or feasible solution is found, and 0 otherwise. The sole objective of the HM is to minimize the cost for finding the solution. However, this objective cannot be formulated mathematically; hence, the ‘‘optimal’’ method is unknown.

The problem of designing good HMs for solving an application problem can, therefore, be considered as a search for such HMs in the space of all possible HMs, without a well-defined objective function of performance measures. Our interest in this paper is on machine learning for the design of such HMs; this learning process can be viewed as an informed search of the space of possible HMs.

Figure 1 shows the major components of a problem solver and the way they are related. Heuristics are used in the Heuristics Selector, Heuristics Applicator, Performance Predictor, and Application Performance Database. The functions of each component are explained as follows, using the load balancing problem[1] (LB) defined in Table 1 as an example of the target application.

The *Heuristics Selector* selects the best heuristics to be applied in the Problem Solver, based on domain knowledge available, resources remaining, and past or predicted aggregate performance measures used to rank the heuristics. The heuristics used are static if they depend on static values determined a priori, and dynamic otherwise. For instance, the static heuristics in LB are the process-transfer protocols, whereas the dynamic heuristics are rules for determining when migration should be initiated and the placement and migration of processes.

The *Heuristics Applicator* is a component that executes the heuristics selected by the Heuristics Selector. In LB, the Heuristics Applicator is a set of utilities either for preempting a process from execution and for moving it to another location, or for placing a newly arrived process at a given site for execution.

The *Application Environment* represents the current state of all variables used by the Problem Solver. This state is affected by the execution of the heuristics by the Heuristics Applicator, with possible delay in the generation of feedback signals (as is shown by the delay element in Figure 1). In LB, the Application Environment is the UNIX operating system that provides either instantaneous performance measures, such as the utilization of the processor and the number of free memory pages at every real-time clock interrupt, or accumulated performance measures, such as the number of disk accesses and packets sent/received between two real-time clock interrupts.

The history of measured performance is captured in the *Application Performance Database* and may be processed in order to reduce its size. If a performance measure is not stationary with time, then a window of this measure may need to be processed in order to facilitate performance prediction in the Performance Predictor. For instance, in LB, filtering can be applied on a window of the CPU occupancies (which measure whether the CPU is occupied or is idle at every real-time clock interrupt) in order to allow the trend of processor utilization to be predicted by the Performance Predictor.

The last component in the Problem Solver is the *Performance Predictor*. This predicts the performance of the heuristics selected by the Heuristics Selector in the absence of their complete performance behavior. It models the performance of the Problem Solver, predicting the performance of the various heuristics if they were applied, and supplies these predictions to the Heuristics Selector. For instance, in LB, a Performance Predictor available in the UNIX operating system is the load average, which is a moving average of the sampled number of active processes over a window of time (say 2 seconds).

1. The objective of load balancing is to find a site with the maximum *relative* speedup with respect to a reference computer for executing any given process. This does not require knowing the actual speedup, as the latter is unknown before the process completes.

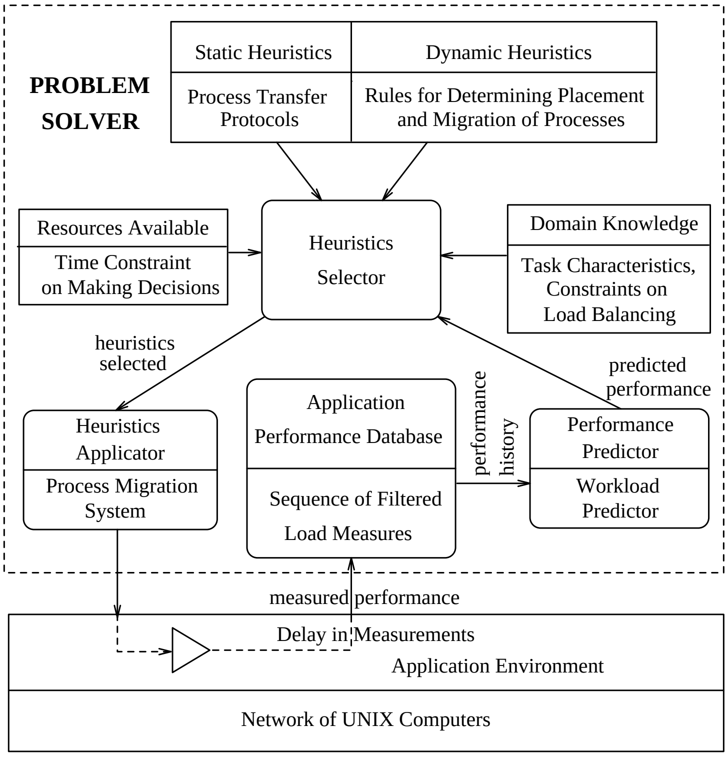

Figure 1. HMs used in the Problem Solver as is illustrated by the load balancing problem defined in Table 1. (The top caption in each box represents the generic function of the component; the bottom caption is the corresponding function in the load balancing problem.)

Table 1. The load balancing problem (LB).

Definition:

Given a process with unknown execution time and a network of UNIX computers with stochastic but non-stationary background workload, determine where the process is to be executed so that the expected speedup with respect to a reference computer is the maximum.

Characteristics:

- ☐ The set of measurable parameters is not well defined. Further, there is no model relating the available measurable parameters, such as processor utilization, disk traffic, network traffic, and memory occupancy.
- ☐ The load balancing HM has an undefined relationship between cost (in terms of the overhead for collecting measurable parameters and for determining the location of execution) and quality (in terms of the speedup of the process mapped).

We present in this paper a new machine learning model for learning HMs, based on a population of competing HMs. The major emphasis of population-based learning is on the systematic scheduling of tests of a possibly infinite set of alternative HMs under *resource constraints*. The ideal would be to learn all the heuristics used in the Problem Solver in Figure 1 in one learning session; however, the complexity of such may be too high under the resource constraints given, and the learning system may have to focus on one or a small subset at a time. For instance, the learning system may focus to learn and improve the Heuristics Selector. In this case, the heuristics used there are to be learned, and the rules selected by it, fixed during learning.

This paper is organized as follows. We present in the next section terminologies and assumptions used, and define the scope where population-based learning is suitable. After surveying previous work on point-based learning in Section 3, we present an integrated population-based and point-based learning model in Section 4 and identify improvements over the classifier-system approach in genetic learning. We discuss in Section 5 issues related to the integrated learning model, which include the evaluation and normalization of performance measures of HMs, the development of meta-heuristics to be used in the learning system, and the statistical scheduling of tests of HMs using limited resources. Section 6 describes the application of population-based learning and experiments carried out for enhancing heuristics used in four applications: a) load balancing of independent processes, b) automated design of artificial neural networks, c) depth perception in stereo vision, and d) combinatorial search using a branch-and-bound algorithm. Throughout the paper, we illustrate our learning method on an application for mapping a set of communicating processes on a network of distributed-memory computers.

2. CHARACTERISTICS OF HEURISTIC METHODS LEARNED

In this section, we characterize HMs to be learned by population-based learning with respect to domain knowledge available, overhead for evaluating a HM on a test case, and the nature of the HMs. Related issues considered include credit assignment and online versus offline learning.

In general, an application problem is solved by HMs of different types: some are derived from extensive background knowledge, while others are developed with little or no background knowledge. The learning and refinement of the former has been addressed extensively in traditional machine learning studies [32], which focus on representing domain-dependent and domain-independent knowledge, and on reasoning and deductions on the knowledge stored. An example is illustrated by a learning method proposed by Pearl [40] which discovers heuristics by first consulting simplified models of the problem domain, and by systematic refinement and deletion of constraints from the original application specifications until a semi-decomposable model is found. The solution to the relaxed problem then constitutes a HM for the application problem.

2.1. Credit Assignment

In this paper, we are interested in the automated design and refinement of HMs for applications with little or no domain knowledge on the HMs concerned. Such knowledge–lean applications lack a detailed world model that captures the relationship among states, decisions, and feedback signals generated by the learning system or measured in the environment. As a result of this lack of domain knowledge, the inference of the world model during learning is either too complex or not fully understood.

The design of a learning system is further complicated when there may be delays in getting the feedback signals due to a decision. In this case, multiple subsequent decisions may have been made between the time a decision was made and its feedback signal received. For knowledge-lean applications, the delay in feedback and the lack of a world model mean that there is little or no domain knowledge for credit assignment, which involves apportioning a feedback signal to individual decisions carried out in the past, as well as to decision elements applied in each decision, in order to refine the HM. The former credit assignment is called temporal, and the latter, structural.

Structural credit assignment [53] entails ways of using feedback signals to refine the individual components or rules of a HM. A lack of domain knowledge in this area means that it is difficult to model the behavior of the HM with respect to the feedback signals obtained and to modify the HM in order to arrive at better ones. An example of such a case is when the HM is implemented as a rule-based system, where there is little domain knowledge for apportioning the feedback signal to individual rules and parameters of rules. To overcome this difficulty, the learning system can maintain not only one incumbent HM but a population of competing ones; the choice of the best one to evaluate or new ones to create is delayed until more tests are performed on the HMs generated. In this way, the testing of different HMs may be interleaved in time. Issues and approaches for such a population-based learning method is explored in detail in this paper.

Temporal credit assignment [53] entails the apportioning of temporal global feedback signals by the learning system to the past decisions that affect these signals. When a decision is applied, its *temporal scope* is the interval of time during which its direct effect can be observed in the application environment. If the temporal scope is infinite and state changes are Markovian, then the effects due to a feedback signal will be attributed only to the most recent decision made in the past, and the effects of other decisions will be felt indirectly through intervening decisions and states. When the temporal scope is finite and state changes are dependent and non-Markovian, then an approximate temporal model is needed for temporal credit assignment.

In order for population-based learning to be feasible, the temporal scope of decisions must be in such a way that it is possible to test HMs incrementally and to evaluate their performance when needed. We identify the following necessary characteristics on the temporal scope of decisions that must be possessed by HMs suitable to be learned by population-based learning. a) Experiments in the target problem can be partitioned into episodes or test cases so that feedback signals due to a HM applied on a test case occur only when the evaluation of the test case is completed. Decisions in the HMs learned will, therefore, have a finite temporal scope. Note that state changes in the temporal scope may be dependent and non-Markovian. b) The performance measures of an episode, such as quality and cost, are dependent only on the state and performance values measured at the end of the episode, and are independent of the sequence of intermediate decisions made and internal states generated. c) The aggregate performance measures of a HM are independent of the order in which test cases are applied. This condition implies that decisions made in one test case do not affect decisions made in other test cases. As a result, computing the aggregate performance measures do not require knowing the effects of decisions for one test case and feedback signals for others. This condition is necessary in population-based learning because HMs may be tested on test cases in any order when multiple HMs are candidates for testing. An example of an aggregate performance measure that satisfies condition (c) is the average of the performance values of applying a HM on a set of independent test cases.

2.2. Online versus Offline Learning

Besides the conditions stated above for credit assignment, learning is feasible only when it is possible to carry out experiments on different HMs with reasonable overhead. Depending on the overhead associated with learning experiments, learning methods can be classified as online and offline.

Online learning involves evaluating HMs on test cases during learning. It should be used when the HM to be learned is scalable between small and realistic test cases. In this case, the HMs can be tested on small test cases during learning with a relatively small overhead. After learning is completed, scalability is verified on the HM learned using realistic test cases.

Offline learning involves evaluating the effects of all possible decision sequences of HMs on test cases ahead of time, and looking up the corresponding responses to a decision during learning. It should be used when a) the HM to be learned is not scalable between small and realistic test cases, and the evaluation of realistic test cases is time consuming; and b) the performance results of evaluating each realistic test case, using all possible decision sequences and starting from a given initial condition, are manageable and can be generated and stored ahead of time. The second condition means that it is possible to generate *offline*, for various initial conditions of a test case, the performance results of applying all possible decision sequences on this test case. During learning, when a decision sequence is applied, its effects can be found by looking up the performance results stored, without actually carrying out the decision sequence. If this condition is not true, then the effects due to a decision sequence will have to be found by actually carrying out the decision sequence during learning. This results in a higher overhead in each learning experiment and, within a fixed amount of time, less number of experiments performed on the HMs.

Offline learning has been applied in various forms in traditional machine learning. In all these cases, certain behavior of the system is established offline before the heuristics are learned. For instance, in learning the weights of feed-forward artificial neural networks, a set of expected output patterns, each corresponding to a given input pattern, are generated offline by instrumenting the target application. As another example, in Problem Solvers with a Performance Predictor (see Figure 1), the learning of heuristics is simpler if the Performance Predictor can be learned offline. This is the case in the load balancing problem defined in Table 1, where the learning of process placement and migration HMs will be easier if the Workload Predictor is learned offline.

Table 2. The process mapping problem (PM) and the learning of its HMs.

Definition: Assuming a set of communicating processes with known processing and communication requirements, and a network of idle distributed-memory computers, the problem is to map the processes to the computers so that the completion time of executing the processes is the minimum. The target problem is NP-hard and has well-defined objective and constraints.
Characteristics: ☐ Measurable parameters are well defined and include the execution times of processes in between communications with other processes, the amount of data in a message, the network speed, and the average communication overhead. ☐ Relationship between cost (execution time) and quality (completion time of the processes mapped) of the heuristic mapping methods is undefined.
Credit assignment: ☐ *Structural*: Difficult, as the HM is rule based [64] and the domain is knowledge lean. ☐ *Temporal*: Simple, as the conditions prescribed for temporal credit assignment in population-based learning are satisfied.
Online learning: ☐ HMs are tested on small and independent test cases during learning; scalability is verified after learning is completed.

2.3. Example

To illustrate the concepts we have defined so far, consider a target application of mapping a set of communicating processes on a network of distributed-memory computers (see Table 2). We assume that the HM is implemented as a set of rules (or HDEs) in the post-game analysis system [64], which is used to find efficient mappings of processes for networks of computers. The performance measures of the HMs involve cost and quality, and their tradeoff is undefined; hence, the objective of designing good HMs is undefined as well. Table 2 also lists issues in credit assignment. Structural credit assignment is difficult for such heuristic mapping methods implemented as a rule-based system, as there is inadequate domain knowledge for relating performance feedback signals to improvements for HMs. Temporal credit assignment, on the other hand, satisfies the conditions prescribed for population-based learning. a) Learning can be carried out using small and independent test cases with finite temporal scopes. b) The performance measures of a HM applied on a test case (a process graph) depend only on the completion time of mapping the process graph on a given network of computers and the overhead for generating such a mapping. These measures are independent of the individual decisions and internal states in the decision sequence. c) The aggregate performance measures (average completion time and average overhead for mapping a set of process graphs on the network of computers) are independent of the order of evaluating the test cases. Further, HMs learned using small test cases are scalable (to some extent) to larger ones. Given these conditions, online population-based learning can be used.

2.4. Other Issues

In addition to the issues on feasibility of learning, HMs can be differentiated according to how difficult it is to learn them by population-based methods. We characterize this difficulty with respect to the nature of HMs to be learned and their possible margin for improvement.

First, population-based learning is suitable for learning performance-related heuristics but not for learning correctness-related ones. A HM is said to be performance related if the constraints of the target problem are trivially satisfied, and the goal of the HM is to find efficient solutions. In contrast, a HM is correctness related if the constraints of the problem are hard to satisfy, and the goal is to find a feasible as well as efficient solution that satisfies the constraints. In knowledge-lean applications, HMs that are performance related are easier to learn than ones that are correctness related.

In learning correctness-related HMs, domain knowledge is neeed for guiding the generation and verification of heuristics. Examples of such domain knowledge include the set of correct operators to apply in the Problem Solver, verification that the objective or optimality condition of the target problem is achieved, syntactic and semantic knowledge used to perturb existing HMs or to compose new ones, and in some cases, a way to prove the correctness of the HM learned. Given this domain knowledge, correctness-related HMs can be learned by examples: the learning system learns these methods by trying to prove them through positive examples and disprove them through negative examples. This learning approach was taken in AQ [31], LEX2 [35], SAGE.2 [25], and TEACHER 1.0 [65]. The key in these learning systems is on techniques for transforming a HM into another when positive or negative examples are found. Note that the learning of correctness-related HMs is not improved even when a population of competing HMs are maintained, as the performance measures available (in terms of the number of positive examples found for a HM) cannot be used to identify a promising HM to test next.

Second, population-based learning methods (and other learning methods as well) can be used for improving performance-related HMs if there is a suitable margin for improving the aggregate performance measures targeted for enhancement. This margin is related to the quality and cost for finding solutions by existing HMs. In general, it will be difficult to improve an existing HM if the solution it finds has a quality very close to the maximum and a cost very close to the minimum. In this case, it will be easier to learn a HM with one or more of the performance requirements relaxed. For instance, if the existing HM has an average quality very close to the maximum, then it will be difficult to learn a new HM with a better average quality, but it may be easier to learn one with a lower average quality and lower average cost.

Table 3. Characteristics of HMs suitable for population-based learning.

Application Environment:
□ Knowledge lean, without a good world model.
Type of HMs:
□ Performance related.
Structural credit assignment:
□ Domain knowledge for structural credit assignment is missing.
Temporal credit assignment:
□ Learning is episodic with finite temporal scopes. If state changes within the temporal scopes are non-Markovian, then an approximate temporal model will be needed for temporal credit assignment.
□ Performance measures for evaluating a test case (episode) are dependent only on the final state reached in the episode and are independent of the intermediate decisions and internal states.
□ Aggregate performance measures of a HM are independent of the order of evaluating the test cases.
Online learning:
□ Aggregate performance measures of HMs learned are scalable between small and realistic test cases. The overhead is low for evaluating a HM on a small test case.
Offline learning:
□ Aggregate performance measures of HMs learned are *not* scalable between small and realistic test cases; the overhead is large for learning with realistic test cases.
□ The performance results of evaluating a realistic test case, using all possible decision sequences and starting from a given initial condition, are manageable and can be generated ahead of time.

2.5. Summary

Table 3 summarizes the scope of HMs that can be learned by a population-based learning model. Problems that are not covered either are too difficult to be learned automatically or can be learned by existing machine learning methods. We are interested in finding better HMs for solving a set of instances of the target application, but not in finding a good HM for solving one particular problem instance, as this is less common in practice. Note that population-based learning addresses ways for coping with a lack of domain knowledge for structural credit assignment. It does not address ways for temporal credit assignment, which must be solved either implicitly due to the nature of the problem, or explicitly by assuming an approximate temporal model.

Examples of HMs that can be learned include thresholds of functions, numerical coefficients that are parts of a rule or function, numerical functions obtained by composition of other numerical functions, symbolic rules, and macro-operators formed by composition of basic operators. The simplest case, therefore, includes parameter tuning,

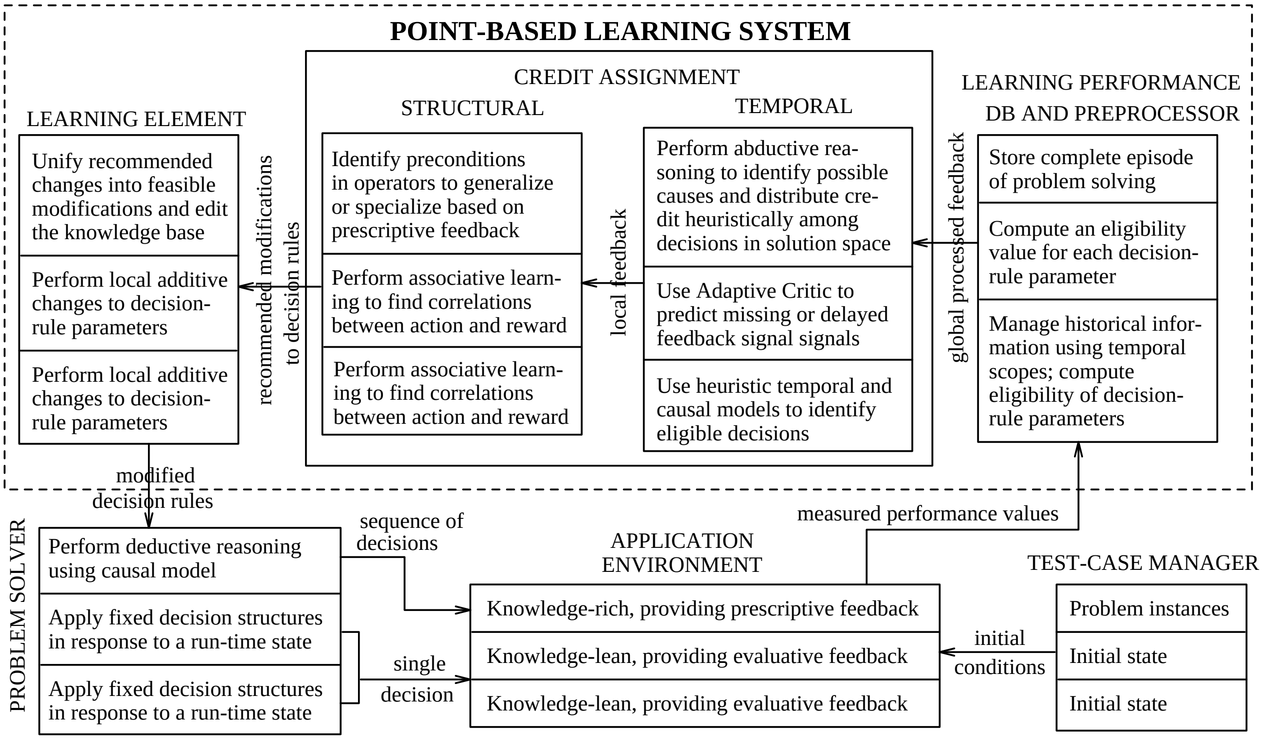

Figure 2. Point-based learning models. (In each block, the box on the top represents actions carried out in Dietterich and Buchanan's model; the box in the middle, Minsky's model; and the box in the bottom, the hybrid model.)

---

while more complicated cases involve generating symbolic rules and relations as well as tuning parameters of rules in a rule-based system. Note that the new HMs generated must be semantically sensible, which must either be enforced by meta-heuristics used in the generation process, or be verified in a post-learning phase.

3. POINT-BASED LEARNING MODELS

In this section, we survey three models developed in traditional machine learning studies for learning HMs, and discuss why they cannot be used to learn HMs with characteristics summarized in Table 3. The existing models examine one incumbent HM at a time; hence, they are termed point-based learning models in this paper. Fundamental work in this area was addressed by Minsky [33], Mitchell [34, 48], and Dietterich and Buchanan [7]. The basic principle in various learning models is based on a generate-and-test paradigm that generates plausible HMs, performs limited tests, and modifies the HMs according to feedback signals obtained. Note that in order for a generate-and-test method to work, it is essential that experiments with different test cases be repeatable, allowing different HMs to be tested under identical experimental conditions. The three models are described briefly as follows and are summarized in Figure 2.

3.1. Dietterich and Buchanan's Model

Dietterich and Buchanan's Model [1, 7], and similar models proposed by Smith, *et al*. [50] and Langley [25], belong to a class of models for learning HMs of target problems with well-defined objectives. They learn by supervised learning in a knowledge-rich environment with prescriptive feedback, which carries explicit information about the desired operators and/or states; the learning system can use the feedback signals for computing an error in order to guide the modification of the HM tested. The HMs learned are generally in the form of static deterministic decision rules represented as preconditions of either operators or operator sequences.

The learning model, shown in Figure 2, includes the Learning Performance Database and Preprocessor, Credit Assignment unit, and Learning Element. The Problem Solver, also called the Performance Element, and its initial conditions are shown as components outside the learning system. Actions carried out in Dietterich and Buchanan's model are indicated by boxes on the top in each block in Figure 2. The *Problem Solver* finds solutions to problem instances by performing an explicit search for the goal state using deterministic decision rules. By reasoning using the *causal model*, which relates preconditions of decision rules to post-conditions, the Problem Solver generates a sequence of decisions. These decisions affect the environment, whose effect is captured by the *Learning Performance Database and Preprocessor* as a complete solution sequence (or state changes). The preprocessed data are then used for credit assignment, which relates the external feedback to objectives and to determine the preconditions of operators responsible for the current feedback. Temporal credit assignment, based on abductive reasoning, identifies possible causes to certain feedback signals from the causal model, and distributes credit heuristically among candidate decisions in the solution sequence. Structural credit assignment, on the other hand, is performed by identifying preconditions in decision rules to generalize or specialize. The result of credit assignment is a set of recommended modifications to the decision rules, which are passed to the Learning Element. The *Learning Element* performs syntactic operations for parsing and for modifying HMs in symbolic representations. Using an internal model of the Problem Solver and knowledge for reasoning, the Learning Element generalizes stored solutions by deriving preconditions and macro-operator sequences, based on solving successfully a few episodes of the target problem. Knowledge in the Learning Element can be specified procedurally either in the form of a learning algorithm, such as that used in LEX [36], or declaratively in the form of editing rules, such as those in SAGE.2 [25], ODYSSEUS [61], and Cupr [23].

3.2. Minsky's Model

Minsky's Model [33] is an older and perhaps less restricted model of learning systems. It applies well for learning HMs of target problems with undefined objectives in knowledge-lean environments, which tend to produce evaluative and possibly delayed feedback signals.

Evaluative feedback carries only implicit information about the desired behavior but explicit evaluation of the observed behavior. Such behavior are intrinsically *a posteriori*, being measured or generated after the behavior has occurred. It requires a critic [60] who has some prior knowledge of the objective function and can assess the goodness of external states or sequences thereof. Scalar evaluative feedback signals are called *reinforcements* [33], and learning from such signals, reinforcement learning.

Evaluative feedback is often generated by dynamic and stochastic HMs. These HMs operate continuously using time-varying data from the application environment, make decisions based on probabilistic distributions, and employ statistical evaluation methods. A notable feature of the episodes is that they are hard to define. As a result, it is hard to retain and store a complete solution or a trace of execution of a HM, thereby requiring the eligibility values of decision-making parameters to be computed online. Moreover, since a smaller amount of information is extracted in each cycle as compared to Dietterich and Buchanan's model, learning tends to take longer.

The model is again shown in Figure 2; actions carried out are shown as boxes in the middle in each block. The *Problem Solver* applies static stochastic decision rules to generate a decision, whose effect on the environment may be delayed. The *Learning Performance Database and Preprocessor* computes incrementally the eligibility of decision-rule parameters using the current feedback signal. Credit assignment is then performed on the global feedback signal. Temporal credit assignment is the major problem to be addressed due to the evaluative feedback and the time-varying nature of the data measured in the environment. It is carried out by the *Adaptive Critic* (also called the Feedback Predictor or the Secondary Reinforcement Device), which learns online to produce rewards (or reinforcements) in the presence of delayed and evaluative feedback. It does not attempt to project into future states, as in Dietterich and Buchanan's model, due to the lack of a causal model. In existing implementations, the feedback signals and states are assumed to follow a Markovian model, which simplifies the maintenance of past states and temporal credit assignment. Structural credit assignment is performed next using associative learning that correlates the decision and the corresponding feedback signal. The result is a set of eligibility values of decision-making parameters in the HM to be learned, which are then applied in the Learning Element. Examples of systems in this class include Klopf's drive-reinforcement model [22] and Sutton and Barto's reinforcement model [54].

3.3. Hybrid Learning Model

There are problems that cannot be solved by either Dietterich and Buchanan's or Minsky's model alone. A problem in this class is characterized by a knowledge-lean learning environment with an ill-defined objective and evaluative feedback, and a non-Markovian temporal scope that can be limited and defined heuristically by a temporal model. Minsky's model is not applicable in this case since it lacks the reasoning mechanism for distributing credit among explicitly stored past decisions. Dietterich and Buchanan's model is not applicable because it requires a well-defined objective with prescriptive feedback. These limitations lead to the development of a hybrid learning model.

The actions carried out in the hybrid model are shown as boxes in the bottom in each block in Figure 2. Since the causal model is usually heuristic and less detailed than that in Dietterich and Buchanan's model, it does not provide post-conditions for finding the result of a decision sequence; rather, it is used mainly for identifying candidate decisions during temporal credit assignment. Decisions, in this case, are made one at a time, and temporal credit assignment is performed heuristically on a stored solution sequence by computing an eligibility value for each individual decision in the history. The hybrid model is characterized by Samuel's Checker Player [44, 45], EURISKO [26], Williams' REINFORCEMENT model [62], LS-1 [51], classifier system [16], and the truck-backer-upper problem of Widrow, *et al* [39].

3.4. Summary

To summarize, we present three point-based learning models in this section. For knowledge-rich environments with prescriptive feedback and a causal model, Dietterich and Buchanan's model can be used. This learning model is not applicable for knowledge-lean environments, which are usually associated with evaluative (and sometimes delayed) feedback and problems with ill-defined objectives. For knowledge-lean environments with Markovian state changes, Minsky's model can be applied. The Markovian property simplifies temporal credit assignment, allowing eligibility values that correspond directly to the decision-making parameters to be computed online. For knowledge-lean environments with non-Markovian state changes but with an approximate temporal model, the hybrid learning model can be used. This learning model works well when the heuristic temporal model is reasonably accurate, and when the time-varying parameters have stationary distributions.

For learning HMs with properties shown in Table 3, only the hybrid point-based model can be applied if there were an approximate temporal model relating decisions to feedback signals. However, additional learning techniques must be developed for coping with the lack of domain knowledge for structural credit assignment. In the next section, we present a population-based learning model that generates new HMs without relying on domain knowledge for structural credit assignment, and that decides the HM to be evaluated depending on test results obtained during learning.

4. INTEGRATED POPULATION-BASED AND POINT-BASED LEARNING MODEL

In this section, we present an integrated learning model that learns new HMs, starting from a pool of competing HMs and generating new ones when needed. This model is developed for learning HMs with properties described in Table 3, whose Problem Solvers lack an internal model of the decision process for structural credit assignment. Such is the case when the HMs are represented as decision procedures.

Population-based learning generates new HMs using mostly domain-independent knowledge and evaluative feedback signals: a new HM generated this way may have very different features from the original one. The learning system schedules tests of HMs dynamically, without committing to a particular incumbent HM for an extended period of time. Structural credit assignment is, therefore, avoided entirely, and intermediate performance results may be used to alter the probability of selecting a HM for evaluation, but not necessary to improve the incumbent HM. On the other hand, point-based learning can be applied when a particular HM is selected for evaluation. Using an approximate temporal model, a hybrid point-based learning method can explore some neighborhood of the initial HM, converging to the nearest locally best HM.

One of the major issues of population-based learning is the scheduling of a fixed amount of resources for testing HMs in order to find a good HM when the resources are expended. Resource scheduling is essential because the number of possible HMs for solving a target problem is very large and possibly infinite, and there are always resource constraints in a practical implementation. Under resource constraints, learning should not be focused on one incumbent HM at a time, as there is inadequate knowledge for structural credit assignment, and the penalty for following only one HM in the absence of such knowledge is large.

4.1. Components of the Learning Model

Figure 3 shows the major components of the learning model. We describe briefly in this section the function of each component, leaving the discussion on their design issues to Section 5. In this discussion, we assume that a HM is represented as a set of fixed decision rules and, in general, a set of fixed structures of HDEs.

In the integrated learning model, the time resource is divided into fixed-size pieces called quanta. At the beginning of a quantum, the *Resource Scheduler* selects either to test an existing HM in the population or to generate a new one. The decision on whether to generate a new HM depends on the resources remaining and the partial performance results of the HMs evaluated. If the decision is to generate a new HM, the *Heuristics Manager* will be invoked. Since the knowledge used for structural credit assignment is weak, the operators for composing new HMs are usually model-free and syntactic in nature and may perturb existing parameters and rules to obtain new ones. Examples of such operators have been studied in classifier systems and include mutation and crossover [6]. New HMs generated this way may be quite different from existing ones.

Once a HM is selected for testing, the *Test-Case Manager* is called to supply a set of test cases. These test cases may either be generated using domain knowledge of the application or be selected from an existing pool.

Using the set of test cases supplied by the Test-Case Manager and the HM selected by the Resource Scheduler, the system learns using the point-based hybrid learning model described in Section 3. In the quantum of time allowed, the *Problem Solver* evaluates the test cases one at a time, each involving the generation of a decision sequence that searches for the goal state. The performance results measured in the *Application Environment* after each test case is evaluated are used for assigning credit to decisions carried out in the decision sequence (temporal credit assignment). The algorithm for temporal credit assignment is similar to the ones used in genetic algorithms [13, 17, 63], and generally involves modification of the probability that a HDE will be invoked in future tests. Structural credit assignment for modifying the elements or the parameters of a HDE is difficult, as there is insufficient domain knowledge.

Testing of the HM selected will be terminated by the Resource Scheduler when either the intermediate performance results are not satisfactory or the time quantum is expended. In either case, the performance results obtained on the test cases evaluated in this quantum are normalized and processed by the Internal Critic. Using associative learning methods, the Internal Critic correlates the performance of this HM to the operators used in generating this HM. It then passes the updated aggregate measures, together with possible changes to the HM

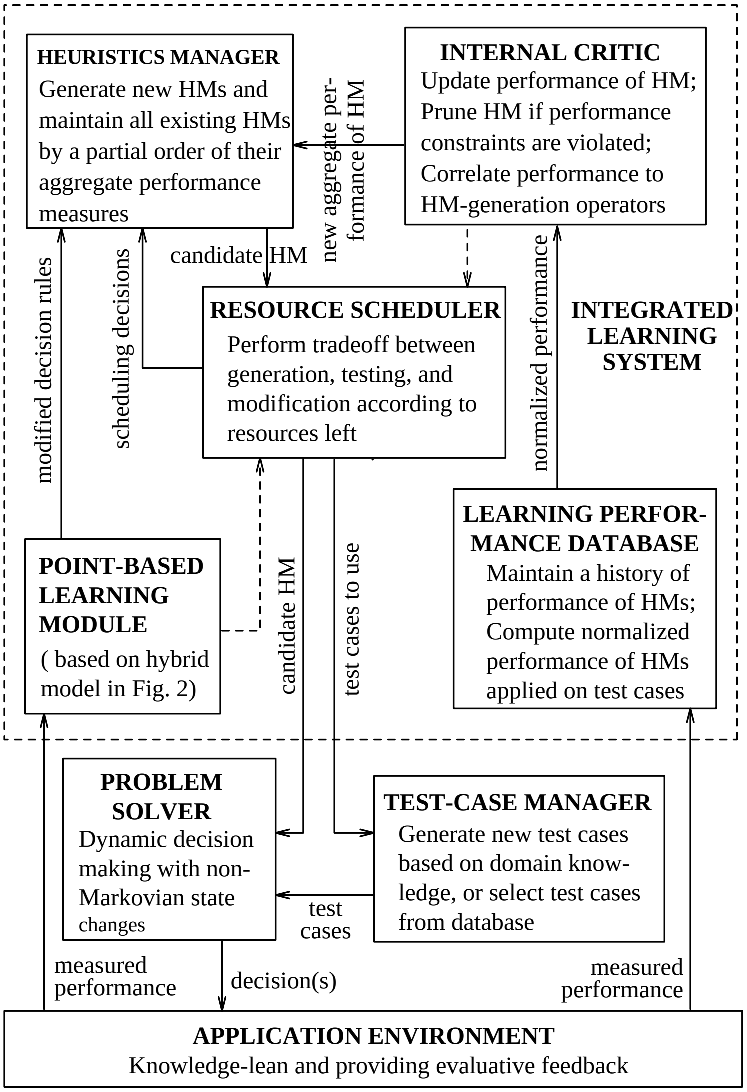

Figure 3. Integrated population-based and point-based learning model. (Dashed arrows indicate completion signals sent to the Resource Scheduler.)

---

concerned, to the Heuristics Manager. The cycle of learning is then repeated until the time allowed is exceeded.

In summary, there are two possible sequences of learning. Referring to Figure 3, the first sequence is based on a point-based approach that calls the Resource Scheduler, Problem Solver, and the hybrid point-based learning method shown in Figure 2. In this case, the Problem Solver evaluates a particular HM on a selected test case. Using evaluative feedback signals and a weak temporal credit assignment algorithm, the point-based learning system makes minor adjustments, and perhaps minor improvements, to the HDEs. The key issue here is on the design of the temporal credit assignment algorithm for assigning credit to each decision carried out in the decision sequence. On the other hand, the second sequence is based on a population-based approach that traverses the Resource Scheduler, Problem Solver, Learning Performance Database, Internal Critic, and Heuristics Manager.

Without relying on domain knowledge for structural credit assignment, it makes major adjustments to the HMs tested, and interleaves testing among different HMs. The key issue here is the scheduling of resources for testing various HMs so that a good HM can be found when resources are expended.

The learning model shown in Figure 3 has different emphasis, depending on whether learning is online or offline. Recall from Section 2 that online learning is used when the HM to be learned is scalable between small and realistic test cases, and that offline learning is used when it is possible to generate ahead of time the performance results of evaluating each realistic test case, using all possible decision sequences and starting from a given initial condition. If online learning is used, then test cases will actually be evaluated by invoking the Problem Solver and Test-Case Manager during learning, and may represent a significant overhead during learning. As a result, it is important to implement the Problem Solver and Test-Case Manager efficiently. Critical issues to be addressed include the study of good scheduling algorithms for allocating time and processor resources, and the design of an efficient Heuristics Manager and Internal Critic. In contrast, if offline learning is used, then the Problem Solver and Test-Case Manager will not be invoked during learning; consequently, resource scheduling is a secondary issue because the overhead is relatively small for looking up the performance results stored for a decision sequence. The critical issues to be studied are, therefore, the design of an efficient Heuristics Manager and Internal Critic.

4.2. Relationship to Classifier Systems

Our integrated learning model shares three basic features with the classifier-system approach in genetic learning [6]. First, a HM in our model corresponds to a policy (which can be represented as a rule-based system) in a classifier system; similarly, a HDE in our model corresponds to a rule in a policy. Second, we assume that our application domain is knowledge lean, as is the case in classifier systems. As a result, we employ weak and domain-independent methods for temporal credit assignment in the point-based learning component of our model. These methods are similar to the ones used in classifier systems, such as the bucket-brigade algorithm [17], Wilson's reinforcement algorithm [63], and the profit sharing plan [13]. Third, the Heuristics Manager in our model employs domain-independent operators for generating new HDEs and HMs; the counterparts are the mutation and crossover operators used in classifier systems for generating new rules and policies.

We have, however, studied three issues that were not addressed adequately in previous studies of classifier systems. First, we identify the temporal properties of HMs that must be satisfied in order for population-based learning to be possible. These conditions are summarized in Table 3 and apply to cases with or without delay between the initiation of a decision and the generation of its corresponding feedback. In contrast, policies in classifier systems are assumed to have no delay between a decision and its feedback. We survey previous learning methods in Section 3 and describe why they cannot be applied for learning HMs for application problems studied in this paper. We also differentiate between applications that should be learned online and ones that should be learned offline. Learning experiments demonstrating the above concepts are shown in Section 6.

Second, we study metrics for evaluating the performance of HMs. Specifically, we show in Section 5.1 that there may be anomalies in evaluation when multiple performance measures are combined into a single aggregate performance measure. We also propose new methods for resolving these anomalies during (Section 5.2) and after (Section 5.1) learning.

Third, we present in Section 5.3 algorithms for scheduling learning experiments, based on dynamic performance data obtained in previous experiments. Resource scheduling is important when learning has to be carried out within a limited time. In contrast, resource scheduling algorithms developed for classifier systems are static in nature, employing schedules that predefine the duration of a generation, the number of policies tested in a generation, and the number for tests performed on a policy. The static scheduling approach is demonstrated by the work of Grefenstette, *et al*., for learning sequential decision rules [14].

5. ISSUES ON THE INTEGRATED LEARNING MODEL

In this section, we discuss issues on designing a system that implements the integrated learning model presented in the last section. The issues considered include performance evaluation of HMs, acquisition and development of meta-heuristics used in the learning system, and resource scheduling. Meta-heuristics used are further classified into the ones for the Internal Critic, Heuristics Manager, and Test-Case Manager.

5.1. Performance Evaluation of Heuristic Methods

In this section, we study issues related to anomalies in performance evaluation of HMs and methods for coping with these anomalies.

5.1.1. Anomalies in Performance Evaluation

HMs are compared on the basis of their performance, which is related to the objective of the target problem to be solved. As is discussed in Section 1, the objective of a HM, as well as its tradeoff, are usually unknown with respect to its performance measures. A possible solution to cope with this issue is to derive the objective function as an aggregate performance function with tunable parameters, and to find a particular combination of values of these parameters that lead to the HM with the best tradeoff. We applied this approach in our earlier work, using objectives based on parametrized functions of ratio of average quality to average cost [18] and ratio of normalized quality to normalized cost [19]. This approach fails for three reasons.

First, it is very difficult to find a particular combination of values of parameters in the aggregate performance function so that when applied in the integrated learning system, the HM with the best tradeoff between cost and quality can be found.

Second, it is difficult to compare the performance of two HMs when they are evaluated by test cases of different sizes or parameters. In this case, the evaluation results may lead to a wrong conclusion that a HM evaluated on test cases of smaller size is better than other HMs evaluated on larger test cases.

Table 4. Example to illustrate anomalies in performance evaluation of HMs. HM 1 is better than HM 2 with respect to aggregate performance measures $\overline{Q}/\overline{C}$ and $\overline{\pi}$; HM 2 is better than HM 1 with respect to $\overline{P}$ and $\overline{\sigma}$; the aggregate measure $\overline{S}$ shows that both HMs have speedups greater than 1 with respect to the other.

HM	Performance	Test Case j					Average Measure
i	Measure	1	2	3	4	5	Over All j's
1	$Q_{1,j}$	10	20	15	10	30	$\overline{Q}_1 = 17$
	$C_{1,j}$	10	20	20	10	50	$\overline{C}_1 = 22$
	$P_{1,j} = \frac{Q_{1,j}}{C_{1,j}}$	1	1	0.75	1	0.6	$\overline{P}_1 = 0.87$
	$S_{1,j} = \frac{C_{2,j}}{C_{1,j}}$	2	1.4	0.5	1	0.2	$\overline{S}_1 = 1.02$
	$\pi_{1,j} = \frac{P_{1,j}}{P_{1,j} + P_{2,j}}$	0.67	0.70	0.43	0.5	0.29	$\overline{\pi}_1 = 0.52$
	$\sigma_{1,j} = \frac{S_{1,j}}{S_{1,j} + S_{2,j}}$	0.8	0.66	0.2	0.5	0.04	$\overline{\sigma}_1 = 0.44$
2	$Q_{2,j}$	10	12	10	10	15	$\overline{Q}_2 = 11.4$
	$C_{2,j}$	20	28	10	10	10	$\overline{C}_2 = 15.6$
	$P_{2,j} = \frac{Q_{2,j}}{C_{2,j}}$	0.5	0.43	1	1	1.5	$\overline{P}_2 = 0.89$
	$S_{2,j} = \frac{C_{1,j}}{C_{2,j}}$	0.5	0.71	2	1	5	$\overline{S}_2 = 1.84$
	$\pi_2 = \frac{P_{2,j}}{P_{1,j} + P_{2,j}}$	0.33	0.30	0.57	0.5	0.71	$\overline{\pi}_2 = 0.48$
	$\sigma_2 = \frac{S_{2,j}}{S_{1,j} + S_{2,j}}$	0.2	0.34	0.8	0.5	0.96	$\overline{\sigma}_2 = 0.56$

Third, there are anomalies in the evaluation of HMs when two HMs are compared using either different aggregate performance functions, or the same aggregate performance function with different parameters. Depending on the aggregate measures and the reference HM used, one HM can be shown to be better or worse than another.

To illustrate the above point, consider the hypothetical performance data of two HMs shown in Table 4. Let $Q_{i,j}$ and $C_{i,j}$ be, respectively, the quality and the cost of the j'th test case evaluated by HM i. The following observations are found from the data in Table 4.

a) $\overline{Q}_1/\overline{C}_1 > \overline{Q}_2/\overline{C}_2$ but $\overline{P}_2 > \overline{P}_1$: This shows that the evaluation based on ratios of averages can be inconsistent with the evaluation based on averages of ratios.

b) $\overline{S}_1 > 1$ and $\overline{S}_2 > 1$: This means that both HMs have speedups greater than 1 with respect to the other. Note that in computing the speedup of HM 1, HM 2 is used as the reference, whereas in computing the speedup of HM 2, HM 1 is used as the reference. What should have been done is that only one HM is chosen as the reference and be used consistently throughout learning and evaluation.

c) $\overline{\pi}_1 > \overline{\pi}_2$ but $\overline{P}_2 > \overline{P}_1$: To overcome the problem that values of different test cases are weighted differently in computing the average, one possible approach is to normalize the performance values before taking the average.

(The normalized performance value of a test case for HM 1 or 2 is found by dividing its performance value by the sum of the performance values of HMs 1 and 2.) In Table 4, $\bar{\pi}_i$ represents the average of normalized performance values for HM *i*. What we found is that the evaluation based on averages of normalized performance values can be inconsistent with the evaluation based on averages of raw performance values.

d) $\bar{\pi}_1 > \bar{\pi}_2$ but $\bar{\sigma}_2 > \bar{\sigma}_1$: This shows that the evaluation based on averages of normalized performance values can be inconsistent with the evaluation based on average normalized speedups.

5.1.2. Methods for Resolving Anomalies

Although the example above is shown with hypothetical data, it points out that there may be anomalies in using aggregate functions for performance evaluation. To resolve these anomalies, we propose an approach based on consistent normalization and statistical analysis of the performance data. Our approach has the following steps.

a) Identify a *reference* or baseline HM upon which all other HMs are compared. This reference HM can be chosen to be the best existing HM for solving the given application problem.

b) Normalize each raw performance measure of a new HM *consistently* with respect to the same measure of the reference HM (evaluated on either the same set of test cases or test cases with the same distribution of performance) so that it is meaningful to compare two HMs based on their normalized measures. There are two approaches to normalization. First, all performance values of a measure of the new HM can be normalized with respect to a representative value of the same measure of the reference HM. Note that the representative value used must be invariant with time. To illustrate this approach, let the average quality and cost of the reference HM be $\bar{Q}_0$ and $\bar{C}_0$, respectively. The normalized values of quality and cost measures are shown in the top half of Table 5. The normalization constants used, namely, $\bar{Q}_0$ and $\bar{C}_0$, are invariant when more tests are performed on the reference HM. Note that if $\bar{Q}_0$ and $\bar{C}_0$ cannot be gotten directly, then unbiased estimates of their population means will be used instead. Second, a raw performance value of a measure of the new HM can be normalized with respect to the corresponding raw performance value of the same measure of the reference HM. This is illustrated in the lower half of Table 5. In this case, the ratio-of-quality (ROQ) and ratio-of-cost (ROC) are taken as the new performance measures. Note that the average ratio-of-quality (resp., ratio-of-cost) measure for the reference HM is 1.

c) Compare two HMs based on individual normalized performance measures. If an aggregate performance measure is used, then it is essential that the aggregate measure be computed from test cases of a single normalized measure but not of multiple normalized measures. For instance, if cost and quality values are available for two HMs, then the HMs can be compared with respect to cost and quality measures separately, but not with respect to aggregate measures such as cost-quality ratios. This step is necessary in order to avoid anomalies when performance is evaluated using parametrized aggregate functions of multiple performance measures. It implies that a multi-dimensional representation of performance measures is needed when the number of measures is greater than one. We propose in the following a graphical representation that shows each measure as a separate dimension of a multidimensional plot.

Figure 4a shows a two-dimensional scatter plot for displaying the normalized cost-quality data of an existing HM found by post-game analysis [64] (see Table 2). It also shows the normalized data for three new HMs we found earlier using population-based learning [19]. In our previous study, point-based learning was not applied to modify a HM in successive tests of the HM. This plot is obtained by applying the following steps.

First, for each set of test cases evaluated by a HM, we determine whether there are any statistical outliers in their quality and cost values. We compute standardized scores ($(x - \mu)/\sigma$) for quality and cost data individually, and consider a data point as a univariate outlier when one of its standardized scores exceeds a threshold, say 8. Note that this computation does not assume any underlying distribution of the data. An additional method, which we did not use, is to find multivariate outliers by computing the Mahalanobis distance [55] for each data point. A data point is considered an outlier when its Mahalanobis distance has less than, say, 0.001 probability of happening. The computation of the Mahalanobis distance requires the underlying data to be multivariate normal; hence, if normality is not true, then the next step must first be applied before computing the distance.

Next, we determine the distribution of the bivariate data points. We first check for normality of each of the univariate distribution by computing measures such as the skewness and the kurtosis [55], or by applying goodness-of-fit tests such as the Kolomogorov-Smirnov and the Geary tests, or by applying the Shapiro-Wilk test [12]. The joint distribution can also be checked for bivariate normality by computing the chi-square plot or the gamma plot [21] and a number of other methods [12]. If the performance values do not have a normal distribution, then a distribution must first be assumed in order to transform the values into another set so that they have a normal distribution. Note that this and the previous steps may need to be repeated a number of times in order to assure that the data points have a normal distribution and to remove outliers from the data points.

Finally, the contour of a constant-probability region is plotted for the bivariate distribution of normalized values in order to indicate the spread of the data. To uniquely characterize the region, we can compute its centroid, whose coordinates are found by calculating the averages of individual performance measures (assuming all points have equal weights). Note that the reference HM has centroid at (1,1). If the original normalized performance values are not normally distributed and a transformation was made in the last step, then the contour found in this step will have to be converted using the inverse transformation into one for the original data points.

Figure 4a shows the result after applying the above steps for the three new HMs learned. These HMs and the best existing HM (used as the reference) are evaluated with respect to test cases used during learning, test cases of the same size but not used during learning, and test cases of large size. The spread of the performance data for each HM evaluated on the same set of test cases is indicated by an elliptical region containing statistically 90 percent of the data points. Note that our normalization method allows performance to be compared with respect to test cases of different sizes and types.

The various thresholds in Figure 4a represent constraints on costs of mappings, normalized with respect to the average cost of the original post-game heuristics. A HM generated during learning is pruned if its average normalized cost exceeds the given threshold by 10 percents. This issue is explored in more detail in the discussion on

Table 5. Two consistent ways for normalizing quality and cost.

Normalization for Quality and Cost Measures						Average
Reference HM	Quality	$Q_{0,1}/\overline{Q}_0$	$Q_{0,2}/\overline{Q}_0$	...	$Q_{0,n}/\overline{Q}_0$	1
	Cost	$C_{0,1}/\overline{C}_0$	$C_{0,2}/\overline{C}_0$	...	$C_{0,n}/\overline{C}_0$	1
New HM	Quality	$Q_{1,1}/\overline{Q}_0$	$Q_{1,2}/\overline{Q}_0$	...	$Q_{1,n}/\overline{Q}_0$	$\overline{Q}_1/\overline{Q}_0$
	Cost	$C_{1,1}/\overline{C}_0$	$C_{1,2}/\overline{C}_0$	...	$C_{1,n}/\overline{C}_0$	$\overline{C}_1/\overline{C}_0$
Normalization for ROQ and ROC Measures						**Average**
Reference HM	ROQ	$Q_{0,1}/Q_{0,1}$	$Q_{0,2}/Q_{0,2}$	...	$Q_{0,n}/Q_{0,n}$	1
	ROC	$C_{0,1}/C_{0,1}$	$C_{0,2}/C_{0,2}$	...	$C_{0,n}/C_{0,n}$	1
New HM	ROQ	$Q_{1,1}/Q_{0,1}$	$Q_{1,2}/Q_{0,2}$	...	$Q_{1,n}/Q_{0,n}$	$1/n \sum_j Q_{1,j}/Q_{0,j}$
	ROC	$C_{1,1}/C_{0,1}$	$C_{1,2}/C_{0,2}$	...	$C_{1,n}/C_{0,n}$	$1/n \sum_j C_{1,j}/C_{0,j}$

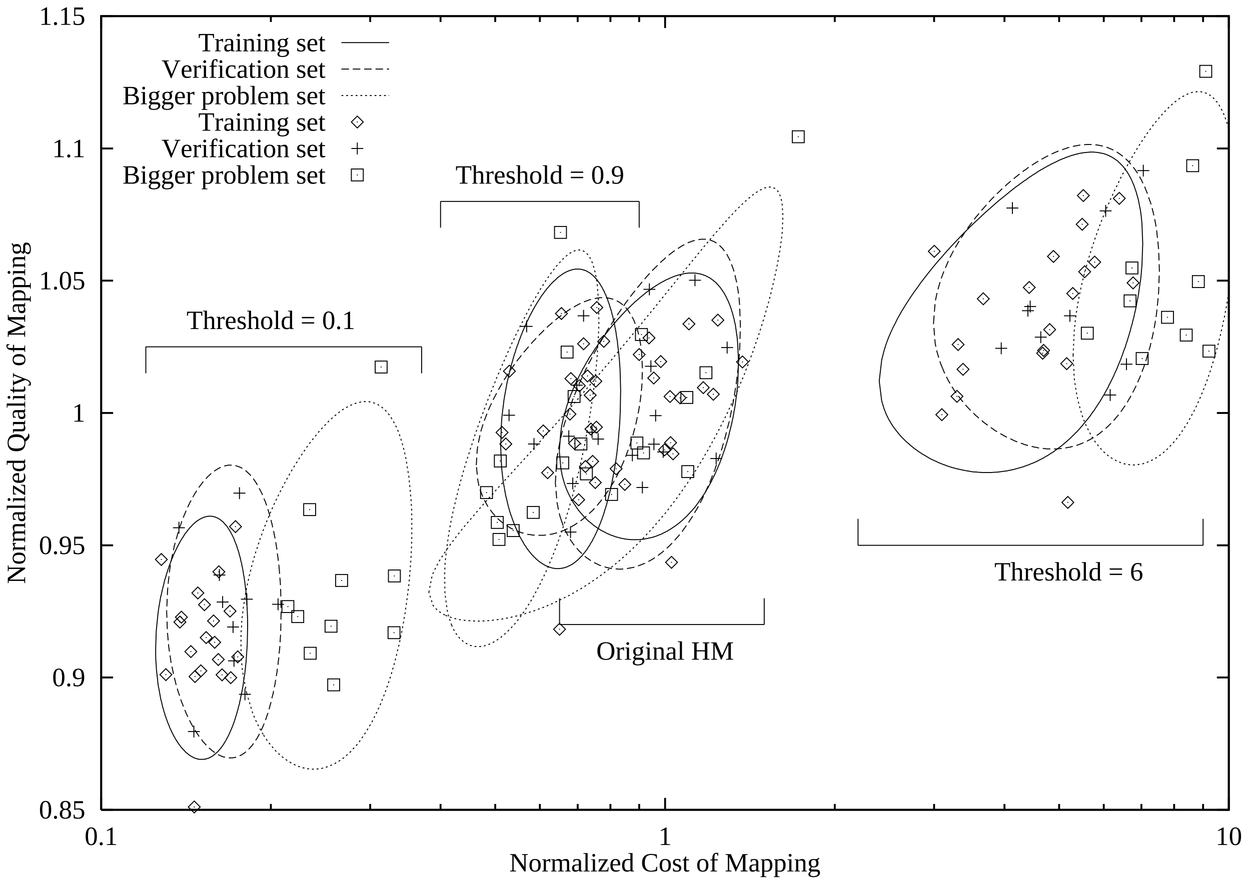

Figure 4a) Using normalized quality and cost measures.

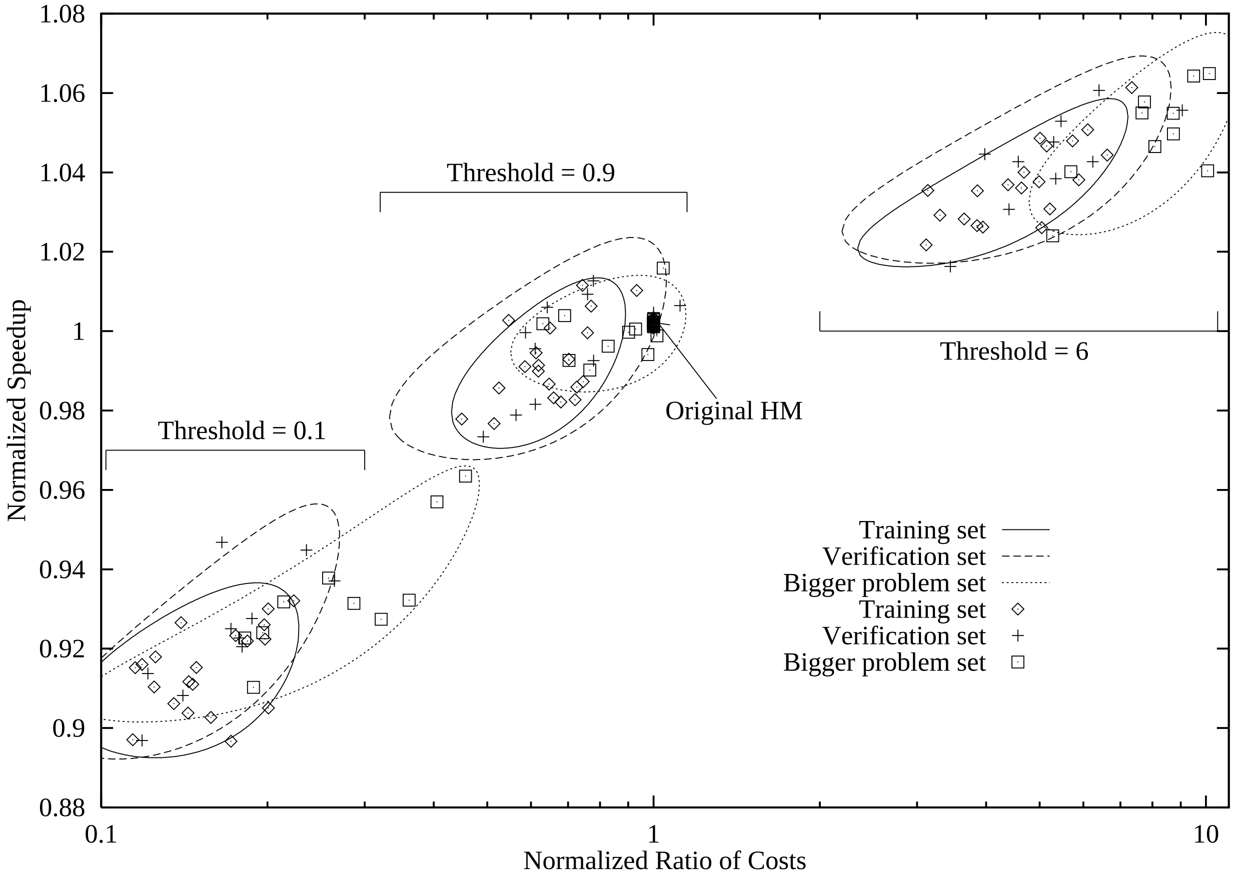

Figure 4b). Using normalized speedup and ratio-of-cost measures.

Figure 4. Scatter plots comparing three HMs learned by population-based learning with respect to a reference HM found by post-game analysis [64].

---

the Internal Critic in Section 5.2.

Figure 4a depicts clearly the relationship of one HM with respect to another and their scalability. Here, we do not try to prove scalability but to verify it statistically. Note that it is possible to use test cases of different types in the learning system to ensure that the HM learned can be generalized. This is discussed in Section 5.2.

The drawback of Figure 4a is that it only shows the distribution of performance values for different test cases but not the relationship for a particular test case between the new and the reference HMs. For instance, we cannot tell from Figure 4a, when a new HM has a centroid better than the centroid of the reference HM, whether the quality (resp. cost) of all or most of the test cases evaluated by the new HM are better than that of the reference HM. To cope with this problem, it is necessary to use a measure that relates the quality (resp. cost) of a new HM to the quality (resp. cost) of the reference HM for *a given test case*. Examples of such a measure are the ratio-of-quality (or speedup) and the ratio-of-cost measures shown in the lower half of Table 5. Recall that the quality of the process mapping problem is taken as the completion time of the processes mapped.

Figure 4b shows a scatter plot using the speedup and the ratio-of-cost as performance measures. Each data point represents the quality (resp. cost) of evaluating a test case by a new HM normalized with respect to the quality (resp. cost) of evaluating the same test case using the reference HM [64]. As is shown in Table 5, every data point of the reference HM after normalization are at (1,1). The thresholds in Figure 4b have similar meaning as the ones in Figure 4a, except that they are defined with respect the ratio-of-cost measure. Note that the objective of what constitutes a good HM in Figure 4b is different from that in Figure 4a, and that the three HMs depicted in Figure 4a are different from the ones depicted in Figure 4a.

5.2. Meta-Heuristics used in the Learning System

The meta-heuristics used in the learning system include the ones for the Internal Critic, Heuristics Manager, and Test-Case Manager. We describe the possible approaches in each area, leaving the details to the references.

5.2.1. Internal Critic

The Internal Critic serves two purposes. First, it relates the global evaluative feedback to the performance objectives of the HM tested. Due to a lack of domain knowledge for structural credit assignment, it is not able to correlate the feedback to individual elements of the HM tested. Rather, it updates the performance measures of this HM and determines whether the HM is to be discarded or be inserted into the HM pool in the Heuristics Manager for further testing in the future. If the HM tested is to be inserted into the HM pool, the Internal Critic should also rank this HM with respect to other HMs in the pool.

The second function of the Internal Critic is to associate the aggregate performance of the HM tested with respect to the effectiveness of the operators used in generating this HM. This information allows the Heuristics Manager to choose effective operators for composing new HMs in the future.

An important issue to be considered in designing the Internal Critic is the criteria used for determining the relative ‘‘strength’’ of the HM tested. One possible way we had attempted was to compute a performance index for each HM according to a *single* parametrized performance function of multiple performance measures. When learning began, a unique combination of parameters were assigned for this function. The performance index of each HM was updated after tests were performed, and HMs were selected for testing by the Resource Scheduler according to their indices. By repeating the learning process a number of times, each with a different combination of parameters, we hoped to find a combination that resulted in a HM with the proper tradeoff between cost and quality. We had applied this approach for learning new HMs for the process mapping problem shown in Table 2 [18, 19] and in tuning parameters in a stereo-vision algorithm (to be discussed in Section 6) [46]. Although we found improvements over the original HMs developed by human experts, this approach is fallible because, as is shown in the last subsection, whether a HM is better or not (that is, whether it has a higher relative ‘‘strength’’ or not) may depend on the parameters used in the single aggregate performance function.

To avoid anomalous ranking of HMs during learning due to the combination of parameters chosen, the objective used by the Internal Critic should be reformulated in such a way that it does not depend on the weight placed on any performance measure. An example of such an objective, as is discussed in Section 5.1 with respect to post-learning evaluation, is to constrain all (resp., all but one) of the normalized performance measures and find a HM that satisfies all the performance constraints (resp., satisfies all but one of the constraints and optimizes the one not constrained). A HM is pruned from further testing when one of its performance constraints is violated. Ranking of HMs used during learning is, therefore, consistent with the method proposed in the last subsection for the post-learning evaluation of HMs.

As an example, one possible objective in learning may be to find a new HM with the highest average quality, assuming that it is tolerable to have an average cost τ times as much as that of the original HM devised by human experts (chosen as the reference HM), where τ is a positive constant. In this case, the Internal Critic computes two aggregate performance measures, each based on an individual normalized performance measure, after testing for outliers and normalizing the performance data at the end of a quantum. The aggregate measures, for instance, can be the average normalized quality and average normalized cost of the region of scattered performance points. A HM is deemed unacceptable and pruned from further learning if its average normalized cost exceeds the constraint τ. HMs shown in Figure 4 were learned this way with τ taken as 1.1 times the threshold values.

Another objective in learning may be to find new HMs with smaller average costs and higher average quality than the ones of the original HM; in this case, a HM is pruned when either or both of the constraints are violated.

5.2.2. Heuristics Manager

The Heuristics Manager maintains existing HMs according to a partial order defined by the performance criteria used in the Internal Critic, and either selects an existing HM or generates a new one for testing when requested by the Resource Scheduler.

When a new HM is to be generated, the Heuristics Manager applies a combination of generation operators to generate one or more new HMs. The operators used are generally model free, since domain knowledge for structural credit assignment is missing. Common operators used are syntactic in nature, taking the structure of one HM and transforming it into another. It is important that the operators used be semantically sensible, generating new HDEs or HMs that are appropriate for the application. Checking for semantic sensibility is usually application-dependent and is hard to generalize.

The generation operators can be classified into ones designed for modifying HDEs and ones for creating new HMs. Operators that modify HDEs are generally grammar-based, transforming a HDE from one form to another. Given a HDE, three such operators are possible: a) *specialize*, in which one HDE is decomposed into multiple ones, each with more specialized preconditions, b) *generalize*, in which two HDEs with similar behavior are combined into a general one with more general preconditions, and c) *mutation*, in which a new HDE is created by perturbing parameter values of an existing HDE. These operators are similar to the ones used in classifier systems [14]. A partial history on the application of these operators may have to be kept to avoid repeatedly decomposing, recombining, and regenerating the same HDE.

Operators for creating new HMs, in the absence of a good model for structural credit assignment, are similar to the crossover operator used in classifier systems [14] for ‘‘breeding’’ new HMs. In creating a new HM, two promising HMs in the HM pool are first identified; HDEs from each are then clustered into two new HMs. The HDEs taken are either ones that, when applied individually or in a sequence, result in local improvement in performance, or ones that bring the solution closer to the final goal. Two contradicting goals need to be observed by the creation operator: a) it should avoid creating HMs with very similar performance, which usually happens when a large number of HDEs are duplicated in the new HMs generated; b) it should preserve HDEs that lead to high performance and use them as building blocks in the new HMs generated.

The generation process applies a combination of the operators described above to arrive at new HMs. In the beginning, it uses no historical information on performance and chooses the operators randomly. As more tests are performed, the operators used, as well as the HDEs or HMs to operate on, can be based on success rates of different operators or operator sequence applied in the past.

5.2.3. Test-Case Manager

The Test-Case Manager is responsible for maintaining a pool of valid test cases and for generating new test cases if necessary. The test cases used for evaluating HMs should satisfy two requirements. First, they should have a common statistical behavior, thereby making it meaningful to evaluate the statistical performance of HMs. Second, their performance must be known with respect to the reference HM. This is needed because all performance measures are normalized with respect to the corresponding measures of the reference HM, as is discussed in Section 5.1. Since the reference HM is likely to be less effective than the HM to be learned, evaluating the reference HM during learning may consume valuable learning time. As a result, the evaluation of the reference HM may have to be carried out before learning starts.

In designing the Test-Case Manager, two issues need to be considered. First, it must determine the size and type of test cases to be used. If the test cases are small, then many of them (and consequently HMs) can be evaluated during learning, but the HM derived from small test cases may not be scalable to larger ones. On the other hand, if the test cases are large, then fewer test cases and HMs can be evaluated in a fixed amount of time. Similar tradeoffs need to be made in deciding whether test cases from different applications should be used. If the HMs to be learned need to be scalable and be general, it may be necessary to use test cases of different sizes and from different applications. This requires more complex resource scheduling algorithms, as there may be large variations in the evaluation times for these test cases.

Second, the Test-Case Manager must decide for all HMs whether a random or a predefined order of test cases are to be used. A predefined order is preferable because the number of data points obtained during learning for a particular HM may be relatively small, and it may not be meaningful to compare two HMs statistically based on a small but different set of test cases. Moreover, the evaluation of two HMs using different test cases may require the evaluation of the reference HM on all the test cases used — an additional overhead needed before learning begins. It is also important that test cases used be statistically independent, and that the generation of test cases be unbiased. Statistical correlation of performance values may be used for detecting any bias.

5.3. Resource Scheduling

Resource scheduling entails the allocation of a limited amount of resources for generating and for testing HMs. Resource scheduling algorithms can be static or dynamic. A static scheduling algorithm determines, before learning begins, the number of generations in the learning process, the number of HMs to be tested in each generation, and the number of tests performed on each HM. Examples of static scheduling algorithms have been studied in classifier systems [6], in which tradeoffs have been evaluated with respect to different number of generate-and-test phases in a fixed amount of time [14] and different overheads in generation [10]. On the other hand, a dynamic resource scheduling algorithm determines dynamically, based on performance statistics obtained during learning, whether a new HM is to be generated or an existing HM is to be tested. This decision may depend on: a) whether existing HMs that perform well have been tested to an adequate degree of confidence, b) whether the performance of all existing HMs falls below a required minimum, c) whether there are an insufficient number of HMs in the population, and d) whether there are plenty of resources remaining. Dynamic scheduling algorithms are potentially better because they avoid testing HMs that have been found to be poor with some initial tests.

The top half of Table 6 shows the characteristics of the general problem for dynamically scheduling the generations and the tests of HMs. Assume without loss of generality that HMs are characterized by quality and cost, and that the objective of learning is to optimize one of the measures, say quality, while constraining the other. (The performance values of feasible HMs are, therefore, a collection, called a *population*, of quality values, with constraints on costs satisfied.) In the general model, performance analysis is difficult due to the fact that many of the parameters of the model are unknown a priori, and that point-based learning and heuristics generation introduce unknown effects on the search space of HMs. A possible approach is to develop efficient resource scheduling algorithms based on a simplified model and to use them as heuristic algorithms for the general model. The assumptions for a simplified but tractable statistical model are shown in the lower half of Table 6.

5.3.1. Related Work

An approach for scheduling resources in learning was studied by Russell and Wefald [43, 59]. The major differences between their work and ours are that they assume a knowledge-intensive domain (with at least a causal model of the environment defined), and a well-defined objective function for the application problem. The RALPH (Rational Agents with Limited Performance Hardware) project they developed aims to combine logicists' approach, which emphasizes correctness, and rationalists' approach, which emphasizes utility maximization. The compromise — bounded or limited rationality — is useful for real-time problem solving,[2] and can be stated as cost-quality

Table 6. General and approximate statistical models.

General Statistical Model

☐ Statistical distributions of populations are unknown and are possibly nonidentical.

☐ The distribution of means of populations is unknown.

☐ Applying point-based learning during testing of a HM leads to minor modification of the HM and a different population of performance values examined.

☐ Heuristics-generation algorithms (such as crossover, mutation, specialize, and generalize) incur a biased search of the space of HMs.

☐ Time to test a HM on a test case may be stochastic with an unknown and possibly population-dependent distribution.

Approximate but Tractable Statistical Model

☐ All populations have normal distributions with unknown means but constant variances.

☐ The means of all populations have a known distribution function with unknown parameters.

☐ Point-based learning makes negligible changes to the HM tested and, hence, the population of performance values examined.

☐ Heuristics-generation algorithms incur a random search of the space of HMs.

☐ The time to test a HM on a test case is constant. Alternatively, the average time may be used as an approximation.

tradeoffs for the problem solver. They considered scheduling not only alternative moves (called ‘‘actions’’) but also search strategies (called ‘‘computations’’ or ‘‘deliberations’’) that could help improve the incumbent’s expected utility. Learning was based on a search method called DTA* [59] that used a common evaluation function for choosing among alternatives. Their approach was to learn from past experience the probability of change in utility (resp. cost) of a move i, given that a computation j had been performed, for all i and j. Using the distributions learned, they derived the value estimated for a move or computation as the difference between its estimated benefit and its estimated cost.

A problem similar to the heuristics-selection problem we address here has been studied extensively as <u>selection problems</u> in statistical inference [4, 15]. These problems can be classified into two types.

The first type was first studied by Stein in 1948 [52] and is called the <u>stopping problem.</u> It deals with finding the minimum number of tests in order to know which population among a finite number of populations is the best to

2. Real-time problems in Russell and Wefald’s studies [43] are defined as those in which the utility of an action is a decreasing function of the time spent in finding that action. The tradeoff is represented as a time-cost function, which gives the loss in utility caused by delay in performing an action. We do not assume such a utility function in this paper.

within a certain degree of confidence. This result and subsequent extensions cannot be applied in our case because they considered a finite number of populations and an unbounded time.

The second type was pioneered by Bechhofer in 1954 [2, 3] and is called the allocation problem. It focuses on allocating a fixed number of tests among a given finite number of populations so that the probability of selecting the population with the maximum mean is maximized. Recent work in this area are predominantly on multistage methods in which the number of tests allowed are divided into stages, and promising populations selected in one stage are passed to the next stage for further testing. The allocation of tests can be static [5], which is determined a priori based on statistical distributions of populations, or can be dynamic [57], which is based on means and variances estimated during testing.

Results in the second type do not apply in our case because they deal with a finite and given number of populations before selection begins; our objective in HM selection, on the other hand, is to find a population (HM) with a large mean value but not necessarily the one with the largest mean, as there are infinitely many populations to start. However, existing statistical algorithms can be applied if the time allowed were divided into two parts: the first part is used for determining a finite number of populations to be tested in the second part, and the second parts assigns tests according to algorithms developed in statistical allocation.

5.3.2. Resource Scheduling Algorithms Studied

Based on the assumptions defined in the lower half of Table 6, we have developed a multistage selection algorithm for selecting a promising population within a time constraint. (Time is the only resource to be scheduled.) In this subsection, we summarizes our previous results in this area [18, 19].

In the general multistage algorithm, $G(T)$, stage i is characterized by a triplet consisting of a) g_i, the particular allocation algorithm to be used for this stage, b) t_i, the duration of the stage, and c) n_i, the number of populations to be considered for testing during the stage [18]. It starts with a presampling stage that estimates the statistical parameters of the target application. This is chosen heuristically to be 10 percent of the total time allowed. The next stage corresponds to coarse initial testing to weed out unworthy populations; each subsequent stage then evaluates populations not eliminated in previous stages and passes a small number of promising populations to the next stage. This general multistage algorithm covers both static and dynamic allocation algorithms.

Analysis of the general multistage algorithm centers on finding for each stage the joint probability distribution between $\bar{x}$, the sample mean of a population, μ, the sample mean of all population means, σ, the standard deviation of all population means, s, the number of samples drawn from each population, and any other variables used in selecting the next HM for testing [18]. Based on the assumptions stated in the lower half of Table 6, we have developed methods for analyzing both static and dynamic algorithms. However, analysis of dynamic algorithms is generally intractable because each pick of a population for testing must be treated as a separate stage.

We have studied and evaluated the following scheduling algorithms [19]:

- single-stage round-robin algorithm, where a predetermined number of populations are sampled a constant number of times in the time[3] allowed;
- two-stage round-robin algorithm, where analysis aims at finding the division of time between the first and the second stage and the parameters of the round-robin algorithm in each stage;
- two-stage round-robin/greedy algorithm, which uses (as heuristics) the same division of time found analytically for the two-stage round-robin algorithm;
- two-stage round-robin/minimum-risk algorithm, which is similar to the last algorithm except that it selects populations for testing in the second stage based on a risk function (defined as the expected squared-error loss of the estimated mean of each population).

For the last two algorithms, performance can be formulated but is too difficult to be solved in closed forms.

We have applied the above scheduling algorithms for learning new process mapping HMs for post-game analysis [64]. We first verified the assumptions stated in the lower half of Table 6 with respect to a randomly selected HM for process mapping. We developed a learning system on a DECStation 5000/200 workstation. Assuming a total CPU time of 16 hours, the available time is divided into two parts, with 12 hours and a round-robin scheduling algorithms in the first part, and 4 hours and a minimum-risk algorithms in the second part. Performance results of three new HMs learned are shown in Figure 4.

5.3.3. Possible Extensions

In general population-based learning, the scheduling algorithms we have studied so far are inadequate because they do not consider improvements caused by the Heuristics Generator and point-based learning. In both cases, evaluative feedback signals received are used to guide either the generation of a new HM or the refinement of an existing HM; the search for new HMs is, therefore, biased. It is this bias that helps learning systems improve over time; however, the statistical modeling of this effect is very hard. We expect that heuristic scheduling algorithms will be used in the general case. Some possible heuristic approaches for addressing this problem are as follows, assuming that heuristic scheduling algorithms are defined with respect to the number of HMs to be generated in each generate-and-test phase, the duration of the testing period, and the number of tests performed on each HM.

In determining the duration of a generate-and-test phase and the corresponding number of populations to be generated, we observe that the populations tested in previous phases may affect the populations to be generated or be tested in subsequent phases. If populations tested in each phase are generated randomly, then we have found that the optimal algorithm is to generate all the populations ahead of time before learning begins [20]. If populations are generated on demand, based on a posteriori distribution of population means found earlier, then Bayesian analysis

3. The time constraint is expressed as the total number of samples drawn, assuming that each draw takes unit time.

can be applied to capture the effect of changes in the a posteriori distribution of population means.

In finding how long a population is to be tested before switching to a different one, we observe that the distribution of populations tested changes with time when point-based learning is applied. In this case, we expect techniques in trend and time-series analyses to be useful for determining when testing should be stopped.

In summary, the scheduling of resources for an integrated learning system is complex because the space in which HMs are generated changes with time and may be hard to model. However, experimental results have shown that intermediate statistical measurements are useful for determining when to generate or what to test next, especially in a learning environment that lacks a world model. We expect that new insights can be gained by developing scheduling algorithms based on simplified models and applying these strategies as heuristics in practice.

6. EXAMPLE APPLICATIONS

In this section, we illustrate the techniques developed in this paper on four applications: a) load balancing of independent processes, b) automated design of artificial neural networks, c) depth perception in stereo vision, and d) combinatorial search using a branch-and-bound algorithm. These applications have many common features that allow new HMs to be learned by an integrated population-based and point-based learning method. However, the first two applications have unique idiosyncrasies that need special considerations: a) they both require the design of a performance predictor that predicts delayed feedback signals; b) they both require offline learning; and c) the design of load balancing HMs further require simplifying assumptions to be made in order to address the problem of overlapping and dependent temporal scopes. For the last two applications, online population-based learning can be applied in a similar way as in the mapping of communicating processes on distributed-memory computers (see Table 2). Note that our discussion only serves to illustrate difficulties involved in applying the learning method; implementations and performance evaluation of the learning experiments are generally application dependent.

6.1. Load Balancing of Independent Processes on a Distributed Computing System

In this subsection, we discuss issues and solutions for learning the design of adaptive and machine-dependent HMs for the load balancing problem defined in Table 1. There are three issues that we consider: the first is related to the development of an environment suitable for learning; and the last two are related to learning workload indices and load placement HMs.

6.1.1. Application Environment for Learning

The first issue concerns the reproduction of an application environment suitable for testing alternative placement and migration HMs. In our case, a technique is needed for reproducing a background workload and for adjusting the background workload in the presence of a test process. This is especially difficult in a time-sharing operating system as the execution of multiple processes may be interleaved in time. Desirable but prohibitively expensive techniques are ones that can reproduce the execution of every machine-level instruction. For practical reasons, we have developed a method that samples periodically the primitive measurable attributes in a computer system, such as disk traffic and memory occupancy (specifically, at every real-time clock interrupt), and that replays them later on by generating synthetic workload with the same behavior [29]. This sample-and-replay method has been found to be very accurate in approximating the physical-level system behavior.

6.1.2. Learning Workload Characterization

The second issue is on the design of a workload measure for identifying a location for a given process to be executed. This measure serves as the Performance Predictor in Figure 1. It is a complex function of a large number of machine-dependent, out-dated, and possibly dependent measurable attributes in the system. Existing techniques are based on either analysis, such as queuing models, or heuristic metrics, such as the number of pending processes, resource utilization levels, and ad hoc averages of queue lengths of selected resources [66].

We have developed a systematic method for learning a machine-dependent workload measure as a function of primitive measurable attributes [30]. Our design is based on an extension of Tesauro's comparator neural network [56] that generates an output indicating the *relative* speedup of a process with respect to a reference workstation, using inputs derived from windows of primitive measurable attributes before the process is executed. To simplify the discussion, consider a network of homogeneous workstations; in this case, the design of the comparator network is identical for all sites. In order to collect the training patterns for the comparator network, we run a set of test processes, one at a time, under a variety of controlled workload conditions on one workstation. A training pattern is then made up of the completion times of a test process run under two different workload conditions, which are characterized by windows of measurable attributes before the execution of the process. To reduce the amount of input information to the comparator neural network, we have developed a scheme for filtering the windows of workload attributes so that only their trends are used for predicting future workload conditions.

6.1.3. Learning Load Balancing HMs

Given the workload measures learned, the final issue is on learning automatically placement and migration HMs. Learning involves the adjustment of parameters in a rule-based representation of the HM, which controls when migrations should be initiated. Structural credit assignment is difficult, as there is little background knowledge available (see Table 7). Temporal credit assignment is also difficult because process-placement decisions may be related in an unknown fashion and tend to have overlapping and dependent temporal scopes. Moreover, online learning is impractical because there are a large of number of alternatives for placing and for migrating a single process during the time that the process is executed.

In order to learn process placement and migration HMs by population-based learning, we need to make two simplifying assumptions so that these HMs satisfy the requirements stated in Table 3 for temporal credit assignment. First, we need to assure that performance measures for evaluating a test case are independent of the intermediate decisions and internal states. This condition will be valid if we restrict the temporal scope of a placement/migration

Table 7. Learning HMs for the load balancing problem defined in Table 1.

Credit assignment of the original problem:
☐ *Structural*: Difficult, as domain is knowledge lean.
☐ *Temporal*: Episodic, but load balancing decisions may be dependent and have overlapping temporal scopes.
Simplified problem:
☐ Restrict the load balancing decision to one process at a time, independent of all future migration decisions for the given process; all other processes in the system are considered as background workload and are not modeled specifically.
☐ Correct wrong decisions by dynamic process migrations.
Offline learning:
☐ Rules learned are not scalable between small and large processes.
☐ Large processes have large temporal scopes.
☐ The possible decisions at a given time are limited by the number of computers in the system.
☐ Alternative decisions can be tested by regenerating an identical system-wide workload.

decision to be only between this decision and the next placement/migration decision for any process in the system. This assumption implies that at every decision point in the system, the placements of all newly arrived processes and partially executed processes are considered anew. It is true when the overhead for migrating a process is negligible. Note that the overhead for migration can be accounted for indirectly by the migration rules, which migrate a process only when there is a reasonable difference in workloads between two computers. Second, we need to assure that the aggregate performance measures are independent of the order of evaluating the test cases. This condition will be valid if we assume that the future background workload in the system is not affected by a placement or migration decision. This assumption allows the placement and migration rules to be learned one process at a time in the presence of a predefined background workload. It is debatable, since one placement decision may affect other placement decisions, which in turn affect the background workload. However, in the absence of information about future workloads and arrivals, and the fact that errors in placements can be corrected later by migrations, this assumption is a reasonable approximation.

The above assumptions reduce the temporal credit assignment to one process at a time: the response-time feedback signal observed for one process is used to modify the placement and migration rules used in arriving at the decision for this process, and all other processes executed in the system are modeled as background workload. Learning is, therefore, episodic, and the performance measure of a test case (in terms of the completion time of the test process) is independent of intermediate decisions and states. Further, the aggregate performance measure of the process placement and migration system (in terms of the average completion time of a set of test processes) is independent of the order of presentation of test cases.

The simplified problem can now be learned by offline population-based learning. Learning begins by collecting the completion times of executing a given set of test processes, one at a time, on all possible sites in the network. By assessing the workloads for all sites before the execution of a process in the set (using the comparator neural network designed) and by knowing the completion times of the process for each site, we can apply population-based learning to tune the parameters used in the load balancing system. Learning can be carried out offline because a) all possible placement decisions are limited by the number of computers in the system; b) an identical system-wide workload can be regenerated so that performance due to alternative decisions, starting from a given initial workload condition, can be evaluated beforehand; and c) for a given set of rules for detecting when migration is to be initiated, the completion times of the test process for all possible locations for migrating the process can be evaluated offline.

6.2. Configuration Design of Artificial Neural Networks

In this section, we apply our integrated learning method for designing an appropriate configuration of a feed-forward artificial neural network (ANN) for an application problem. The target application is specified by a set of input-output patterns, and the design problem is to find an ANN that responds correctly to any input pattern in the set (and perhaps generalizes its outputs for new input patterns). The objective of the design process is to identify a ‘‘good’’ ANN configuration (but not necessarily train it to completion) using a fixed amount of computational resources. Table 8 summarizes the issues for this problem.

6.2.1. Existing Methods

Traditionally, a feed-forward ANN can be designed by first hypothesizing a network configuration with some initial weights, and by applying the back-error propagation procedure [28, 42] or its variations to ‘‘learn’’ the correct weights for the network. It is generally assumed that the network is trained when its total sum-of-squared error (TSSE) between the actual and the desired outputs falls below a maximum threshold, even though training may have only converged to a local optimum. If training does not converge within a reasonable amount of time, then heuristic schemes will be applied to either pick a new configuration or modify an existing configuration before further training. Examples of these heuristic schemes can be classified into five types: ad hoc method [42], iterative refinement method [8], start-big-and-remove method [24, 37, 47], start-small-and-add method [9, 49], and iterative refinement with backtracking [58]. All these schemes except the last one are based on a greedy search that only examines one incumbent network at any time. The last scheme examines multiple partially trained networks simultaneously; however, it lacks a general measure for identifying promising configurations and a procedure for scheduling computational resources.

The traditional design process can be considered as a hierarchy of two levels of point-based learning. Let an ANN be a HM for solving the application problem. At the lower level, we can view the test cases as the set of

Table 8. The automated design of an ANN.

Definition: Given a set of input-output patterns of the target application, the problem is to find a feed-forward ANN trained by the back-error-propagation procedure so that the TSSE between the actual and the desired outputs of the ANN is below a maximum threshold.
Characteristics:
☐ The set of measurable parameters are ill-defined.
☐ Relationship is undefined between the cost (or complexity) of the network and its quality (or TSSE) when training is terminated.
Credit assignment in training a given network:
☐ Structural: Defined by the back-error propagation procedure.
☐ Temporal: None, as feedback is immediate.
Credit assignment in the design process:
☐ *Structural*: Ad hoc, due to a lack of domain knowledge.
☐ *Temporal*: Episodic, but the temporal scope of a decision to modify a network is as long as the time it takes to train the network.
Online learning for the design process:
☐ Use existing schemes [8, 9, 24, 37, 42, 47, 49] for modifying network configurations in the Heuristics Manager.
☐ Use a performance predictor for predicting missing feedback signals.

training patterns specified for the application, a decision in a HM as the selection of a training pattern, and an episode as the time for training the network with one training pattern using the back-error propagation procedure. Structural credit assignment for the ANN selected for training is provided by the back-error propagation procedure. Temporal credit assignment, on the other hand, is not necessary because the feedback signal for each input pattern trained is available without delay.

At the higher level, the design process modifies the ANN selected for training when little improvement on TSSE is found. Many such existing schemes [8, 9, 24, 37, 42, 47, 49] are point-based, focusing on one incumbent network at a time. These schemes use immediate evaluative feedback, based on past training performance of the network, to apportion credit structurally to components of the incumbent network. A decision is, therefore, an application of an operator for modifying the network, and its temporal scope ends when either the network is fully trained or time runs out. Temporal credit assignment is not addressed, as the final feedback (whether training converges or not) is related in a complex fashion to the intermediate decisions and states.

The above discussion reveals four related issues on the automated design of an ANN. First, there are infinitely many possible networks, each may take an exceedingly long time to train; hence, it is impossible to enumerate and train all of them to completion within a fixed amount of time, space, and resource constraints. Second, expert knowledge for predicting good networks is ad hoc in nature and may be application dependent, rendering it difficult to use in the design process. It is, therefore, essential to develop a way to capture and generalize this expert knowledge based on training examples for various applications. Third, the objective of the network to be designed is ill-defined, based on a subjective tradeoff between training time and network cost. Hence, it is important to have a design process that can propose alternate configurations under a variety of cost-quality tradeoffs. Last, the aggregate performance measure of the ANN designed, in terms of whether training converges or not, is not known until either the network is fully trained or time runs out. Further, it is difficult to relate this aggregate measure to the large number of intermediate performance measures available during training; examples of the latter include the current TSSE, the training time expended so far, and the rate of change of TSSE.

6.2.2. Integrated Learning of ANN Configurations

The issues discussed above can be resolved to a large extent by applying population-based and point-based learning in the design process. By maintaining a population of competing ANNs, the design system can defer decisions on the network to be trained, modifications to be performed, and new networks to be generated, until some partial training has been carried out on each. Population-based learning allows network modifications without relying on accurate domain knowledge for structural credit assignment. Existing schemes for modifying network configurations [8, 9, 24, 37, 42, 47, 49] can be used as operators in the Heuristics Manager; the particular operator to apply can be deferred until each network in the pool has been trained partially. On the other hand, point-based learning is carried out when a network is selected for training. As stated before, the back-error propagation algorithm provides structural credit assignment in the training process.

The unique difficulty with using population-based learning is that neither the conditions defined in Table 3 for online learning nor those defined for offline learning are satisfied. The design process cannot be learned offline because generating the state-space of realistic ANNs amounts to finding an ANN for solving the target application. The conditions defined for online population-based learning are not satisfied because the temporal scope of each episode is very large, and training cannot be performed on small networks and generalized to larger ones. In order for online population-based learning to be carried out, it is necessary to develop a heuristic performance predictor that discerns promising networks from poor ones with only partial training. This predictor estimates for the Internal Critic the relative aggregate performance of a given ANN as compared to the performance of other competing ANNs in the population.

The performance predictor needed here is similar to the one for workload characterization in load balancing. In this case, inputs to the predictor consist of time-varying patterns of TSSE values versus time for two different networks in the population, and the problem is to differentiate between them which one is more promising. Here, a network is more promising than another means that they both satisfy the given cost and complexity constraints, while the training time of the former is predicted to be smaller than the latter. The design of the performance predictor is simpler than that for the load balancing problem because traces of TSSE values versus training times for various ANNs can be collected ahead of time. Our initial results show that such a predictor can discern which one of two ANNs will converge faster with over 90 percent accuracy after each has been trained for a reasonable number of epochs.

6.3. Online Learning for Stereo Vision and Combinatorial Search

In this section, we discuss the application of online population-based learning for two new applications. The way that population-based learning is applied here is similar to that for the mapping of processes on a network of distributed-memory computers we discussed earlier [19] (see Table 2 and Figure 4).

The first application deals with stereo vision, an inexpensive method for calculating the depth information in a scene. This is done by matching and triangulating the corresponding features in the scene taken from two different view points. A general stereo-vision algorithm refines iteratively the depth information by matching increasingly detailed features (tokens) found in the scene. This is necessary because with highly detailed scene descriptions, it is hard to find unique correspondence between the features without any initial depth knowledge. The iterative method is also considered to be a plausible approach used in biological vision systems [27].

There are two issues in stereo vision that are related to learning. First, there are a large number of parameters to be tuned in a stereo-vision algorithm. The values of these parameters, as well as the granularity of the features and the number of interactive refinements, are influenced strongly by the imaging domain. It is, therefore, essential to develop a systematic method for tuning these parameters in limited time. Second, the objective of a stereo-vision algorithm is ill-defined with respect to measurable attributes in the system, such as speed of the algorithm, density of depth information determined, and accuracy of depth measurements. Hence, it is important to have a design process that can propose alternate parameter values for a variety of cost-quality tradeoffs.

The HMs for this problem are suitable to be learned by an online integrated learning approach for similar reasons as the process mapping problem (see Table 2). Our initial results show that highly accurate and efficient HMs can be found by population-based learning [46].

The second application deals with combinatorial search problems that can be solved by a branch-and-bound search. A branch-and-bound algorithm is an organized search of the space of all feasible solutions for solving a combinatorial optimization problem. By repeatedly partitioning the space of all feasible solutions into smaller and smaller subsets until either all feasible solutions are found, or infeasibility/dominance is encountered, the process guarantees that the optimal solution is found.

There are many existing HMs for exploring the search tree; examples of HMs amenable to learning are the selection function for defining what node to expand next, the decomposition function for decomposing a search node into descendent search nodes, and the heuristic pruning function for identifying unpromising search nodes. Each of these functions can be formulated in terms of parameters that can be measured during the search. For reasons similar to the ones shown in Table 2, these functions can be learned by online integrated learning methods.

Note that HMs in this application have only one performance measure, namely, the number of nodes searched before the solution is found, and that they are all performance related. Population-based learning method is not suitable for learning correctness-related heuristics, such as dominance relations [65].

Our initial results on learning new decomposition functions show that significant reduction in the search overhead can be attained. One problem with the integrated learning method for this application is that the functions learned may not be semantically meaningful: it is possible that functions learned are tailored to the specific test cases used. Consequently, it is important that these functions be interpreted after they are learned; if necessary, the learning process will need to be repeated using different test cases.

7. CONCLUSIONS

In this paper, we have presented a learning-by-example method that integrates a point-based approach with a population-based approach for learning new HMs for problem solving. Traditional learning approaches are generally point-based in nature, focusing on modifications of an incumbent HM. In a population-based learning approach, a population of competing HMs is maintained; the choice of the HM to test depends on partial results of testing the HMs on test cases.

We have focused on learning performance-related HMs that involves tradeoffs between quality of solution found and overhead (or cost) incurred in finding the solution. We are interested in HMs that do not have a well-defined relationship between quality and cost. We have defined the characteristics of application problems whose HMs can be learned by the proposed learning method (see Table 3). We have addressed issues of anomalies in performance evaluation of HMs, criteria for evaluating HMs during and after learning, acquisition of meta-knowledge for the learning system, and design of efficient scheduling algorithms for finding good HMs in limited resources.

We have illustrated our learning method on five applications. HMs for three of these applications (processing mapping, stereo vision, and combinatorial search) can be learned in a straightforward way. Two other applications that require special considerations are load balancing of processes for a distributed network of workstations and automated design of artificial neural networks. In the load balancing application, temporal scopes of episodes interfere with each other. This problem has been solved by making reasonable assumptions on the type of load balancing and migration HMs to be learned and by redefining the meaning of episodes. The temporal scopes of episodes can then be partitioned so that they do not interfere with each other. In the neural-network-design problem, the temporal scope of an episode is as long as the time it takes to find an appropriate network. To cope with this problem, an offline design of a performance predictor is needed. The predictor predicts delayed feedback signals to be available in the future, and estimates for the Internal Critic the training time of a given neural network relative to another in the population. Note that when the predictor is very accurate, population-based learning degenerates to be point-based.

In summary, we have addressed in this paper the following key issues:

- methods for coping with a lack of domain knowledge for structural credit assignment in learning performance-related HMs;
- temporal properties of HMs that must be satisfied in order for population-based learning to be possible;
- methods for resolving anomalous behavior in performance evaluation; and
- efficient scheduling of computational resources for learning.

Our solutions to these issues extends previous results on statistical allocation and learning in classifier systems. We have shown that population-based learning is a viable alternative to structural credit assignment, and that population-based learning methods are widely applicable and their conditions of applicability are characterized mainly by the nature of their episodes.

ACKNOWLEDGEMENTS

The author is indebted to many fruitful insights and help from research staff in his group in preparing this paper. In particular, the author would like to thank P. Mehra, A. Ieumwananonthachai, A. Aizawa, L. Chu, T. Karnik, and S. Schwartz for their discussions, and to A. Ieumwananonthachai for generating the results on learning new heuristic methods for process mapping. The author would like to thank Prof. H. Tanaka and Fujitsu Limited, who provided support when the author was visiting the University of Tokyo, during which part of this paper was written.

[1] A. Barr and E. A. Feigenbaum, *The Handbook of Artificial Intelligence*, vol. 1, 2, and 3, William Kaufmann, Los Altos, CA, 1981, 1982.

[2] R. E. Bechhofer, C. W. Dunnett, and M. Sobel, ‘‘A Two-sample Multiple Decision Procedure for Ranking Means of Normal Populations with a Common Unknown Variance,’’ *Biometrika*, vol. 41, pp. 170-176, 1954.

[3] R. E. Bechhofer, ‘‘A Single-Sample Multiple Decision Procedure for Ranking Means of Normal Populations with Known Variances,’’ *Ann. Math. Statist.*, vol. 25, no. 1, pp. 16-39, Institute of Mathematical Statistics, Ann Arbor, MI, March 1954.

[4] R. E. Bechhofer, J. Kiefer, and M. Sobel, *Sequential Identification and Ranking Procedures*, Univ. of Chicago Press, Chicago, IL, 1968.

[5] R. E. Bechhofer, A. J. Hayter, and A. C. Tamhane, ‘‘Designing Experiments for Selecting the Largest Normal Mean when the Variances are Known and Unequal: Optimal Sample Size Allocation,’’ *J. of Statistical Planning and Inference*, vol. 28, pp. 271-289, Elsevier Science Pub., 1991.

[6] L. B. Booker, D. E. Goldberg, and J. H. Holland, ‘‘Classifier Systems and Genetic Algorithms,’’ *Artificial Intelligence*, vol. 40, no. 1, pp. 235-282, North Holland, 1989.

[7] T. G. Dietterich and B. G. Buchanan, *The Role of Critic in Learning Systems*, Tech. Rep. STAN-CS-81-891, Stanford Univ., CA, Dec. 1981.

[8] S. E. Fahlman, ‘‘Faster-Learning Variations on Back-Propagation: An Empirical Study,’’ *Proc. Connectionist Models Summer School*, pp. 38-51, Morgan Kaufmann, Palo Alto, CA, 1988.

[9] S. E. Fahlman and Christian Lebiere, ‘‘The Cascade-Correlation Learning Architecture,’’ in *Advances in Neural Information Processing Systems 2*, ed. D. S. Touretzky, pp. 524-532, Morgan Kaufmann, San Mateo, 1990.

[10] J. M. Fitzpatrick and J. J. Grefenstette, ‘‘Genetic Algorithms in Noisy Environments,’’ *Machine Learning*, vol. 3, no. 2/3, pp. 101-120, Kluwer Academic Pub., Boston, MA, Oct. 1988.

[11] M. P. Georgeff, ‘‘Strategies in Heuristic Search,’’ *Artificial Intelligence*, vol. 20, pp. 393-425, North Holland, 1983.

[12] R. Gnanadesikan, *Methods for Statistical Data Analysis of Multivariate Observations*, John Wiley & Sons, New York, NY, 1977.

[13] J.J. Grefenstette, ‘‘Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms,’’ *Machine Learning*, vol. 3, no. 2/3, pp. 225-246, Kluwer Academic Pub., Boston, MA, Oct. 1988.

[14] J. J. Grefenstette, C. L. Ramsey, and A. C. Schultz, ‘‘Learning Sequential Decision Rules using Simulation Models and Competition,’’ *Machine Learning*, vol. 5, pp. 355-381, Kluwer Academic Pub., Boston, MA, 1990.

[15] S. S. Gupta and S. Panchapakesan, ‘‘Sequential Ranking and Selection Procedures,’’ in *Handbook of Sequential Analysis*, ed. P. K. Sen, pp. 363-380, Dekker, NY, 1991.

[16] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, MI, 1975.

[17] J. H. Holland, ‘‘Properties of the Bucket Brigade Algorithm,’’ *Proc. Int’l. Conf. Genetic Algorithms and Their Applications*, pp. 1-7, The Robotics Inst. of Carnegie-Mellon Univ., Pittsburgh, PA, 1985.

[18] A. Ieumwananonthachai, A. N. Aizawa, S. R. Schwartz, B. W. Wah, and J. C. Yan, ‘‘Intelligent Mapping of Communicating Processes in Distributed Computing Systems,’’ *Proc. Supercomputing 91*, pp. 512-521, ACM/IEEE, Albuquerque, NM, Nov. 1991.

[19] A. Ieumwananonthachai, A. Aizawa, S. R. Schwartz, B. W. Wah, and J. C. Yan, ‘‘Intelligent Process Mapping Through Systematic Improvement of Heuristics,’’ *J. of Parallel and Distributed Computing*, vol. 15, pp. 118-142, Academic Press, June 1992.

[20] A. Ieumwananonthachai and B. W. Wah, ‘‘Parallel Statistical Selection in Multiprocessors,’’ *Proc. Int’l Conf. on Parallel Processing*, vol. III, pp. 190-194, Pennsylvania State Univ. Press, University Park, PA, Aug. 1992.

[21] R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[22] A. H. Klopf, ‘‘Drive-Reinforcement Learning: A Real-Time Learning Mechanism for Unsupervised Learning,’’ *Proc. Int’l Conf. on Neural Networks*, vol. II, pp. 441-445, IEEE, 1987.

[23] M. M. Kokar and W. W. Zadrozny, ‘‘A Logical Model of Machine Learning,’’ *Proc. First Workshop on Change of Representation*, 1988.

[24] J. K. Kruschke, ‘‘Creating Local and Distrtibuted Bottlenecks in Hidden Layers of Back-Propagation Networks,’’ *Proc. Connectionist Models Summer School*, pp. 120-126, Morgan Kaufmann, Palo Alto, CA, 1988.

[25] P. Langley, ‘‘Learning to Search: From Weak Methods to Domain-Specific Heuristics,’’ *Cognitive Science*, vol. 9, pp. 217-260, Ablex Pub. Co., Norwood, NJ, 1985.

[26] D. B. Lenat, ‘‘Theory Formation by Heuristic Search; The Nature of Heuristics II: Background and Examples,’’ *Artificial Intelligence*, vol. 21, pp. 31-59, North Holland, 1983.

[27] D. Marr, in *Vision*, W. H. Freeman and Co., New York, NY, 1982.

[28] J. L McClelland and D. E. Rumelhart, *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises,* Vol. 3, MIT Press, Cambridge, MA, 1988.

[29] P. Mehra and B. W. Wah, ‘‘Physical-Level Synthetic Workload Generation for Load-Balancing Experiments,’’ *Proc. First Symposium on High Performance Distributed Computing*, pp. 208-217, IEEE, Syracuse, NY, Sept. 1992.

[30] P. Mehra and B. W. Wah, ‘‘Adaptive Load-Balancing Strategies for Distributed Systems,’’ *Proc. 2nd Int’l Conf. on Systems Integration*, pp. 666-675, IEEE Computer Society, Morristown, NJ, June 1992.

[31] R. S. Michalski, ‘‘A Theory and Methodology of Inductive Learning,’’ in *Machine Learning*, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Tioga, 1983.

[32] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (ed.), *Machine Learning: An Artificial Intelligence Approach,* 1, 2, William Kaufmann, Los Altos, CA, 1983, 1985.

[33] M. Minsky, ‘‘Steps Toward Artificial Intelligence,’’ in *Computers and Thought*, ed. E. A. Feigenbaum and J. Feldman, pp. 406-450, McGraw-Hill, New York, 1963.

[34] T. M. Mitchell, P. E. Utgoff, B. Nudel, and R. Benerji, ‘‘Learning Problem-Solving Heuristics Through Practice,’’ *Proc. 7th Int’l Joint Conf. on Artificial Intelligence*, pp. 127-134, William Kaufman, Los Altos, CA, 1981.

[35] T. M. Mitchell, ‘‘Learning and Problem Solving,’’ *Proc. 8th Int’l Joint Conf. on Artificial Intelligence*, pp. 1139-1151, William Kaufman, Los Altos, CA, Aug. 1983.

[36] T. M. Mitchell, P. E. Utgoff, and R. B. Banerji, ‘‘Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics,’’ in *Machine Learning*, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Tioga, 1983.

[37] M. C. Mozer and P. Smolensky, ‘‘Using Relevance to Reduce Network Size Automatically,’’ *Connection Science*, vol. 1, no. 1, pp. 3-16, Carfax Pub. Co., Oxfordshire, United Kingdom, 1989.

[38] A. Newell, J. C. Shaw, and H. A. Simon, ‘‘Programming the Logic Theory Machine,’’ *Prof. 1957 Western Joint Computer Conf.*, pp. 230-240, IRE, 1957.

[39] D. Nguyen and B. Widrow, ‘‘The Truck Backer-Upper: An Example of Self-Learning in Neural Networks,’’ *Proc. Int’l Joint Conf. on Neural Networks*, vol. II, pp. 357-363, IEEE, 1989.

[40] J. Pearl, ‘‘On the Discovery and Generation of Certain Heuristics,’’ *The AI Magazine*, pp. 23-33, AAAI, Winter/Spring 1983.

[41] J. Pearl, *Heuristics--Intelligent Search Strategies for Computer Problem Solving,* Addison-Wesley, Reading, MA, 1984.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, ‘‘Learning Internal Representations by Error Propagation,’’ in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. D. E. Rumelhart, J. L McClelland and the PDP Research Group, vol. 1, pp. 318-362, MIT Press, Cambridge, MA, 1986.

[43] S. Russell and E. Wefald, ‘‘Principles of Metareasoning,’’ *Artificial Intelligence*, vol. 49, pp. 361-395, Elsevier, Amsterdam, 1991.

[44] A. L. Samuel, ‘‘Some Studies in Machine Learning Using the Game of Checkers,’’ *IBM J. Research and Development*, vol. 3, pp. 210-229, IBM, 1959.

[45] A. L. Samuel, ‘‘Some Studies in Machine Learing Using the Game of Checkers II--Recent Progress,’’ *J. of Research and Development*, vol. 11, no. 6, pp. 601-617, IBM, 1967.

[46] S. R. Schwartz, *Resource Constrained Parameter Tuning Applied to Stereo Vision*, M.Sc. Thesis, Dept. of Electrical and Computer Engineering, Univ. of Illinois, Urbana, IL, Aug. 1991.

[47] J. Sietsma and R. J. F. Dow, ‘‘Neural Net Pruning--Why and How?,’’ *Int’l Joint Conf. on Neural Networks*, vol. 1, pp. 325-333, IEEE, 1988.

[48] D. Sleeman, P. Langley, and T. M. Mitchell, ‘‘Learning from Solution Paths: An Approach to the Credit Assignment Problem,’’ *AI Magazine*, vol. 3, pp. 48-52, 1982.

[49] G. Smith, *Back Propagation with Dynamic Topology and Simple Activation Functions*, Tech. Rep. TR 90-12, School of Information Science and Technology, The Flinders Univ. of South Australia, Adelaide, 1990.

[50] R. G. Smith, T. M. Mitchell, R. A. Chestek, and B. G. Buchanan, ‘‘A Model for Learning Systems,’’ *Proc. 5th Int’l Joint Conf. on Artificial Intelligence*, pp. 338-343, William Kaufman, Los Altos, CA, Aug. 1977.

[51] S. F. Smith, ‘‘Flexible Learning of Problem Solving Heuristics through Adaptive Search,’’ *Proc. Int’l Joint Conf. on Artificial Intelligence*, pp. 422-5, Morgan Kaufman, 1983.

[52] C. Stein, ‘‘The Selection of the Largest of a Number of Means, Abstract,’’ *Ann. Math. Statist.*, vol. 19, p. 429, 1948.

[53] R. S. Sutton, *Temporal Credit Assignment in Reinforcement Learning*, Ph.D. Thesis, Univ. of Massachusetts, Amherst, MA, Feb. 1984.

[54] R. S. Sutton and A. G. Barto, ‘‘Toward a Modern Theory of Adaptive Networks: Expectation and Predicition,’’ *Psychological Review*, vol. 88, no. 2, pp. 135-170, American Psychological Association, Inc., Lancaster, PA, 1984.

[55] B. G. Tabachnick and L. S. Fidell, *Using Multivariate Statistics*, Second Edition, Harper and Row, New York, NY, 1989.

[56] G. Tesauro, ‘‘Connectionist Learning of Expert Backgammon Evaluations,’’ *Machine Learning*, pp. 200-206, Kluwer Academic Pub., Boston, MA, 1988.

[57] Y. L. Tong and D. E. Wetzell, ‘‘Allocation of Observations for Selecting the Best Normal Population,’’ in *Design of Experiments: Ranking and Selection*, ed. T. J. Santner and A. C. Tamhane, pp. 213-224, Marcel Dekker, New York, NY, 1984.

[58] B. W. Wah and H. Kriplani, ‘‘Resource Constrained Design of Artificial Neural Networks,’’ *Proc. Int’l Joint Conf. on Neural Networks*, vol. III, pp. 269-279, IEEE, June 1990.

[59] E.H. Wefald and S.J. Russell, ‘‘Adaptive Learning of Decision-Theoretic Search Control Knowledge,’’ *Machine Learning*, pp. 408-411, Kluwer Academic Pub., Boston, MA, 1989.

[60] B. Widrow, N. K. Gupta, and S. Maitra, ‘‘Punish/Reward: Learning with a Critic in Adaptive Threshold Systems,’’ *Trans. Systems, Man, and Cybernetics*, vol. SMC-3, no. 5, pp. 455-465, IEEE, 1973.

[61] D. C. Wilkins, ‘‘Knowledge Base Refinement Using Apprenticeship Learning Techniques,’’ *Proc. National Conf. on Artificial Intelligence AAAI-88*, 1988.

[62] R. J. Williams, ‘‘On the Use of Backpropagation in Associative Reinforcement Learning,’’ *Proc. Int’l Conf. on Neural Networks*, vol. I, pp. 263-270, IEEE, July 1988.

[63] S. W. Wilson, ‘‘Hierarchical Credit Allocation in Classifier Systems,’’ in *Genetic Algorithms and Simulated Annealing*, ed. L. Davis, Research Notes in Artificial Intelligence, Pitman, London, 1987.

[64] J. C. Yan and S. F. Lundstrom, ‘‘The Post-Game Analysis Framework--Developing Resource Management Strategies for Concurrent Systems,’’ *Trans. on Knowledge and Data Engineering*, vol. 1, no. 3, pp. 293-309, IEEE, Sept. 1989.

[65] C. F. Yu and B. W. Wah, ‘‘Learning Dominance Relations in Combinatorial Search Problems,’’ *Trans. on Software Engineering*, vol. SE-14, no. 8, pp. 1155-1175, IEEE, Aug. 1988.

[66] S. Zhou, *Performance Studies of Dynamic Load Balancing in Distributed Systems*, Tech. Rep. UCB/CSD 87/376 (Ph.D. Dissertation), Computer Science Division, Univ. of California, Berkeley, CA, 1987.