JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 15, 118–142 (1992)

Intelligent Process Mapping through Systematic Improvement of Heuristics*

ARTHUR IEUMWANANONTHACHAI

Center for Reliable and High-Performance Computing, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, Urbana, Illinois 61801

AKIKO N. AIZAWA†

University of Illinois at Urbana–Champaign, Urbana, Illinois 61801

STEVEN R. SCHWARTZ

Motorola, Incorporated, MS IL27-G79, 1501 Shure Drive, Arlington Heights, Illinois 60004

BENJAMIN W. WAH‡

Center for Reliable and High-Performance Computing, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, Urbana, Illinois 61801

AND

JERRY C. YAN§

Sterling Federal Systems, Incorporated

In this paper, we present the design of a system for automatically learning and evaluating new heuristic methods that can be used to map a set of communicating processes on a network of computers. Our learning system is based on testing a population of competing heuristic methods within a fixed time constraint. We develop and analyze various resource scheduling strategies based on a statistical model that trades between the number of new heuristic methods considered and the amount of testing performed on each. We implement a prototype learning system (TEACHER 4.1) for learning new heuristic methods used in post-game analysis, a system that iteratively generates and refines mappings of a set of communicating processes on a network of computers. Our performance results show that a significant improvement can be obtained by a systematic exploration of the space of possible heuristic methods. © 1992 Academic Press, Inc.

* This research was supported in part by National Aeronautics and Space Administration Grant NCC 2-481 and NGT 50743 (NASA Graduate Fellowship Program) and in part by National Science Foundation Grant MIP 88-10584.

† On leave from National Center for Science Information System, Tokyo, Japan.

‡ Contact author (wah@aquinas.csl.uiuc.edu).

§ Present address: MS269-3, NASA Ames Research Center, Moffett Field, CA 94035.

1. INTRODUCTION

To meet the resource constraints and processing requirements of real-time computing systems, efficient scheduling of resources is necessary. Process mapping is an important element in real-time systems whose resources are limited and whose applications often have somewhat predictable behavior. The problem is NP-hard, and optimal mapping algorithms generally require an exponential amount of time and/or space. Heuristic methods have been applied, but their design is usually ad hoc, and their performance relies strongly on the characteristics of the target problem and the experience of the designers. Due to the nature of this problem, the number of possible heuristic methods is very large; as a result, some of these are likely to be better than the ones proposed by the experts.

Our goal in this paper is to develop a statistical method for exploring systematically the space of possible heuristics under specified time constraints. Our method intends to get the best heuristic method based on a tradeoff between its solution quality and its execution time. Learning of heuristics has been explored in our earlier work in the TEACHER 1.0 system [22], which learned dominance relations, TEACHER 2.0 system [12], which learned heuristics for selection in combinatorial search,

0743-7315/92 $5.00

TEACHER 3.0 [17], which learned the design of network configurations for artificial neural networks trained under a back-error propagation algorithm, and TEACHER 4.0 [18], which studied the scheduling of a learning-by-example paradigm under resource constraints.

Our focus in this paper is on mapping a set of communicating processes on a network of loosely coupled computers, with an objective of minimizing the completion time of the processes mapped. In our work, we assume that a process cannot be partitioned at run time, that the mapping is static, that a process is not replicated for execution on multiple computers, that the system is dedicated for one application at a time, and that the processing behavior of the application can be reproduced easily at design time. This last assumption is relaxed from the assumption that the processing requirements of the application are deterministic and known exactly at design time.

The post-game analysis system developed by Yan [19, 21] with Lundstrom [20] is one method that successfully deals with this problem. Given a set of communicating processes and a set of input data, the system incrementally modifies the mappings of processes using information collected in previous runs. Based on a set of heuristic decision rules (called *heuristic decision elements*), the post-game system follows three steps: (1) it formulates multiple optimization subgoals, based on timing data gathered from program execution, (2) it proposes new mappings, and (3) it prioritizes and resolves conflicting proposals. The main advantage of this system is that the target application is simulated and its mapping incrementally improved, *without* relying on any abstract program model or any particular objective function. Previous studies show that post-game analysis can consistently out-perform random placement, dynamic load balancing, and clustering algorithms by as much as 15% in process completion times.

The heuristic decision elements used in post-game analysis are developed by designers based on previous experience on designing similar algorithms. They do not adapt to different computing conditions, such as hardware configuration, average workload per processor, and mix of computation and communication of processes. As a result, it is possible to improve the performance of these heuristic methods, especially when the methods are applied in situations different from those that they were designed for. In this paper, we develop a learning framework for systematically generating and testing new heuristic methods used in post-game analysis for proposing new mappings. As there are infinitely many possible heuristic methods and time is always limited, our learning system evaluates an appropriate number of heuristic methods within a fixed time constraint.

Since the relationship between actions carried out by a heuristic method and the performance of the resulting mapping found is unknown without actual simulations, and there is little domain knowledge available for generating good mapping heuristics, our system must emphasize how to schedule the limited time in order to explore the maximum set of potentially good heuristic methods. Our system evaluates each heuristic method on a sequence of test cases that reflect realistic conditions of the mapping problem. We develop scheduling algorithms that trade between the number of new heuristic methods to be generated and the amount of tests to be performed on each. We analyze our model statistically for the case where each heuristic method has performance values that are normally distributed. We develop a method for obtaining a near-optimal tradeoff for the special case where the average performance values of all heuristic methods are normally distributed.

In the following section we survey the existing work on process mapping. In Section 3, we present an overview of the TEACHER 4.0 framework. Section 4 contains implementation details of TEACHER 4.1, our extension to learning heuristics for post-game analysis. The model of our statistical sequential selection strategy for evaluating heuristic methods is covered in detail in Section 5. Experimental results are presented in Section 6, and conclusions are drawn in Section 7.

2. PROCESS MAPPING PROBLEM

The problem of finding good mappings for a set of communicating processes on a network of computers has been addressed in numerous previous studies. Most studies attempt to find a mapping that provides the minimum completion time. Previous work can be classified into static and dynamic approaches. In this section we outline these studies along with their drawbacks. We then present the post-game analysis method that was designed to address many of these problems. Finally, we present some possible extensions for improving the system performance.

Static strategies are applied at design time and are usually deterministic. They can be further classified based on their analysis methods. Cost-based methods use simplistic objective functions to evaluate merit while graph-based methods use graph-theoretical models and techniques. The problems with these approaches are that their abstract models are simplified and do not result in the optimal solution and that the deterministic optimization problems they model are NP-hard.

Dynamic strategies are nondeterministic run-time strategies based on information gathered dynamically. This information is likely to be incomplete and out-of-date due to the overhead of collecting it. Load sharing, load balancing, bidding protocols, and Bayesian decision theory are examples of this approach. Results developed are usually restricted to independent jobs. This is in part

due to the difficulty in finding analytic models on the behavior of interacting processes. The relationship among the computation time of the processes, the time spent in waiting for messages from other processes (or waiting time, depending on the routing strategy), and the time spent waiting for the processor to become available (or contention time, depending on the processor scheduling strategy) are too complex to be accurately predicted and examined in advance. Consequently, dynamic strategies use simplified objectives and computation models. These models often ignore either the waiting or the contention times, and the workload on a computer is characterized simply by the number of active processes.

Post-game analysis solves some of the problems mentioned above. It tries to find the best mapping while using the minimum amount of resources. Initially, the processes are executed on the target machine using a random mapping. The method then iteratively refines the current process mappings using information collected in between program executions. Proposal-generation heuristic decision elements, which represent independent optimization subgoals, use the information collected during the execution to propose perturbations to the mapping. The resulting proposals are only partial descriptions of the necessary transformation. These proposals are then combined based on priority-assessment, transformation-generation, and feasibility heuristic decision elements to develop the actual transformation. The program is then executed (or simulated) using the new mapping. The entire process is repeated until no new changes are proposed by the heuristic method. One significant advantage of post-game analysis over earlier work on process mapping is that it does not assume any model of the processes and the computer architecture.

Figure 2.1 shows the post-game analysis system. The system is built on AXE, a simulation package that aids in collecting statistics during simulation and in experimentation with different machine architectures.

Post-game analysis is designed for *ensemble architectures* consisting of homogeneous and regularly connected processing elements (or sites), such as hypercube multicomputers and distributed systems connected by local area networks. Each processing element has its own processor, a local memory large enough to hold any number of processes, and an operating system that sends, receives, and routes all messages.

Each distributed program is represented as a collection of autonomous processes created at compile-time or at run-time. There is no shared data, and all communication must be done via message passing. This software model includes programs implementing divide-and-conquer strategies (such as quick-sort and merge-sort), blackboard-solving paradigms (such as speech recognition), and pipelining (such as image processing).

The key to improving the performance of post-game analysis lies in the development of heuristic methods that can assess the improvement achieved and the transformations to be taken next. Currently, the designer must supply heuristic decision elements for proposing and evaluating mapping decisions. In this paper, we present and evaluate an automated system for learning new heuristic methods used in post-game analysis. In addition to the possibility of generating better mappings using the heuristic methods learned, the new heuristic methods may better adapt to new architectural configurations, specific application characteristics, and different mix between communication and computation. In the following sections, we describe this population-based learning framework.

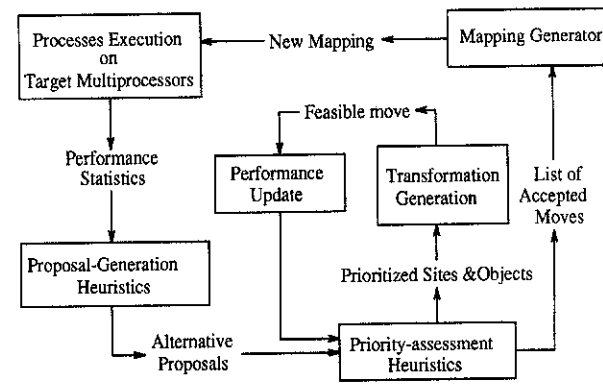

FIG. 2.1. Post-game analysis framework.

3. HEURISTICS LEARNING

Our goal in heuristics learning is to find a heuristic method that maximizes the performance of the mappings generated by post-game analysis, where a *heuristic method* in this context is a collection of *heuristic decision elements* used in post-game analysis. This problem can be expressed as a search through the space of possible heuristic methods, which can be characterized by an objective to be optimized and a set of constraints to be satisfied. For this problem, the objective is ill-defined because the tradeoff between the quality of the mapping found and the time spent to find that mapping is not well defined. In addition, time is limited, so the learning process must be terminated when time is expended.

Due to the size of the space of heuristic methods and the amount of time required for a full evaluation of a heuristic method, it is impractical to find by enumeration the best heuristic method for all possible objectives, even using high-speed computers. For example, each threshold in post-game heuristic method is a real number and can be represented by any one of the infinitely many possible values. It is also hard to have experts provide knowledge for generating good heuristic methods be-

cause parameters that affect the performance of heuristic methods might be interdependent in a way unknown even to the experts. However, the two approaches can be combined by using expert knowledge to guide the search for better heuristic methods, and by utilizing high-speed computers to perform computation-intensive experiments. This approach, called *population-based learning,* is studied in this paper.

There have been many systems developed for automated learning of new heuristic methods. In the following sections we first present issues important in developing a heuristics learning system. We then review previous works on heuristics learning, and the overall framework of our learning system.

3.1. Issues on Heuristic Learning

There are four important issues involved in heuristics learning. First, the method for performance evaluation of heuristics must be defined. This involves (a) comparing heuristic methods based on performance of these methods measured over test cases, and (b) verifying the quality of the heuristic methods selected by the learning system. One major difficulty in performance comparison is that there is no well defined relationship among the various performance attributes, such as cost and quality. Consequently, it is difficult to define what should be learned by the learning system, and how to show that one method is better than another. The latter involves normalization of performance values, an area where there are no systematic methods.

Second, techniques for generating test cases must be studied. The test cases used must be representative of the target application domain, and the performance collected based on these test cases must reflect the performance of the heuristic method on the target application.

Third, meta-knowledge for generating new heuristic methods must be developed in the learning system. The existing generation methods can be classified as model-based and model-free. Model-based heuristics-generation methods use a model relating the attributes of the heuristic method to its performance in generating new heuristic methods. On the other hand, model-free methods generate new heuristics based on predefined general knowledge and do not depend on the model of the target application. An example of such is a random perturbation of existing methods. Due to less domain knowledge involved, model-free methods are more general but may take longer to find good heuristics.

Last, the issue of scheduling the available resources in the learning system must be addressed. This involves deciding on the number of heuristic methods to test and the amount of tests performed on each. This issue is important because the number of heuristic methods is too large to be evaluated fully within any reasonable time constraint.

Solutions to the first three issues with respect to learning post-game heuristics are presented in Section 4; solutions to the last issue are described in Section 5.

3.2. Point-Based versus Population-Based Learning

The many different approaches that have been developed for learning heuristics can be classified into point-based and population-based approaches [13, 18]. Most existing learning systems are *point-based* and maintains one incumbent heuristic method that is modified in place by the learning system. Since old heuristic methods are discarded, substantial knowledge is required in justifying any change to the heuristic method. As a result, an explicit model of the problem solver that describes the relationship between the heuristic method's specification and the problem solver's performance must exist. Further, since only one incumbent heuristic method is maintained and modified, there is no need for explicit resource scheduling.

For the mapping problem considered in this paper, the relationship between the heuristic method's specification and the performance of the target problem solver is unknown. Hence, there is little domain knowledge available for guiding the generation of new heuristic methods, and existing point-based techniques are inappropriate.

The alternative is a *population-based* approach that maintains a pool of competing heuristic methods and tries to find the best heuristic method within the pool. Since multiple heuristic methods are kept concurrently, and learning is interleaved among different methods, this approach is more suitable when only model-free heuristics generation mechanisms are available, or when model-free and model-based mechanisms are used together. Further, this approach is natural to be implemented in a parallel processing environment, where several heuristic methods are evaluated concurrently. One of the key issues that need to be addressed in this approach is the scheduling of resources, which decides which heuristic methods to be evaluated next and when to generate new heuristic methods. This is in contrast to point-based methods where resource scheduling is usually implicit.

One existing example of population-based learning is the classifier system based on genetic approach [5, 6]. This maintains a pool of heuristic methods and creates new ones from existing ones that perform well in the past, while pruning inferior ones from the pool. One limitation of this approach is that it does not take advantage of the domain knowledge available in generating new heuristic methods. In addition, resource (in terms of time) is scheduled statically and does not use intermediate performance results to decide whether testing of a particular heuristic method should be continued.

To apply population-based learning while considering resource constraints, we present in the next section the design of TEACHER 4.0, a system implementing re-

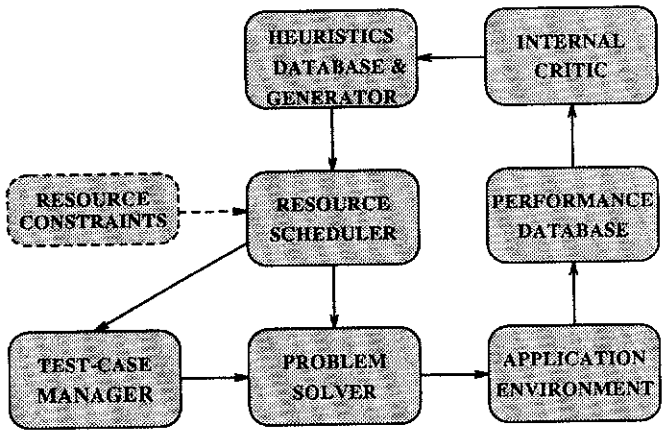

FIG. 3.1. TEACHER 4.0 framework.

source-constrained population-based and point-based learning. In Section 4, we present the design of TEACHER 4.1, a system for learning new heuristic methods used in post-game analysis.

3.3. TEACHER 4.0 Framework

The TEACHER 4.0 (or Techniques for Automated Creation of HEuRistics version 4.0) learning model [18] (see Fig. 3.1) is a general framework for implementing population-based learning systems under resource constraints in both knowledge-lean and knowledge-rich domains. It has six main components: (a) *Target Problem Solver*, which evaluates the heuristic method using a test case, (b) *Performance Database Manager*, which maintains the past performance of heuristic methods in the pool, (c) *Internal Critic*, which provides feedback based on the performance measured to indicate how well a particular heuristic method has performed, (d) *Resource Scheduler*, which decides on the best way to use the available resources, (e) *Heuristics Database Manager and Generator*, which generates new heuristic methods and maintains a pool of existing ones, and (f) *Test-Case Manager* which generates and maintains the database of test cases used in heuristics evaluation.

If domain knowledge is plentiful, then the Heuristics Generator plays an important role, and resource scheduling is less critical. In contrast, in knowledge-lean domains, the Heuristics Generator is relatively primitive, and resource scheduling that decides which heuristic methods to test and the amount of tests performed is more important. Such is the case in the process mapping problem addressed here. The focus in this paper is, therefore, placed on developing efficient resource scheduling algorithms.

To avoid spending a large amount of time on poor heuristic methods, the evaluation process is divided into small subtests called *quanta*. This allows the system to perform additional tests only on heuristic methods that demonstrate some merits during previous quanta. In each quantum, the Target Problem Solver evaluates the selected heuristic method using test cases provided by the Test-Case Manager. These test cases can be generated randomly or supplied by the users.

There are two types of test cases that can be used. First, we can choose relatively small test cases so that they can be evaluated fully within the quantum. The alternative is to use test cases dictated by the application requirements to thoroughly test the heuristic methods. The advantage of the latter approach is that the heuristic method is tested under more realistic conditions; however, its drawback is that if the evaluation of the test case is not completed at the end of the assigned quantum, it is difficult to assess which heuristic methods to test next. In the process mapping problem, performance of heuristic methods scales well (to a limited extent) between small and large test cases. Therefore, we opt to test the heuristic methods selected using small test cases and verify the performance of the final method selected after learning is completed.

At the end of a quantum, the Resource Scheduler decides on one of the three following actions: (1) select the next heuristic method to evaluate from the pool, (2) generate a new heuristic method to be placed in the pool and possibly remove an existing one from the pool, and (3) select a group of the best heuristic methods and stop learning when time is expended. The decision to take is based on the resource scheduling algorithm and the performance of existing heuristic methods in the pool.

If choice (1) or (3) is selected, then the resource scheduler will use the feedbacks provided by the Internal Critic to select the heuristic method(s) from the Heuristics Database. If choice (2) is selected, then the Heuristics Generator is used to generate a new heuristic method. In knowledge-lean application domains, mechanisms for generating new heuristic methods are rather simple. Examples include simple strategies, such as random or greedy, and genetic operators, such as crossover and mutation.

4. LEARNING HEURISTICS FOR POST-GAME ANALYSIS

In this section, we present the implementation of TEACHER 4.1 for learning new heuristic methods in post-game analysis. We describe our solutions to three of the four issues discussed in Section 3.1. First, we describe the type of post-game heuristic decision elements that we want our system to learn. Next, we discuss the techniques for evaluating the performance of heuristic methods during learning as well as after learning. We then discuss briefly the method for generating test cases in our experiments and the structure of the Heuristics Generator. We cover resource scheduling in more details in the next section.

4.1. Post-Game Heuristics

In our learning system, the post-game analysis system [20] is the *problem solver* for the target application. The problem solver can be considered as a collection of rules or procedures, whose operations are controlled by a set of parameters. A *heuristic method* is defined when the rules or procedures as well as their control parameters are specified. Recall that rules or procedures used in the problem solver are called *heuristic decision elements*.

There are three parts in the post-game analysis system that can be improved by learning: proposal generation, priority assessment, and transformation generation (see Fig. 2.1).

The proposal-generation component consists of a collection of heuristic decision elements used for generating proposals based on independent optimization subgoals. Each *proposal-generation heuristic decision element* can be represented as a record composing of four fields: (a) the reason field that specifies the motivation behind the decision element; (b) the quantifier field that specifies the scope of the decision element; (c) the condition field that specifies the condition in which proposals are generated; and (d) the action list that specifies the proposals to be generated. Each action is a record containing the action type, parameters for the proposal, and an expression specifying the weight of the proposal. In generating a new proposal, each heuristic decision element is applied according to its quantifier field. The condition expression of each decision element is then evaluated for each subject specified by the quantifier. When the evaluation of an expression returns a nonzero value, each action in the action list is executed to generate a proposal and its corresponding weight.

There are two *priority-assessment heuristic decision elements*, each of which can be represented by an expression. One of these specifies the priority of each process, and the other specifies the priority of each site. The two expressions are used for comparing two processes or two sites in order to determine their relative importance. The partial orders obtained are used to determine the order in which moves are considered.

The transformation-generation component determines the transformation that will be applied to perturb the current mapping of the processes. It consists of two types of heuristics: the *transformation-generation heuristic decision element* that generates possible transformations from the ordered lists of sites and processes, and the *feasibility heuristic decision elements* for checking the feasibility of a move generated by the transformation-generation heuristics. For the transformation-generation heuristic decision elements, there are currently two types of moves that can be generated: *migration* [20] that migrates the top-order process to the top-order site, and *swap* [21] that swaps the locations of the two top-order processes. For the feasibility heuristic decision elements, there are two different sets, each of which can be represented as a collection of expressions stating the condition in which the transformation would be infeasible. For each transformation considered by post-game analysis, an expression in the collection corresponding to the type of transformation is evaluated. If any expression returns a nonzero value, then the corresponding transformation is rejected.

To apply learning, we have modified the post-game analysis system so that all the heuristic decision elements are represented as data structures readable in the form a data file.

4.2. Internal Critic

The objective of the Internal Critic is to evaluate how well the problem solver has performed using the given heuristic method. For our target application, the performance of post-game heuristic method h on test case v consists of two measurements: $c(h, v)$, the *cost* or the amount of time needed to find the mapping, and $q(h, v)$, the *quality* of the mapping found, where $q(h, v)$ is the reciprocal of $m(h, v)$, the completion time of the given set of processes mapped.

There are two problems with using the quality and cost measures to evaluate the performance of heuristic methods. First, the values of these measures are test-case dependent and can vary widely. This makes it difficult to compare the performance of heuristic methods that are evaluated on different test cases. Hence, these values need to be *normalized* so that they are consistent across the entire range of test cases. In this case, since the goal is to find a heuristic method that performs better than the one used in the original system, we can use the performance of the original heuristic method [21] as the basis for normalization. $c_n(h, v)$, the normalized mapping cost, and $m_n(h, v)$, the normalized completion time of the process mapped, can be computed using the equations

$$c_n(h, v) = \frac{c(h, v)}{c(h, v) + c_{orig}(v)}, \tag{4.1a}$$

$$m_n(h, v) = \frac{m(h, v)}{m(h, v) + m_{orig}(v)}, \tag{4.1b}$$

where $m_{orig}(v)$ and $c_{orig}(v)$ are, respectively, the completion time and the cost of the original post-game heuristic method on test case v. Note that the normalized values are always between 0 and 1, and that the original post-game heuristic method has a normalized cost and completion time of 0.5. The normalized quality of the mapping, $q_n(h, v)$, is defined as the reciprocal of $m_n(h, v)$.

$$q_n(h, v) = \frac{m(h, v) + m_{orig}(v)}{m(h, v)}. \tag{4.1c}$$

The second problem is that the proper tradeoff between quality and cost measures is ill defined: the quality of the mapping generally improves as more time is spent on finding the mapping. However, only the following unrealistic boundary cases are known: (a) unlimited time for the heuristic method that finds the best mapping, (b) zero time for the heuristic method and hence zero cost. In this paper, we assume that the tradeoff is defined by a family of heuristic functions of quality and cost, which are used by the Internal Critic to distinguish between heuristic methods that perform well from those that do not. A family of functions are used because a single function is inadequate in capturing all the different levels of tradeoffs in different applications. We assume that the family of functions is parameterized by T_{scale} and are represented as follows:

$$Q(T_{scale}) = \max_{h} \sum_{v} \frac{q_n(h, v)}{0.5 + (c_n(h, v) - 0.5)/T_{scale}}. \quad (4.2)$$

The parameter T_{scale} controls the relative weight between cost and quality by adjusting the range of the normalized cost while maintaining the cost of the original heuristic method at 0.5. The effect of Eq. (4.2) is to scale $c_n(h, v)$ so that it is possible to investigate alternatives where cost is dominant, or where performance is dominant, or both. Note that Eq. (4.2) is not unique, and our choice is heuristic in nature.

Unfortunately, the performance of each heuristic method can vary widely even when evaluated on the same test case but with different initial mappings. Therefore, each test case must be evaluated over a number of initial random mappings until a given confidence level has been reached. In practice, there is usually a limit on the number (say 30) of initial mappings that can be used for evaluating a test case. For each initial mapping, the performance is computed using Eqs. (4.1) and (4.2). The evaluative feedback of a heuristic method on a test case is then the average of its performance over the set of initial mappings.

4.3. Performance Evaluation of Heuristics

In this section, we describe methods for evaluating the performance of the heuristic method selected by the learning system. This involves determining the statistical distribution of performance of the heuristic method selected on test cases similar to those used in the learning system, and finding the scalability and generality of the heuristic method on larger test cases and test cases drawn from different application problems.

In order to compare the performance of one heuristic method to that of another, it is not enough to use the average performance values, as the average performance conveys no information about the distribution of performance values. Further, post-learning evaluation of performance should not be compared using $Q(T_{scale})$ defined in Eq. (4.2) because $Q(T_{scale})$ is heuristic and may not capture the proper tradeoff between cost and quality. To show simultaneously the cost and quality of different heuristic methods, we need to show the quality of the heuristic method for each test case against its cost for the same test case. This allows the relationship between cost and quality to be depicted without predefining a heuristic performance function. Such plots are shown in Section 6.3.

A problem in showing the relationship between cost and quality of different heuristic methods evaluated on different sets of test cases is that their values may not be in the same range. Hence, all quality and cost values must be normalized. To give consistent basis for comparing performance values, we choose to normalize them against the average quality and cost of the original post-game heuristic methods. That is, the normalized quality and cost are

$$\hat{c}_n(h, v) = \frac{c(h, v)}{E[c_{orig}(v)]} \quad (4.3a)$$

$$\hat{m}_n(h, v) = \frac{m(h, v)}{E[m_{orig}(v)]} \quad (4.3b)$$

$$\hat{q}_n(h, v) = \frac{E[m_{orig}(v)]}{m(h, v)}, \quad (4.3c)$$

where $E[m_{orig}(v)]$ and $E[c_{orig}(v)]$ are the average completion time and the cost of the original post-game heuristic method on test case v. Note that this normalization is meaningful only if the expected value is found for test cases belonging to a set with a common distribution of performance values.

4.4. Test-Case Generation

The Test-Case Manager of TEACHER 4.1 is used to generate a set of test cases that are representative of the problem domain and the hardware environment and that can be evaluated within a reasonable amount of time during learning.

A *test case* in this case is represented by the specification of the set of communicating processes to be mapped, the hardware on which the processes are mapped, and the input values to the processes. An example of a test case is a merge-sort program on a 16-node hyercube machine evaluated with a data set of size 300. The same program running on different machine architectures or with different sizes of array to be sorted is considered to be a different test case. Programs using different algorithms, such as merge-sort and quick-sort, are considered different test cases as well.

In the analysis carried out in this paper, we assume

that the quality and cost values of test cases are drawn from a common joint distribution. In reality, the cost and quality values of a test case are application dependent and cannot be controlled. In our learning experiments, we tried to select test cases so that their quality and cost values satisfied the assumptions made in our analysis. In particular, we chose test cases that had a constant number of processes to be mapped on a given architectural configuration, processes that had the same communication pattern, and execution times of each code segment drawn from a common distribution.

Moreover, we assume that small test cases are used, and the verification of scalability of the heuristic method learned is carried out in a post-learning phase. Of course, the attribute we use to distinguish test cases in one set from those of another set is rather ad hoc. We plan to study in the future the problems of using test cases of different types and the normalization of performance results of these test cases during learning.

4.5. Generation of Post-Game Heuristics

The generation of new post-game heuristics is difficult. First, a heuristic method has both numeric and symbolic parts, and the symbolic part is more difficult to modify automatically. Further, it requires substantial amount of domain knowledge in order to invent new heuristic decision elements that are missing in the current heuristic method. In this paper, we only consider automated generation methods that start with an initial pool of heuristic methods and transform them into new ones by a few well defined operators.

To develop the Heuristics Generator, we must first identify its desirable characteristics. The learning process can be modeled as a search within the heuristics space; the operators used by the Heuristics Generator control the direction in which search is conducted. Ideally, the area containing heuristic methods with the highest performance should first be located. This is difficult for knowledge-lean application domains. Hence, the Heuristics Generator must explore different regions of the heuristics space simultaneously and focus on promising regions based on evaluative feedbacks obtained during learning.

Initially, when learning starts, there is no performance results on any heuristic methods. Without domain knowledge, we assume that a random sequence of mutation, insertion, and deletion is applied initially on the original post-game heuristic methods in order to generate new ones. Subsequently, there are four types of operators for transforming a heuristic method into another: cross-over, mutation, insertion, and deletion. Cross-over is a global operator that combines features from two existing heuristic methods and splits them into two new methods. Mutation is the most commonly used operator that modifies the parameter value of an existing heuristic decision element. For our application, mutation can be applied to (a) adjust the threshold of the proposal-generation heuristic decision elements, (b) change the sites and processes priority-assessment heuristic decision elements by selecting from one of six possible combinations, and (c) change the transformation-generation heuristic decision element by selecting from one of six possible choices. The modification may be carried out in the direction that shows the maximum improvement in performance in the past. Note that both cross-over and mutation are commonly used in genetic algorithms and are used more frequently when past performance history is available. Finally, an insertion operator adds a new proposal-generation heuristic decision element to the existing heuristic method, while a deletion operator removes one of the existing proposal-generation heuristic decision element.

A special case of the mutation operator is the greedy operator, which in our current prototype has three phases. In the first phase, the generator locates an existing heuristic method that performs the best in a region of heuristics space. Next, the generator tries to find the sequence of transformations that cause the greatest improvement in performance within that region. Last, a new heuristic method is generated from the heuristic method selected using the sequence of operators found in the second phase. Our current implementation is rule based, so additional operators and rules can be added easily.

In our prototype, we set the probability of random generation based on the number of existing heuristic methods. It is reduced linearly from 1.00 when no existing heuristic method is in the pool to 0.10 when the number of heuristic methods reaches 65. The remaining generation operators are split between the cross-over and the greedy mutation operators, with mutation selected 80 percent of the time.

5. STATISTICAL CANDIDATE SELECTION UNDER TIME CONSTRAINTS

In this section, we describe and analyze the resource scheduling strategies for our learning framework. The first part of this section outlines the problem, previous work in the area, and difficulties found in applying previous results to our problem. Next, we describe our proposed scheduling strategies that address these difficulties. This is followed by the evaluation of various selection strategies, and their application to the learning of post-game heuristic methods.

5.1. Problem Overview

In TEACHER, we are faced with choosing the best heuristic method from a pool of heuristic methods, each of which is associated with a set of performance values. This problem can be restated in statistical term as fol-

lows: given a set of populations (or candidates) consisting of normally distributed numbers (with unknown means and variances), the goal is to select the one with the highest population mean by testing a certain number of samples from these populations.[1] In our case, the populations are post-game heuristic methods, and the numbers comprising the elements of the populations are the performance values associated with applying the heuristic method to the given test cases.[2] Making one pick from a population is analogous to testing the heuristic method on one test case. The goal is to chose the population with the highest mean within a given amount of tests (or picks).

In the past, this problem of finding the population with the best mean has been known as the *selection problem*. Previous work in this area can be classified into two types. The first type is known as the *stopping problem* and was first studied by Stern in 1948 [14]. Its objective is to develop a strategy for selecting populations for testing so that the amount of tests required to reach certain stopping criteria is minimized. The second type is known as the *allocation problem* and was pioneered by Bechhofer in 1954 [1]. The original problem suggested in his paper is to decide the optimal allocation of picks given a fixed total number of picks, assuming population means and variances are known. Optimal solutions to both types of problems are unknown. Many extensions have been proposed to accommodate various tradeoffs and relaxed assumptions.

The major problem in applying traditional statistical methods to the heuristics selection process is that the emphasis has been on testing a *finite* set of populations, rather than on finding the *best* population from a large set in a given *time constraint*. The case in which there are possibly more populations than total allowed testing time has not been addressed before. This condition is highly relevant in our case because we know our deadlines and often have far more options than we could possibly explore.

To address the selection problem in heuristics learning, the time allowed can be broken down into two separate parts: the first part finds the number of populations that are to be tested in more detail in the second part. Finding a suitable number of populations to test in the first part is studied in the next subsection. Once the populations to be tested and the time allowed for the second part are determined, scheduling the tests of populations in the second part is equivalent to the allocation problem studied in statistics. Note that the objective of the statistical allocation problem is to maximize $P(CS)$, the probability of correctly selecting the population with the highest population mean when time is expended, whereas the objective of our selection problem is to maximize the average population mean value of the population selected. Maximizing $P(CS)$ is not meaningful in our case because the number of possible populations are too large to be evaluated in the time limit. Due to this difference in objectives, most of the analysis used in statistical allocation problems do not apply to our problem.

Existing allocation strategies can be further classified into *static* and *dynamic*. Static allocation strategies have a selection sequence fixed ahead of time, independent of the values of the picks obtained during selection. One simple strategy of this type is the *round-robin* strategy that takes samples from each population in turn. It is a good strategy for maximizing the worst-case $P(CS)$ when all populations have the same variance [2]. Its drawback is that the number of picks made from each population is the same, meaning that the worst population is tested to the same extent as the best, an obviously inefficient way of using resources. This problem is inherent in all static allocation strategies.

In contrast to the static approach, dynamic (or adaptive) allocation strategies select the population for testing based on previous sample values. An example of this is the *greedy* policy which takes samples from the population that currently has the maximum sample mean. The potential problem with this approach is that it considers only populations that look good in the current stage, and might discard the best population in an early stage. Another example is the strategy proposed by Tong and Wetzell [16], which dynamically (or adaptively) selects the next population for testing based on estimated means and variances of populations in order to optimize $P(CS)$ when time runs out.

5.2. Resource Scheduling Strategy

Based on the problems described in the previous section, the scheduling strategies that we develop must take limited resources like testing time into consideration, and must be able to deal with situations in which there are more populations so that it would be impossible to test all populations even once. Under these conditions, the scheduling strategies must trade carefully between the number of populations to be tested and the accuracy of the sample-mean values. If more populations are tested, then the accuracy of the sample-mean values would decrease. However, if more samples are tested from a population, then less number of populations are tested in the time constraint, and there is a greater chance of missing a good population.

There are two parts to this problem. First, the desirable level of tradeoff depends on factors such as the distribution of the populations, the distribution of the perfor-

[1] Population mean and variance are properties of a population. They can be estimated by the sample mean and variance if limited samples are drawn from the population.

[2] A test case is provided by the Test-Case Manager. See Section 4.3 for details.

mance values of each population, and the distribution of the testing time (or sampling overhead) required for each test. In the heuristics learning system, these various distributions are unknown and need to be estimated before learning begins. Second, the scheduling strategies developed must select the most appropriate level of tradeoff so that we can expect to find a population with a large mean value when time is expended. The design of such strategies is difficult, and optimal strategies are not expected to exist.

In this section, we describe the solutions developed for these two problems. We first describe the method for estimating the various distributions of the populations before learning. We then present the multistage selection strategy. Analysis of the performance of several multistage strategies is presented in Section 5.3.

5.2.1. *Presampling Strategy*

To estimate the statistical parameters of the target application, we assume that the initial 10% of T, the allotted learning time, is assigned to presample the populations. During this period, the learning system will evaluate as many heuristic methods as possible and collect performance data in terms of sample means and sample standard deviations. Each heuristic method is evaluated on four different test cases to assure a reasonable confidence in its statistical values.

At the end of presampling, the learning system collects the performance data and computes (a) an estimated average (μ_0) and variance (σ_0^2) of all the sample means of candidates, (b) an estimated variance σ^2 of each candidate, assuming all candidates have the same variance, and (c) the average sampling overhead, $\bar{c} = T/(10k)$, where k is the total number of tests performed during the presampling period. The average number of tests that can be performed in the entire duration is estimated to be $\bar{T} = T/\bar{c} = 10k$. The resource scheduler then uses these estimates to choose the appropriate selection strategy along with appropriate parameters values.

5.2.2. *Multistage Testing Procedure*

We formulate the general selection strategy $G(T)$ as a series of stages, $G_i(g_i, t_i, n_i)$, where i ranges from 1 to m, the number of stages. Each stage is characterized by a triplet consisting of (a) g_i, the particular allocation strategy to be used for the 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 [9]. This multistage testing procedure (see Fig. 5.1) can accommodate both static and dynamic allocation strategies. With this method, the first stage corresponds to coarse initial testing to weed out unworthy populations, followed by a more careful evaluation of the better populations. Only populations that have the top n_{i+1} sample mean values at the end of stage i are taken into stage $i + 1$ for further testing. Note that in the last stage, only the top population needs to be selected. This method allows many new populations to be tested in the early stages, while providing more thorough testing of promising populations in later stages.

The performance of multistage testing depends on the number of stages and the parameter values for each stage. The values of these parameters must be selected properly based on the given problem. Factors that affect these values include the size and distribution of each population, the total amount of testing time, and the number of possible populations and their distributions. To develop a procedure for determining the appropriate parameter values based on the given distributions, we need to know their relationship with the performance of the multistage procedure. This is presented in the next section.

5.3. Performance Evaluation of Multistage Selection Procedure

In this section, we evaluate the performance of several multistage selection strategies. We focus on only single-stage and two-stage strategies because (a) the amount of improvement in going from two-stage strategies to three-

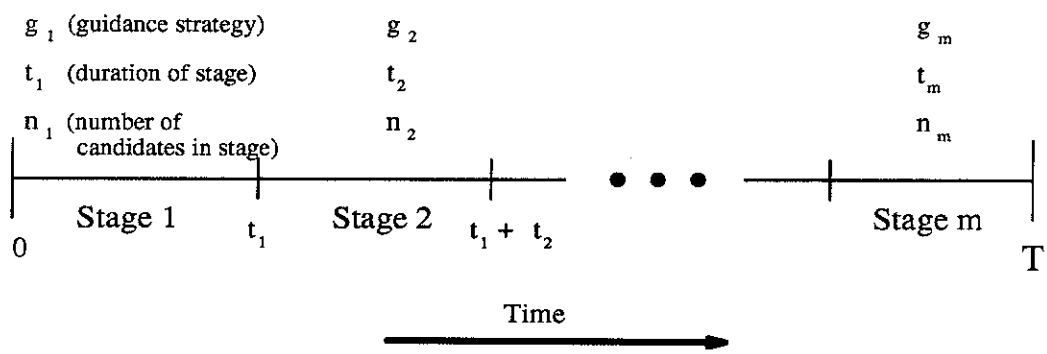

FIG. 5.1. Multistage testing procedure.

stage ones is less than that of going from one-stage to two-stage ones, and more importantly (b) the number of parameters that must be selected increases with the number of stages, and there is no analytic method for determining the best values for these parameters.

Monte Carlo simulations and mathematical analysis are two methods for evaluating performance. Although mathematical results are more desirable, they may be too complex to be useful in our problem or may be impossible to derive. We compromise by deriving analytic results for simplified cases and by verifying the validity of the analytic results through Monte Carlo simulations of the actual learning problem. We first present the assumptions used in our analysis. We then present general steps used in analyzing the multistage strategies along with an upper bound on their performance. These steps are then applied to the special case in which the sampling overhead is unity for all populations. We analyze the control of static strategies and present simulation results on dynamic strategies. The analysis of two-stage static strategies (round-robin in both stages) is used as heuristics to determine the appropriate parameter values when a dynamic strategy is used in the second stage. Last, we present the results for the case with stochastic sampling overheads.

5.3.1. *Assumptions*

There are three assumptions we made on the distribution of a population in the analysis of the multistage procedure.

First, we assume that each population is normally distributed, and that sample values in a population are independent and identically distributed (i.i.d.). Empirical verification of this assumption is shown in Table 6.1.

Second, we assume that the mean of each population is drawn from a known distribution f_μ, and that the values of all population means are i.i.d. That is, a randomly selected population has a population mean that is normally distributed, and is independent of means of other populations. In many cases f_μ can be approximated as normal. We further assume that each population has an infinite number of samples. This assumption can be verified for reasonably large training sets and deadlines.

Last, we assume that the standard deviation of each population is drawn from a known distribution f_σ, and that the values of all population standard deviations are i.i.d. Further, we assume that the mean and standard deviation of each population are independent.

The assumptions specified here are somewhat more restrictive than the ones used in studying traditional selection problem [7]. Traditional analysis of selection strategies do not assume any distribution of the population-mean values because (a) it deals with finite number of populations and does not need to determine the most appropriate number of populations to test in the time allowed, and (b) its goal is to optimize $P(CS)$ instead of maximizing the average performance of the population selected.

5.3.2. *Analysis of General Multistage-Selection Method*

We present in this section detailed analysis of multistage selection strategies, assuming that the sampling overhead is unity. We discuss the general steps for analyzing stage by stage of such strategies. We then present an upper bound on the performance.

Let T be the duration of the current stage, n be the number of populations to evaluate in this stage, and k be the number of populations that will be selected for further evaluation in the following stages, if any.

The key to the analysis depends on finding the joint probability distribution (f_{joint}) between the sample mean, the population mean, the population standard deviation, the number of samples drawn from each population, and any other variables used in selecting the next heuristic method for further testing. For static strategies, f_{joint} depends only on $\bar{x}$, the sample mean, μ, the population mean, σ, the population standard deviation, and s, the number of samples; that is, $f_{joint} = f_{\bar{x},\mu,\sigma,s}$. Note that s is a strategy-dependent random variable and may be different for different populations. For dynamic strategies, since the candidates are selected based on some parameters of the joint distribution, each pick must be treated as an individual stage, and the joint distribution must be updated after each pick. In this case, the analysis is generally complex because selection is based on the dynamic ordering of candidates in the pool, which depends on the sequence of past decisions made. To analyze dynamic strategies, the joint distribution of the performance values of all populations based on their performance ranking must be maintained. The steps required for analyzing dynamic allocation strategies are too complex and strategy dependent and will not be discussed here.

For static allocation strategies, the selection sequence is fixed ahead of time and all populations are treated in the same way. Therefore, only the joint probability distribution of a random population is needed. In this case, there are four general steps in the analysis of each stage using information available from the analysis of previous stages.

(1) Determine $f_{\bar{x},\mu,\sigma,s}$, the joint probability distribution (p.d.f.) for a random population at the end of the current stage. For the first stage, $f_{\bar{x},\mu,\sigma,s}$ can be found based on the allocation strategy, f_μ, the given distribution of population means, and f_σ, the given distribution of population standard deviations. For any other stage, $f_{\bar{x},\mu,\sigma,s}$ can be found based on the allocation strategy used in the stage and $f_{\bar{x},\mu,\sigma,s,(k:n)}(x, \mu, \sigma, s)$ from step (4) of the previous

stage. This is the only step in which the chosen allocation strategy may affect the performance.

(2) Determine $F_{\bar{x}}$ and $f_{\bar{x}}$, the cumulative distribution function (c.d.f.) and p.d.f. of the sample means of the candidates evaluated at the end of the current stage. This is the marginal distribution of $\bar{x}$ and is derived based on $f_{\bar{x},\mu,\sigma,s}(x, \mu, \sigma, s)$ from step (1):

$$f_{\bar{x}}(x) = \sum_s \int_{\sigma=-\infty}^{+\infty} \int_{y=-\infty}^{+\infty} f_{\bar{x},\mu,\sigma,s}(x, y, \sigma, s)\, dy\, d\sigma. \quad (5.1)$$

(3) Determine $f_{\bar{x},(k:n)}$, the p.d.f. of the populations that have one of the top k among n sample means at the end of the current stage. The subscript notation $(k:n)$ means the top k out of n populations.

At the end of each stage, we select the k populations with the highest sample means to be evaluated further in the following stage, if any. $f_{\bar{x},(k:n)}$, the p.d.f. of a random population from among the k selected populations, is simply the average of the top k order statistics [8] of the sample means ($f_{\bar{x}}(x)$):

$$f_{\bar{x},(k:n)}(x) = \frac{f_{\bar{x}}(x)}{k} \sum_{i=n-k+1}^{n} i \binom{n}{i} [F_{\bar{x}}(x)]^{i-1}[1 - F_{\bar{x}}(x)]^{n-i}. \quad (5.2)$$

(4) Determine $f_{\bar{x},\mu,\sigma,s,(k:n)}$, the joint p.d.f. for the k populations with the top sample means among n original populations. This is the only result that must be carried over to the next stage for static strategies because, in this case, the dependency among populations at the end of the current stage can be ignored. Using Bayes' theorem, we derive the equation

$$\begin{aligned} &f_{\bar{x},\mu,\sigma,s,(k:n)}(x, \mu, \sigma, s) \\ &\quad = P[x, \mu, \sigma, s \mid x \text{ is in the top } k] \\ &\quad = \frac{P[x \text{ is in top } k \mid x, \mu, \sigma, s]\, f_{\bar{x},\mu,\sigma,s}(x, \mu, \sigma, s)}{P[x \text{ is in top } k]} \\ &\quad = \frac{f_{\bar{x},(k:n)}(x)\, f_{\bar{x},\mu,\sigma,s}(x, \mu, \sigma, s)}{f_{\bar{x}}(x)}. \end{aligned} \quad (5.3)$$

For the last stage, $k = 1$. Equation (5.2) is reduced to

$$f_{\bar{x},(1:n)}(x) = n\, f_{\bar{x}}(x)[F_{\bar{x}}(x)]^{n-1}. \quad (5.4)$$

In the last stage, it is also necessary to find $f_{\mu(selected)}$, the distribution of the population mean of the candidate with the highest sample mean. This can be found from the marginal distribution of $f_{\bar{x},\mu,\sigma,s,(1:n)}(x, \mu, \sigma, s)$ from the last stage. By using Eqs. (5.3) and (5.4), we obtain

$$\begin{aligned} &f_{\mu(selected)}(\mu) \\ &\quad = \sum_s \int_{x=-\infty}^{+\infty} \int_{\sigma=-\infty}^{+\infty} f_{\bar{x},\mu,\sigma,s,(1:n)}(x, \mu, \sigma, s)\, d\sigma\, dx \\ &\quad = \sum_s \int_{x=-\infty}^{+\infty} \int_{\sigma=-\infty}^{+\infty} n[F_{\bar{x}}(x)]^{n-1} f_{\bar{x},\mu,\sigma,s}(x, \mu, \sigma, s)\, d\sigma\, dx. \end{aligned} \quad (5.5)$$

The expected value of the population mean of the candidate selected is

$$E[\mu_{(selected)}] = \int_{y=-\infty}^{+\infty} y f_{\mu(selected)}(y)\, dy. \quad (5.6)$$

For comparison, the performance of any multistage selection strategy is bounded by that of a hypothetical (but unrealistic) strategy which is able to determine the candidate with the highest expected performance after sampling each candidate only once. If the number of tests that can be performed in the given time limit is T, then the maximum number of heuristic methods that can be sampled by the hypothetical strategy is also T. The distribution of the sample mean for the best candidate among T populations is the Tth order statistic of the population-mean distribution. Let $F_\mu(x)$ and $f_\mu(x)$ be the c.d.f. and p.d.f. of the population mean of a random heuristic method, then the p.d.f. of the best of T random heuristic methods is [8]

$$f_{best}(x) = T[F_\mu(x)]^{T-1} f_\mu(x). \quad (5.7)$$

This upper bound is somewhat loose because it assumes that we would be able to determine the candidate with the best population mean every time with one sample from each population. Moreover, it does not take f_σ, the distribution of the population standard deviation, into consideration.

5.3.3. *Static Allocation Strategies*

In static allocation strategies, the selection sequence is fixed and is independent of the values of the picks made during the selection process. Since the sequence is fixed, it is likely that testing time be wasted on some populations that have already shown poor performance from the few initial tests. This could be partly overcome by using a multistage selection process which can prune off inferior populations early in the process.

Static strategies do have one point in their favor: they are much easier to analyze than dynamic ones. In addition, they can be used in the first stage of a multistage process employing dynamic allocation strategies in order to obtain initial statistics. In this section, we analyze the

performance of the *single-stage round-robin* and the *two-stage round-robin* strategies. We also provide a method for determining the parameter values for these strategies.

The *single-stage round-robin* strategy, $G(RR, T, n)$, takes samples (or tests) from each population (or candidate) in turn, where n is the number of populations to be evaluated, and T is the deadline. Given n and T, s, the number of tests for each population, is constant and is equal to T/n when the sampling overhead is unity. Assuming that each population has normal distribution with variance σ^2, population mean μ, and s sample drawn, the sample-mean values have a normal distribution with mean μ and variance σ^2/s. Assuming that the distribution of all population means is f_μ and the distribution of all population standard deviation is f_σ, we can determine the joint distribution of the population mean, population standard deviation, and sample mean when the deadline is reached:

$$f_{\bar{x},\mu,\sigma,s}(x, \mu, \sigma, s) = \begin{cases} f_{\bar{x}|\mu,\sigma,s}(x \mid \mu, \sigma, s) f_\mu(\mu) f_\sigma(\sigma) & \text{for } s = \dfrac{T}{n} \\ 0 & \text{otherwise} \end{cases}$$

$$= \begin{cases} \dfrac{f_\mu(\mu) f_\sigma(\sigma)}{\sigma\sqrt{2\pi n/T}} e^{-(x-\mu)^2/(2\sigma^2 n/T)} & \text{for } s = \dfrac{T}{n} \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

From this equation, we follow the general steps described in the previous section to arrive at the distribution of the sample mean at the end of the stage (Eq. (5.1)) and the distribution of the population mean of the candidate selected when time runs out (Eqs. (5.5) and (5.6)):

$$f_{\bar{x}}(x) = \int_{\sigma=-\infty}^{+\infty} \int_{y=-\infty}^{+\infty} f_{\bar{x}|\mu,\sigma,s}\left(x \,\middle|\, y, \sigma, \frac{T}{n}\right) f_\mu(y) f_\sigma(\sigma)\, dy\, d\sigma. \tag{5.9}$$

$$f_{\mu(selected)}(\mu) = n f_\mu(\mu) \left\{ \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f_{\bar{x}|\mu,\sigma,s}\left(x \,\middle|\, \mu, \sigma, \frac{T}{n}\right) f_\sigma(\sigma) \right. \tag{5.10}$$

$$\left. \times \left[\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} F_{\bar{x}|\mu,\sigma,s}\left(x \,\middle|\, \mu, \sigma, \frac{T}{n}\right) f_\mu(\mu) f_\sigma(\sigma)\, d\mu\, d\sigma \right]^{n-1} d\sigma\, dx \right\}$$

$$E[\mu_{(selected)}] = \int_{x=-\infty}^{+\infty} f_{\bar{x},(1:n)}(x) \int_{\mu=-\infty}^{+\infty} \int_{\sigma=-\infty}^{+\infty} \frac{\mu f_{\bar{x},\mu,\sigma,s}(x, \mu, \sigma, (T/n))}{f_{\bar{x}}(x)}\, d\sigma\, d\mu\, dx$$

$$= \int_{x=-\infty}^{+\infty} f_{\bar{x},(1:n)}(x)\, E[\mu | \bar{x} = x]\, dx. \tag{5.11}$$

For the special case in which f_μ is $N(\mu_0, \sigma_0^2)$ and all population standard deviations are constants equal to σ, we derive a more specific result using Eqs. (5.4), (5.8), and (5.11),

$$f_{\bar{x},(1:n)}(x) = \frac{n}{\sqrt{2\pi(\sigma^2 n/T + \sigma_0^2)}} e^{-((x-\mu_0)^2/2(\sigma^2 n/T + \sigma_0^2))} \times \Phi\left(\frac{x - \mu_0}{\sqrt{\sigma^2 n/T + \sigma_0^2}}\right)^{n-1} \tag{5.12}$$

$$E[\mu_{(selected)}] = \int_{x=-\infty}^{+\infty} \frac{x\sigma_0^2 + \mu_0 \sigma^2 n/T}{\sigma^2 n/T + \sigma_0^2} f_{\bar{x},(1:n)}(x)\, dx$$

$$= \mu_0 + \int_{y=-\infty}^{+\infty} \frac{\sigma_0^2 y n e^{-y^2/2} [\Phi(y)]^{n-1}}{\sqrt{2\pi(\sigma^2 n/T + \sigma_0^2)}}\, dy, \tag{5.13}$$

where $\Phi(x)$ is the c.d.f. for an $N(0, 1)$ distribution.

From the equations above, it is clear that in this case only the ratio between σ^2 and T affects the performance. As T increases or σ^2 decreases, the average performance of the population selected by this strategy improves. This happens because the accuracy of the sample mean as a predictor of the population mean increases as more time is given or if the variances are reduced.

In the simple one-stage round-robin strategy, the only parameter that can be adjusted is n, the number of populations to be evaluated. One method for determining the optimal n, $1 \le n \le T$, that leads to the optimal $E[\mu_{(selected)}]$ for a given problem and deadline is to find n that makes $dE[\mu_{(selected)}]/dn = 0$. In most cases, however, this cannot be solved easily because it is difficult to get a closed-form solution for the integration.

If there is only one n that makes the derivative goes to 0, then there is a simple method for finding the optimal value of n for the given problem. In this case, if $E[\mu_{(selected)}]$ at n_1 is larger than that at n_2, and $n_1 < n_2$,

then $n_{opt} < n_2$; if $n_1 > n_2$, then $n_{opt} > n_2$. Figure 5.2a shows that this condition is usually met. Under this condition, the optimal n can be found by a binary search.

For more complex multistage strategies using the round-robin strategy in the first stage, we need to derive the joint distribution of the population means and sample means of the top k among n populations at the end of the first stage using Bayes' theorem:

$$f_{\bar{x},\mu,\sigma,s,(k:n)}(x_i, \mu_i, \sigma_i, s_i | n-k+1 \le i \le n) = f(\mu_i, \sigma_i, s_i | x_i, n-k+1 \le i \le n) \times f_{\bar{x}}(x_i | n-k+1 \le i \le n)$$

$$= f_{\bar{x}}(x_i | n-k+1 \le i \le n) \prod_{i=n-k+1}^{n} f(\mu_i, \sigma_i, s_i | x_i). \quad (5.14)$$

The above equation is equal to 0 unless $x_i < x_{i+1}$ for all i, $n-k+1 \le i \le n$. Using order statistic [8], we can derive $f_{\bar{x}}(x_i | n-k+1 \le i \le n)$:

$$f_{\bar{x}}(x_i | n-k+1 \le i \le n) = \frac{n!}{(n-k)!} [F_{\bar{x}}(x_{n-k+1})]^{n-k} \prod_{i=n-k+1}^{n} f_{\bar{x}}(x_i). \quad (5.15)$$

We can combine Eqs. (5.14), (5.15), and (5.8) to simplify the result as follows:

$$f_{\bar{x},\mu,\sigma,s,(k:n)}(x_i, \mu_i, \sigma_i, s_i \mid n-k+1 \le i \le n) = \begin{cases} \dfrac{n!}{(n-k)!} [F_{\bar{x}}(x_{n-k+1})]^{n-k} \displaystyle\prod_{i=n-k+1}^{n} f_{\bar{x}|\mu,\sigma,s}\left(x_i \middle| \mu_i, \sigma_i, \frac{T}{n}\right) f_\mu(\mu_i) f_\sigma(\sigma_i) \\ \quad \text{where } s_i = T/n \text{ for all } i,\ (n-k+1) \le i \le n, \text{ and } x_{n-k+1} < x_{n-k+2} < \cdots < x_n \\ 0 \quad \text{otherwise.} \end{cases} \quad (5.16)$$

If the next stage is a static strategy where the selection of candidates is independent of the past performance of the candidates selected, then Eq. (5.16) can be *approximated* using just the average of the joint distribution for the top k candidates at the end of the stage. This can be derived by substituting Eqs. (5.2) and (5.8) in Eq. (5.3):

$$f_{\bar{x},\mu,\sigma,s,(k:n)}(x, \mu, \sigma, s) = \begin{cases} \dfrac{1}{k} \displaystyle\sum_{i=n-k+1}^{n} i \binom{n}{i} [F_{\bar{x}}(x)]^{i-1} [1 - F_{\bar{x}}(x)]^{n-i} f_{\bar{x}|\mu,\sigma,s}\left(x \middle| \mu, \sigma, \frac{T}{n}\right) f_\mu(\mu) f_\sigma(\sigma) & s = \dfrac{T}{n} \\ 0 & \text{otherwise} \end{cases} \quad (5.17)$$

The *two-stage round-robin* strategy, $\{G_1(RR, rT, n_1), G_2(RR, (1-r)T, n_2)\}$, is built on the single-stage round-robin strategy. In this strategy, a fraction r of the total time T is used in the first stage to evaluate n_1 populations. At the end of the first stage, n_2 populations ($n_2 < n_1$) with the largest sample means are taken into the second stage for further evaluation. The second stage takes all the remaining time and also uses a round-robin strategy.

For each stage, the number of tests performed on each population is the same with $s_1 = rT/n_1$, and $s_2 = ((1-r)T)/n_2$, assuming unit sampling overhead. The sample-mean value of a population at the end of the second stage is dependent on (a) u, the sample-mean value of this population at the end of the first stage after s_1 tests, and (b) x, the sample-mean value found for this population at the end of the second stage after $s_1 + s_2$ tests.

Given u and x for a population, we can determine v, the sample-mean value of this population for the s_2 tests performed in the second stage:

$$v = \frac{(s_1 + s_2)x - s_1 u}{s_2}. \quad (5.18)$$

From this and the law of total probability [3],

$$f(x) = \int_{y=-\infty}^{+\infty} f(x, y)\, dy, \quad (5.19)$$

we can find $f^{(2)}_{\bar{x},\mu,\sigma,s}$, the joint p.d.f. of the population mean and the sample mean of the candidate selected at the end of stage 2,

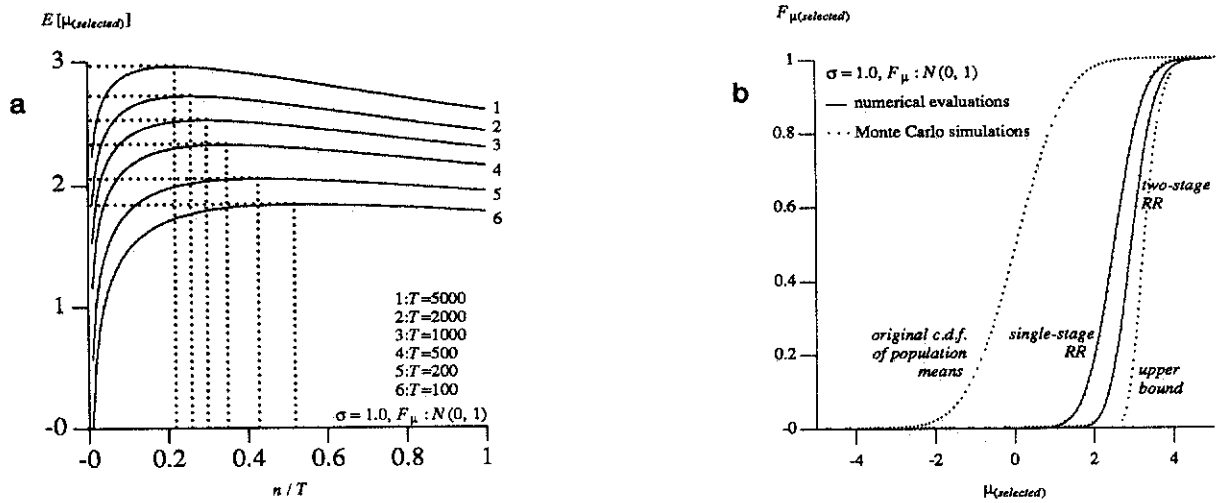

FIG. 5.2. Performance of static allocation strategies. (a) Effect of n on the performance of single-stage round-robin selection. (b) Performance of single-stage and two-stage round-robin strategies.

$$f^{(2)}_{\bar{x},\mu,\sigma,s}(x,\mu,\sigma,s) = \begin{cases} \int_{u=-\infty}^{+\infty} \text{Prob}[x,\mu,\sigma,s_1+s_2,\text{sample mean } u \text{ for } s_1 \text{ at the end of stage 1}]\,du \\ \quad \text{for } s = s_1 + s_2, \\ 0 \quad \text{otherwise,} \end{cases}$$

$$= \begin{cases} \int_{u=-\infty}^{+\infty} f_{\bar{x}|\mu,\sigma,s}\left(\dfrac{(s_1+s_2)x - s_1 u}{s_2} \;\middle|\; \mu,\sigma,s_2\right) f_{\bar{x},\mu,\sigma,s,(n_2:n_1)}\left(u,\mu,\sigma,\dfrac{rT}{n_1}\right) du \\ \quad \text{for } s = s_1 + s_2, \\ 0 \quad \text{otherwise,} \end{cases} \tag{5.20}$$

where $f_{\bar{x},\mu,\sigma,s,(n_2:n_1)}$ is the joint distribution of the candidates at the beginning of stage 2. Equation (5.20) can then be solved using Eq. (5.17).

When $s_2 \gg s_1$, the above equation can be simplified by considering the s_1 samples selected in stage 1 to be completely random when analyzing the performance of the second stage. In this case, we ignore the conditional distribution of the sample means obtained in the first stage. With this simplifying assumption and using Eq. (5.10), we can find $f_{\mu(selected)}$ in terms of $g_{\mu,(n_2:n_1)}$, the p.d.f. of the population mean of the top n_2 populations from the first stage:

$$g_{\mu,(n_2:n_1)}(\mu) = \int_{x=-\infty}^{+\infty}\int_{\sigma=-\infty}^{+\infty} f_{\bar{x},\mu,\sigma,s,(n_2:n_1)}\left(x,\mu,\sigma,\frac{rT}{n_1}\right) d\sigma\,dx \tag{5.21}$$

$$f_{\mu(selected)}(\mu) = n g_{\mu,(n_2:n_1)}(\mu)\left\{\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f_{\bar{x}|\mu,\sigma,s}(x|\mu,\sigma,s_1+s_2)\right. \tag{5.22}$$

$$\left.\times\left[\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} F_{\bar{x}|\mu,\sigma,s}(x|y,z,s_1+s_2)\, g_{\mu,(n_2:n_1)}(y) f_\sigma(z)\,dz\,dy\right]^{n-1} dx\,d\sigma\right\}.$$

The distribution of $\mu_{(selected)}$ for single-stage and two-stage round-robin strategies for a problem with normally distributed population means is shown in Fig. 5.2b. The analytical values are computed using Eqs. (5.12) and

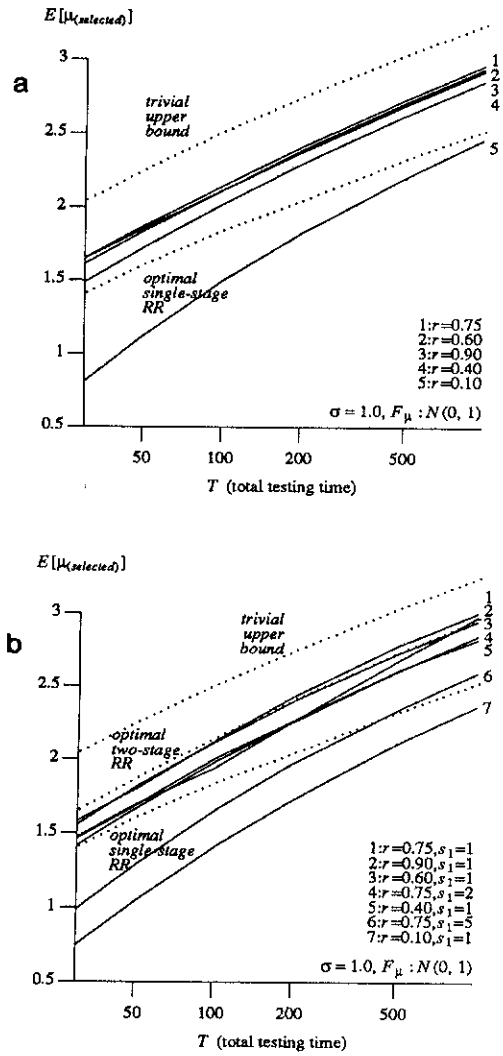

FIG. 5.3. Effect of r with $f\mu = N(0, 1)$ and $\sigma = 1$. (a) A two-stage round-robin method. (b) A two-stage round-robin/greedy method with various s_1.

(5.21) and are compared to the results of Monte Carlo simulations, the upper bound defined in Eq. (5.7), and the original f_μ. We see that a two-stage round-robin strategy performs much better than even the best single-stage round-robin strategy.

A two-stage round-robin strategy has more parameters to be selected, namely, r, n_1, and n_2. One method for finding appropriate values for these parameters is to make the same assumption we made for the single-stage round-robin strategy; that is, there is only one local maximum which is also the global optimum. With this assumption, the values of r, n_1, and n_2 can be determined by using a binary search. This is far more efficient than enumerating all possible combinations of r, n_1, and n_2. Figure 5.3a shows the results of using this method to select parameters for various values of r. From this figure, $r = 0.75$ appears to lead to the best performance.

5.3.4. *Dynamic Allocation Strategies*

Dynamic allocation strategies select populations to test based on dynamic performance information. In our study, we assume that the population selected is based on the mean, variance, or distribution of performance values gathered so far for all populations under consideration. All dynamic strategies require the application of a static strategy in the first stage in order to gather the initial statistics necessary for decision making. In this section we present two dynamic allocation strategies and their performance evaluation.

The simplest form of dynamic allocation strategy is the *two-stage round-robin/greedy* strategy, $\{G_1(RR, rT, n_1), G_2(\text{Greedy}, (1 - r)T, n_2)\}$. With the greedy method, samples are taken from the population that currently has the maximum sample mean in the second stage. The problem with this method is that it considers only populations that look good in the current stage and might discard the best population early in the process.

One dynamic strategy that focuses on the population with high sample means, but also tests the ones with smaller means if they were not tested enough, was proposed by Tong and Wetzell [16]. Their adaptive allocation strategy was designed to optimize $P(CS)$ when the selection process ended. Since we are interested in maximizing the mean of the selected population rather than $P(CS)$, we have chosen to develop a similar allocation strategy using this new objective. We define a risk function expressed as the expected squared-error loss of the mean estimation of each population, weighted by the degree of belief for the selection calculated using a joint t-distribution. This new strategy selects the next population so that the weighted risk is locally minimized. Using Monte Carlo simulations, we find that this new *minimum-risk* allocation strategy performs as well as Tong and Wetzell's strategy.

To minimize the risk at a future stopping point, dynamic programming can be used to find the best population to test in each future time instant so that the expected risk when time is expended is minimized. This allows time constraint to be considered in the sampling process. However, the complexity of the method is very high because the number of choices in each step is the same as the number of populations, and the number of possible choices grows exponentially with the number of steps. In practice, only a one-step lookahead policy that selects the population with the minimum risk is practical.

Risk Function. Let m be the number of populations. For the ith-order population based on sample means, let μ_i be its population mean, σ_i be its population standard deviation, $\bar{x}_i$ be its sample mean, S_i be its unbiased sam-

ple standard deviation, and l_i^k be the number of samples tested in stage k. If L_i, the estimated loss of the population mean values for the ith-order population, is expressed using the squared-error function, then

$$L_i(\mu_i, \bar{x}_i) = (\mu_i - \bar{x}_i)^2. \tag{5.23}$$

Given $\bar{x}_i$ and σ_i, we can calculate the expected value of L_i noting that $E[L_i(\mu_i, \bar{x}_i)] = Var[\mu_i - \bar{x}_i]$.

When σ_i's are known, the random variable $(\mu_i - \bar{x}_i)\sqrt{l_i^k}/\sigma_i$ has a standard normal distribution, $N(0, 1)$. In contrast, when σ_i's are unknown, the random variable $(\mu_i - \bar{x}_i)\sqrt{l_i^k}/S_i$ has a Student's t-distribution with $l_i^k - 1$ degrees of freedom. From these distributions, we can derive $E[L_i]$ when σ_i's are known,

$$E[L_i(\mu_i, \bar{x}_i)] = Var[\mu_i - \bar{x}_i] = \frac{\sigma_i^2}{l_i^k}, \tag{5.24}$$

as well as when σ_i's are unknown,

$$E[L_i(\mu_i, \bar{x}_i)] = Var[\mu_i - \bar{x}_i] = \left(1 + \frac{2}{l_i^k - 3}\right)\frac{S_i^2}{l_i^k} \quad \text{for } l_i^k > 3. \tag{5.25}$$

When the population that currently has the maximum sample mean is selected in stage k, the expected estimation loss (or risk R_k) is expressed as

$$R^k = E[L^k(\mu_{(best)}, \bar{x}_{(best)})] = \sum_{i=1}^{m} P_i^k \, E[L_i(\mu_i, \bar{x}_i)], \tag{5.26}$$

where P_i^k represents the probability that the ith-order population has the best population mean based on the current information in stage k.

The value of P_i^k is calculated using the joint probability of (a) population i having population mean y, and (b) all populations $j \neq i$ having population mean smaller than y. When the values of σ_i's are known, P_i^k can be expressed as

$$P_i^k = \int \prod_{j \neq i} \Phi\left(\frac{y - \bar{x}_j}{\sigma_j/\sqrt{l_j^k}}\right) d\Phi\left(\frac{y - \bar{x}_i}{\sigma_i/\sqrt{l_i^k}}\right), \tag{5.27}$$

where $\Phi(x)$ is the c.d.f. for an $N(0, 1)$ distribution. When the values of σ_i's are unknown, P_i^k can be expresses as

$$P_i^k = \int \prod_{j \neq i} F_t\left(l_j^k - 1, \frac{y - \bar{x}_j}{S_j/\sqrt{l_j^k - 1}}\right) \times dF_t\left(l_i^k - 1, \frac{y - \bar{x}_i}{S_i/\sqrt{l_i^k - 1}}\right), \tag{5.28}$$

where $F_t(v, x)$ is the c.d.f. of the t distribution with v degree of freedom. The value of risk R^k in Eq. (5.26) can then be solved using Eqs. (5.24) and (5.27), or Eqs. (5.25) and (5.28), depending on whether σ_i's are known.

One-Stage Look-Ahead Policy. With a one-stage look-ahead policy, the strategy to be taken in stage k is the one that, under the constraint that only one of l_i^k can be increased by 1, minimizes Eq. (5.26) in stage $(k + 1)$.

ONE-STAGE POLICY: $l_j^{k+1} = l_j^k + 1$

$$\text{when } j = \min_i \{R^{k+1} \mid l_i^{k+1} = l_i^k + 1\}. \tag{5.29}$$

In the special case in which one knows the "best" population, the choice made by this strategy is simple: only the "best" population is tested because $P_i^k = 1$ if μ_i is the highest one, and $P_i^k = 0$ for other populations.

In the normal case it is assumed that the value P_i^k is changing slowly and $P_i^{k+1} \approx P_i^k$. The actual value of P_i^k is calculated based on the information we currently have in stage k. This implies that we need at least four samples from each population in order to apply the minimum-risk allocation strategy when σ_i's are unknown (see Eq. (5.25)).

Approximate Minimum-Risk Method. The complexity of Eqs. (5.27) and (5.28) makes it difficult to apply the minimum-risk method even for small population sizes. To reduce this complexity, we approximate P_i^k for the case when σ_i's are *known*. In this approximation, we compute P_i^k as the joint probabilities of (a) the top ordered population (i.e., the one with the highest sample mean) having population mean higher than the second one, and (b) other populations having population means higher than the top one. Therefore, $\hat{P}_i^k$, the approximate P_i^k, becomes

$$\hat{P}_i^k = \begin{cases} \int \Phi\left(\dfrac{y - \bar{x}_{n-1}}{\sigma_{n-1}/\sqrt{l_{n-1}^k}}\right) d\Phi\left(\dfrac{y - \bar{x}_i}{\sigma_i/\sqrt{l_i^k}}\right) & i = n, \\ \int \Phi\left(\dfrac{y - \bar{x}_n}{\sigma_n/\sqrt{l_n^k}}\right) d\Phi\left(\dfrac{y - \bar{x}_i}{\sigma_i/\sqrt{l_i^k}}\right) & i \neq n, \end{cases} \tag{5.30}$$

where $\bar{x}_i$ represents the sample mean of the ith-order population, and n is the total number of populations under consideration. Equation (5.30) can be further simplified using the fact that the difference of two normal distribution $N(\mu_a, \sigma_a^2)$ and $N(\mu_b, \sigma_b^2)$ is also a normal distribution $N(\mu_a - \mu_b, \sigma_a^2 + \sigma_b^2)$. Hence,

$$\hat{P}_i^k = \begin{cases} \Phi\left(\dfrac{\bar{x}_i - \bar{x}_{n-1}}{\sqrt{\sigma_i^2/l_i^k + \sigma_{n-1}^2/l_{n-1}^k}}\right) & i = n, \\ \Phi\left(\dfrac{\bar{x}_i - \bar{x}_n}{\sqrt{\sigma_i^2/l_i^k + \sigma_n^2/l_n^k}}\right) & i \neq n. \end{cases} \tag{5.31}$$

With this approximation, the total computation time

for the minimum-risk method is reduced significantly from $O(n^3)$ to $O(n)$. Our simulation study shows the performance of this approximation is slightly worse than that of the original minimum-risk method.

For the case in which σ_i's are unknown, Eq. (5.31) can also be used to find $\hat{P}_i^k$. In this case, the sample standard deviations can be used instead of the population standard deviations.

Performance Analysis. To analyze dynamic allocation strategies properly, each sample drawn must be treated as an individual stage with $k = n = n_2$. This is true because f_{joint} depends on the joint distribution before the test and changes each time a new test is done.

The equations derived are very complex and time consuming to evaluate, even for the simplest dynamic allocation strategy. For studying the performance of dynamic strategies, it suffices to use Monte Carlo simulations. Note that it is much harder to determine the appropriate values for the parameters of dynamic allocation strategies. Simulation results in Fig. 5.3b show that a two-stage round-robin/greedy strategy can perform better than the optimal two-stage round-robin strategy. However, if its parameters are not set properly, its performance can be worse than even the single-stage round-robin strategy. The results also indicate that the case with $r = 0.75$ and $s_1 = 1$ (one sample drawn from each population in stage 1) performs very well. This turns out to be the best for a two-stage round-robin strategy in the range of parameters studied. As a heuristic approach for selecting parameters for two-stage dynamic strategies, the same r, n_1, and n_2 as for a two-stage round-robin strategy can be used.

Figure 5.4 shows the average performance, $E[\mu_{(selected)}]$. The results are plotted over a range of deadlines. The dynamic strategies use the same parameter values as the optimal two-stage round-robin strategy.

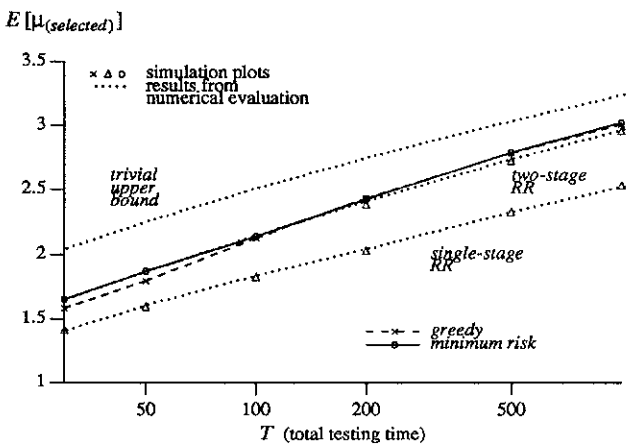

FIG. 5.4. Performance comparison for single-stage round-robin, two-stage round-robin, two-stage round-robin/greedy, and two-stage round-robin/minimum-risk allocation strategies for $f\mu = N(0, 1)$ and $\sigma = 1$.

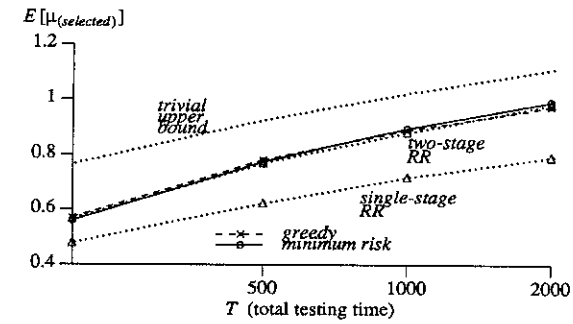

FIG. 5.5. Performance of multistage strategies under stochastic sampling overheads.

These results show that dynamic strategies can outperform the best static strategies.

5.3.5. *Allocation Strategies under Stochastic Sampling Overhead*

When the sampling overhead is stochastic, there are variations on the number of samples (or tests) drawn in different runs. An accurate analysis would require the knowledge of the distribution of the number of samples drawn from each population in a stage of the selection process, a term that is a high-degree convolution of the distribution of sampling overheads, $g_c(x)$. The analysis is even more complex when the performance value of the sample drawn and its corresponding overhead are correlated, which happens frequently since better heuristics tend to take more time. This is true for the learning of post-game heuristics in this paper. Due to these reasons, we will not attempt to analyze exactly the performance of allocation strategies when the sampling overhead is stochastic.

To find the approximate performance, we can use the average number of samples drawn in each stage instead of the exact distribution [10]. This is equal to T, the total amount of time available, divided by $\bar{c}$, the average sampling overhead. The performance of multistage selection strategies with T time units and stochastic sampling overhead is then approximated by the performance of multistage selection strategies with $T/\bar{c}$ time units and unit sampling overhead.

Figure 5.5 shows the effectiveness of this approximation. The average performance, $E[\mu_{(selected)}]$, for several multistage strategies are plotted over a range of deadlines. The performance value in this case is defined as q/c, where c is the sampling overhead and q is the quality measure with distribution $N(0, 1)$ and $\sigma = 1$. The distribution of the sampling overhead is specified as follows:

$$g_c(x) = \begin{cases} 0.4e^{-0.4(x-2)} & x \geq 2 \\ 0 & x < 2. \end{cases} \tag{5.32}$$

The parameters for two-stage strategies are selected using the optimal parameters of two-stage round-robin strategy for time = $T/\bar{c}$; $f_\mu(\mu)$ is $N(0, 1/\bar{c}^2)$; and $\sigma = 1/\bar{c}$.

5.4. Application Strategy for Learning Post-Game Heuristics

In order to apply the scheduling strategies we have developed to the learning of post-game heuristics, we must first verify that the assumptions made in the analytical model are correct for the process mapping problem. Experimental verification that the distribution of performance values within a population is normal, that the distribution of population means is normal, and that the variances of all populations are identical are shown in Section 6.1. The actual values of these parameters and the overhead for testing a heuristic method have to estimated before learning begins.

Based on the general application strategy described in Section 5.2, the resource scheduling strategy for learning post-game heuristics can be summarized as follows:

(1) During the initial 10% of the allotted time T, the presampling strategy described in Section 5.2.1 is applied to estimate (a) average μ_0 and variance σ_0^2 of all sample means of populations, (b) variance σ^2 of each candidate, assuming all candidates have the same variance, and (c) the average sampling overhead, $\bar{c} = T/(10k)$, where k is the total number of tests performed during the presampling period. The average number of tests that can be performed in the entire duration is $\bar{T} = T/\bar{c} = 10k$.

(2) Assuming that the distribution of the population means is a normal distribution, $N(\mu_0, \sigma_0^2)$, and that 75% of the remaining time is allocated to the first stage of a two-stage round-robin selection algorithm ($r = 0.75$), the scheduler uses a binary search to find n_1 and n_2, the optimal number of populations to be sampled in the two stages. These values are computed using Eq. (5.21) using $f_\mu \approx N(\mu_0, \sigma_0^2)$ with the values μ_0, σ_0^2, and σ^2 obtained in Step 1, and $\bar{T}$ as the total number of tests available.

(3) We choose to use the two-stage round-robin/minimum-risk strategy as the primary resource scheduling strategy in the remaining time based on parameters r, n_1, and n_2 determined in Step 2. This strategy is more robust and is less likely to fail than other multistage strategies we have evaluated by Monte Carlo simulations.

(4) The two-stage round-robin/minimum-risk is carried out until the time is expended. At the end of the second stage, the heuristic method with the highest average performance is returned.

6. EXPERIMENTAL RESULTS

In this section we present the results of applying TEACHER 4.1 to learn post-game heuristics. First, we describe verifications of assumptions made in the analytical model and the learning experiments performed. Next, we present plots for showing the tradeoff between cost and quality of the heuristic methods selected by the learning system. Finally, we demonstrate scalability and generality of the heuristic methods learned.

6.1. Post-Game Heuristics Learning Experiments

To evaluate our method for learning heuristics for post-game analysis, we apply the post-game system on a target problem based on the divide-and-conquer paradigm. In this application, each process computes for a random amount of time, sends a message to each of its child processes in order to start their computation, and waits for the results from its descendents before reporting to its parent. There are a total of 105 processes that are mapped to a 3-by-3 mesh architecture. Twenty problem instances, each with a common communication pattern and CPU times for each code segment drawn from a uniform distribution ($U(50, 150)$), are included in the Test-Case Database.

The 20 test cases are evaluated for 100 different heuristic methods. Each set of 20 points is then tested for normality using both the Kolmogorov–Smirnov Test and the Geary Test [3]. The results summarized in Table 6.1 demonstrate the validity of the normality assumption.

Figure 6.1 shows the cumulative distribution of the mean performance (Eq. (4.2)) of 100 random heuristic methods, each evaluated on 20 test cases with 31 initial mappings for each test case. The graph illustrates our assumption that the distribution of population means is normal.

We apply the learning system described in Sections 4 and 5 to learn new post-game heuristic methods for several intermediate objective functions used in the Internal Critic. Since the actual learning experiments are very time-consuming, we use results from actual as well as

TABLE 6.1

Test of Normality for 100 Random Heuristic Methods, Each Run with 20 Test Cases and Averaged over 31 Initial Mappings for Each Test Case

T_{scale}	Test	Min	Avg	Max	#Fails /Total
15	Kolmogorov–Smirnov ($\alpha = 0.05$, KS < 0.19)	0.06314	0.122008	0.228907	4/100
	Geary Test ($\alpha = 0.05$, |G| < 1.96)	0.007155	0.800058	2.302100	6/100
50	Kolmogorov–Smirnov ($\alpha = 0.05$, KS < 0.19)	0.061655	0.118434	0.195710	2/100
	Geary Test ($\alpha = 0.05$, |G| < 1.96)	0.007323	0.685574	2.15586	3/100

Note. At $\alpha = 0.05$, Kolmogorov–Smirnov test with 20 samples requires the value to be less than 0.19, while Geary Test at $\alpha = 0.05$ requires the absolute value to be less than 1.96 [3].

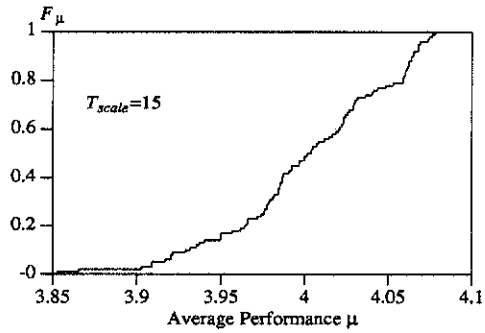

FIG. 6.1. Distribution of population means of 100 random heuristic methods used in post-game analysis; performance is computed using Eq. (4.2).

simulated runs to present a full picture of the learning system performance. In the simulation run, only the second stage of the learning experiment is actually executed; the first stage is simulated using statistical data of heuristic methods gathered during an actual run. The savings in execution time is significant, as a simulation run requires only 25% of the time required by an actual run. The problem with simulations is that heuristics generation is not modeled, as only heuristic methods generated in a previous run are used during the simulation. Some justification to this approach can be found by comparing the results between the actual and simulation runs shown later in Table 6.2.

TABLE 6.2
Result of Learning for Different T_{scale}

T_{scale}	Experiment type	Intermediate performance (orig = 4.00) (Eq. (4.2))	Number of quantum	Average normalized performance (Eq's 4.3a and 4.3c, Orig = 1.0)	
				Cost	Mapping quality
1	SIM	20.6870	305	0.1292	0.9042
8	SIM	4.2020	290	0.2740	0.9697
15	RUN	4.0764	258	0.2740	0.9697
30	RUN	4.0077	191	0.3172	0.9727
50	RUN	4.0087	128	2.2063	1.0133
100	SIM	4.0537	147	4.7188	1.0360
1000	SIM	4.0726	149	4.7188	1.0360
∞	RUN	4.0481	109	4.5179	1.0225

Note. Time allowed for learning is 16 h of CPU time on a DECStation 5000 Model 200; the strategy used is two-stage round-robin/minimum-risk; the time for the initial presampling period is 96 min. The execution time taken by post-game analysis averaged over all test cases using the existing heuristic method of Yan [21] with Lundstrom [20] is 19.89 min per test case on a DECStation 5000/200. The corresponding average completion time of the mappings found by the original heuristic methods is 1454.05 seconds.

We carried out two experiments to evaluate our learning system. In the first experiment, we applied the learning system to learn post-game heuristic methods for intermediate objective function $Q(T_{scale})$ (Eq. (4.2)) with four different values of T_{scale}. The pool of initial heuristic methods for all experiments consists of 10 predefined heuristic methods. Information collected during the learning experiment for T_{scale} of 15 was then used to simulate the learning experiment for 4 other values of T_{scale}. At the end of each learning experiment, we fully evaluate the selected heuristic method over the entire training database. The duration of an actual run is 16 h of CPU time on a DEC-Station 5000 model 200 with 16 Mbytes of memory; each quantum of time corresponds to evaluating a heuristic method over one test case with up to 31 initial mappings. This is done by testing each test case until a 97.5% confidence interval (based on the Student's t-distribution) of the average performance across the initial mappings is within 3% of the average value.

In the second experiment, we investigated the effect of learning time on the quality of the heuristic methods selected by our learning system. Based on the heuristic methods and their performance information gathered during an actual learning experiment with T_{scale} of 15, we simulated learning experiments with durations ranging from 4 h to 16 h and three different values of T_{scale}.

In the next section, we describe the step involved in evaluating the performance results we have obtained from our experiments. The actual performance results are presented in Section 6.3.

6.2. Performance-Tradeoff Plots

In Section 4.3, we explain why the average performance information by itself does not provide a complete picture on the effectiveness of the heuristic methods selected and the tradeoff between cost and quality of the mapping obtained by each heuristic method. To evaluate properly the performance of the various heuristic methods, we need to evaluate cost and quality separately. We propose using the plot of the normalized quality of a heuristic method on each test case against its normalized cost on the same test case as a way for evaluating the performance of that heuristic method.

The first step in obtaining this plot is to compute the normalized values of the cost and mapping quality by Eq. (4.3). Although a scatter plot can be achieved with this step, additional steps are required to obtain a more comprehensive view of the performance of heuristic methods.

Second, we eliminate from the performance data any anomalous data points which might unduly influence the statistical result we want to observe. We are mainly inter-

ested in *univariate outliers*, i.e., the few extreme cost or mapping quality values which do not fit with the other data points. This can be detected simply by computing the standardized score $(x - \mu/\sigma)$ of the normalized cost and quality for the given heuristic method. Any data point with standardized scores in excess of 8.00, i.e., the point is more than eight standard deviations away from the mean, is considered an outlier and is removed. Note that this computation does not assume any underlying distribution of the data. *Multivariate outliers*, which are the cases that have an unusual combination of cost and quality, can also be detected by computing the Mahalanobis distance for each data point. A data point with Mahalanobis distance that have less than 0.001 probability of happening can be considered an outlier [15]. The application of this method requires that the joint distribution is a multivariate normal distribution. We did not apply multivariate-outlier detection here because the performance data after eliminating the very few (5 out of 9000 data points) univariate outliers appear to be reasonable.

Next, we determine the joint distribution between the two normalized measures for the given heuristic method. Based on the performance data, we hypothesized that the joint distribution is a bivariate normal distribution. To validate this assumption, we first checked that the marginal distributions of cost and quality were normal. There are many methods for evaluating univariate normality; examples include computing the values of skewness and kurtosis [15], applying the goodness-of-fit test, such as Kolmogorov–Smirnov and Geary tests, and applying the Shapiro–Wilk test [4]. We have applied all these tests on our performance data; with only one or two exceptions, normality was accepted at level $\alpha = 0.1$. Next, we evaluated bivariate normality of the joint distribution. The method we used was the chi-square plot or gamma plot [11], which depicts the ordered square distance from the centroid (mean values) to each data point against the chi-square distribution. If the data have a bivariate normal distribution, this plot would appear as a straight line. Again, bivariate normality was verified for our data. Other methods for checking multivariate normality can be found in the reference [4]. If the performance values do not have a normal distribution, then a distribution has to be first assumed so that the values can be transformed into another set which can be tested for normality.

Finally, we obtained the 90% constant probability density contour of the bivariate normal distribution representing the joint distribution of the normalized cost and mapping quality. To do so, we first computed the mean vector, $\mathbf{M}$, and the variance-covariance matrix, Σ, for the distribution. We then derived the axes for the ellipse that represented this contour from the eigenvalues $\{\lambda_1, \lambda_2\}$ and eigenvectors $\{\mathbf{e}_1, \mathbf{e}_2\}$ of the covariance matrix, Σ. The axes of the ellipse were represented as the vectors $\chi_2^2(0.1)\sqrt{\lambda_1}\mathbf{e}_1$ and $\chi_2^2(0.1)\sqrt{\lambda_2}\mathbf{e}_2$, respectively [11]. These two vectors represent the direction and length of the axes for the ellipse with respect to the centroid of the data set, $\mathbf{M}$. The resulting contour encompasses the area where 90% of the data from the given bivariate normal distribution should appear. If the original performance values are not normally distributed and a transformation was made in the previous step, then the contour found in this step has to be converted using the inverse transformation into one for the original data points.

6.3. Experimental Results on Learning Heuristic Methods

Using a DEC-Station 5000 model 200 with 16 Mbytes of memory, it took approximately 30 days of CPU time to acquire all the results shown in Section 6. The data shown in Section 6.1, which were based on fully evaluating 100 randomly generated heuristic methods, required about 14 days of CPU time. Each actual learning experiment (with 16 h of learning time) took approximately 48 h of CPU time to complete, while each simulated learning experiment took approximately 8 h of CPU time. Finally, for the data shown in Section 6.3 and 6.4 on fully evaluating the heuristic methods selected, about 7 days of CPU time were used.

Table 6.2 shows the performance of heuristic methods learned using different T_{scale}. The third column of Table 6.2 shows the objective value of $Q(T_{scale})$ reported by the Internal Critic when learning is terminated. The last two columns show the performance by evaluating the selected heuristic methods exhaustively over the entire training set. We see that our learning system consistently found heuristic methods that outperform the original post-game heuristic methods of Yan [21] and Lundstrom, which have a normalized average cost and quality of 1.0. Moreover, the intermediate objective $Q(T_{scale})$ is quite consistent with the results obtained by exhaustive evaluation. Note that the original heuristic methods already generate mappings of high quality. As a result, heuristic methods that generate mappings of higher quality is more difficult to find: a small improvement in quality means a large increase in cost. On the other hand, our learning system was able to find much faster heuristic methods that have minor degradation in quality.

Table 6.2 also illustrates the dependency of the intermediate objective function on the sampling overhead. The number of selections finished during learning decrease as T_{scale} increases. This happens because heuristic methods that perform well for higher values of T_{scale} are likely to require higher sampling overhead, and the number of quanta completed during the training period is reduced as T_{scale} increases. These observations show that if the objective function is dependent on the sampling overhead, then the distribution of sampling overheads for good heuristic methods are likely to be different from that for random heuristic methods.

Figure 6.2 shows the normalized graphical plot of cost

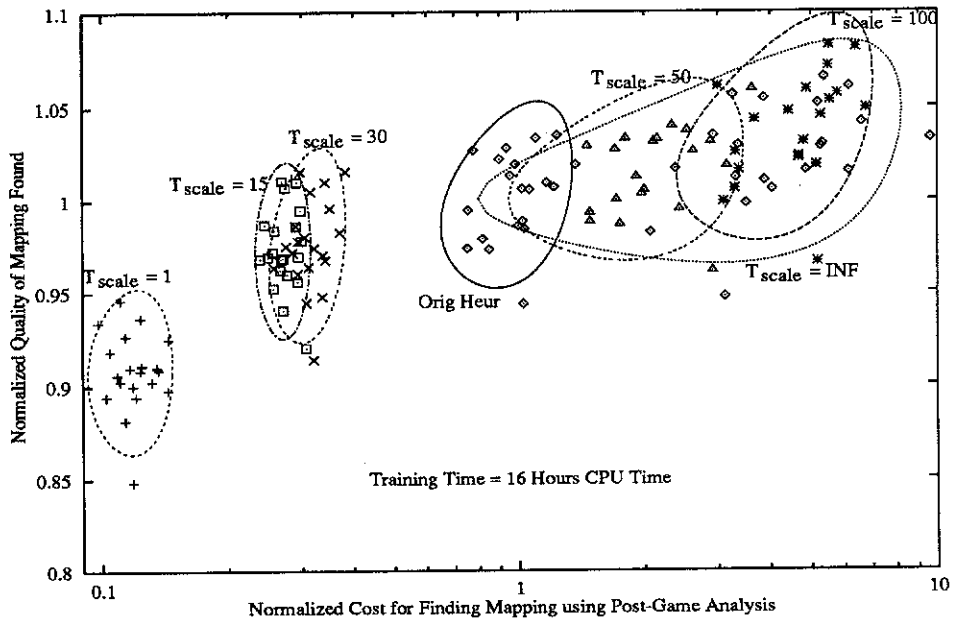

FIG. 6.2. Performance of heuristic methods selected by different intermediate objective functions on test cases in the training database.

versus mapping quality for some of the heuristic methods shown in Table 6.2. This plot, obtained using the steps described in Section 6.2, provides a better picture of the tradeoff between cost and quality for each heuristic method than the information shown in Table 6.2. Each data point in this figure corresponds to the average performance of a heuristic method on a test case in the training database. The graph shows the tradeoff between the quality of the mapping found and the execution time of the heuristic methods selected for various values of T_{scale}. Recall that T_{scale} controls the relative importance between these two measures; as T_{scale} increases, the execution time of the heuristic method becomes less important. Consequently, both the cost of the heuristic method selected and the quality of the mapping found are improved. To choose the heuristic method appropriate for the application, an automated learning system that aids users in experimenting with various alternatives is, therefore, very important.

Figure 6.3 shows the normalized plot for the second set

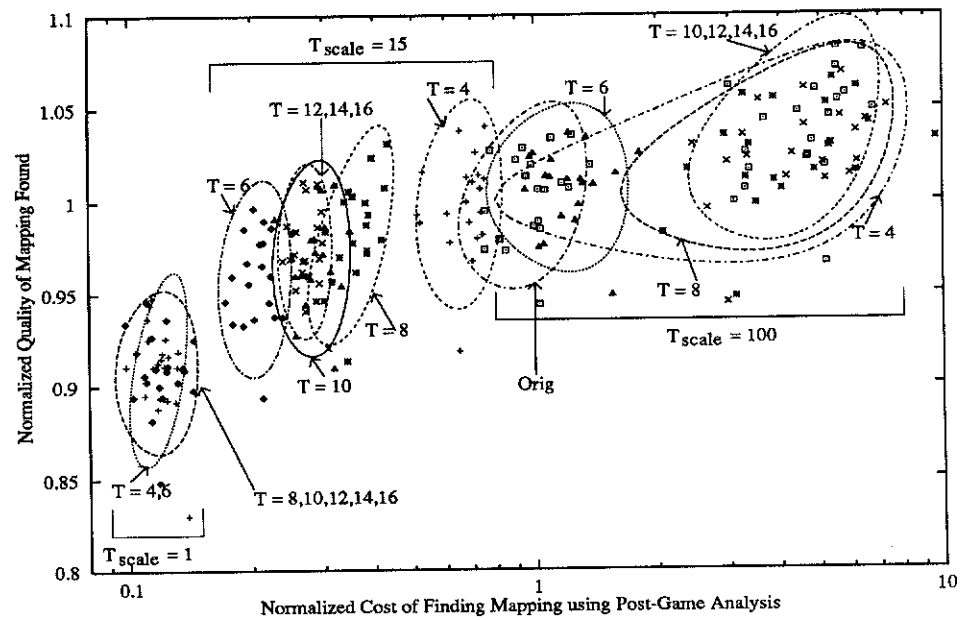

FIG. 6.3. Performance of heuristic methods selected using different learning time, T, ranging from 4 to 16 h.

of experiments in which the amount of learning time is varied. The learning experiments were performed for three different values of T_{scale} (1, 15, and 100) and seven different learning times (4, 6, 8, 10, 12, 14, and 16 Hours). Only one of the results is derived from an actual run of the learning system (T_{scale} = 15 and learning time = 16 h). Simulation runs based on this run were used to select the other heuristic methods.

From Figs. 6.2 and 6.3, we observe that the cost for finding a mapping increases exponentially with respect to increase in the quality of the mapping found. We also observe that the quality of the mapping found by the post-game system is already very good. Even after 16 h of learning, the best heuristic method (T_{scale} = 100 in Fig. 6.3) provides only 5% improvement in the quality, while the cost increases by more than four times. On the other hand, we can reduce the cost significantly with only minor degradation in mapping quality. For example, Fig. 6.3 shows that with 4 h of learning time, our learning system can find a heuristic method that uses only 10% of the cost of the existing post-game heuristic method and finds mappings that are 10% worse (T_{scale} = 1).

Figure 6.3 also shows the improvement in performance of the heuristic methods selected as learning time increases. This is more obvious for T_{scale} = 1 and 100. For T_{scale} = 1, the heuristic methods selected have lower cost as the learning time increases. For T_{scale} = 100, the heuristic methods selected not only have higher mapping quality as the learning time increases, but also have more consistent performance in term of both cost and quality. For T_{scale} = 15, the improvement in performance is harder to characterize. Even in this case, we observe that the heuristic method selected after 16 h of learning has more consistent performance than that of other heuristic methods selected using less learning time.

6.4. Verification of Heuristics Performance

The experimental results in Table 6.2 indicate that our learning system produces heuristic methods that outperform existing post-game heuristics. However, before these heuristics can be applied in general, we need to verify that the performance of these heuristic methods are not restricted to the test cases defined in the training database.

To demonstrate the generality of the heuristic methods learned, we apply the heuristic methods selected on three other test sets. Each test set contains 10 test cases drawn from a unique distribution. The test cases in the first set are drawn from the same distribution as the test cases in the training database. This is used to verify that the performance of the heuristic method is not restricted to the training set, and that the performance is consistent across the problem domain. To check the scalability of the heuristic methods learned, test cases in the second test set are drawn from an implementation of a divide-and-conquer paradigm with 204 processes on a 16-node hypercube architecture. The CPU time for each code segment for this test set is also drawn from a uniform distribution $U(50, 150)$. Last, test cases in the third test set are drawn from an entirely different problem domain that implements a distributed blackboard program with 115 processes on a 3-by-3 mesh architecture. The idle times between new observations for each observer are drawn from a uniform distribution $U(0, 12)$.

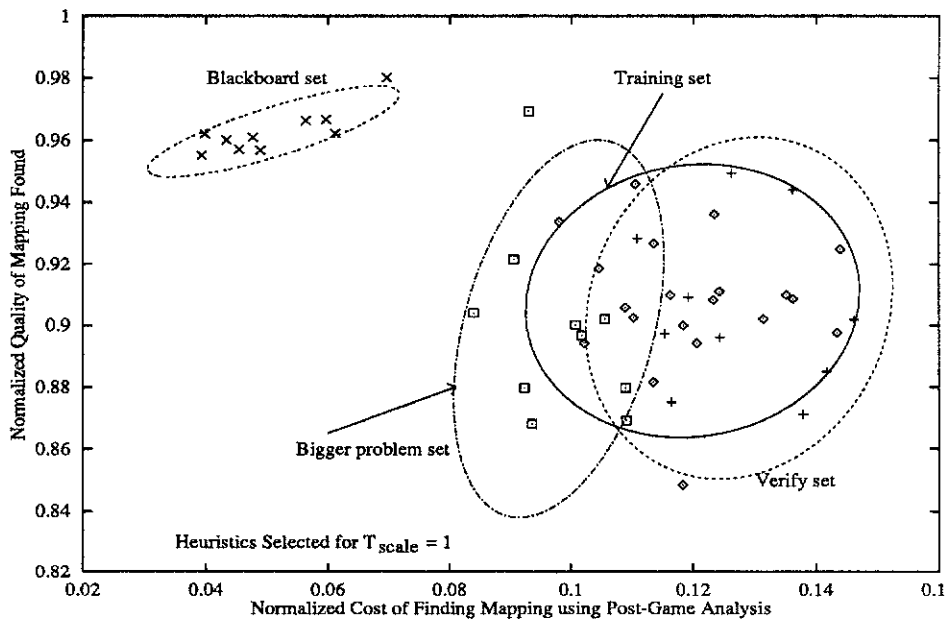

FIG. 6.4. Performance of heuristic methods selected for T_{scale} = 1 on different test sets.

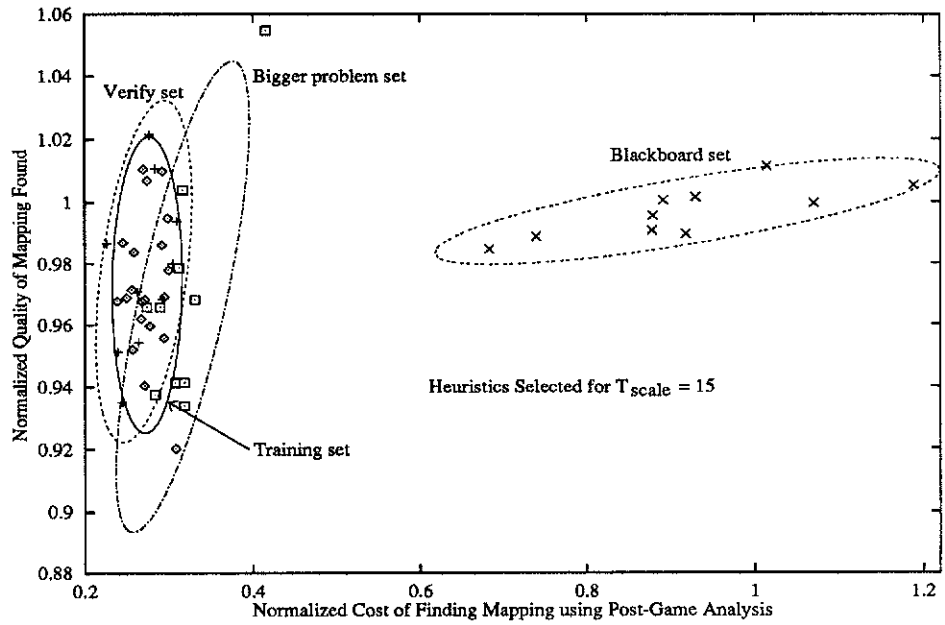

FIG. 6.5. Performance of heuristic methods selected for T_{scale} = 15 on different test sets.

Figures 6.4 and 6.5 compare the performance of the heuristic methods learned for T_{scale} of 1 and 15 on the three test sets defined above. These plots show that the heuristic methods selected perform consistently for the type of problems represented by the training database. These figures also indicate that the heuristic methods learned have performance that can be generalized to large problems of the same type. However, the heuristic methods selected does not perform well with respect to test cases drawn from the distributed blackboard problems, which have different characteristics. To allow the heuristics learned to generalize, test cases from different problem types may have to be used during learning. The use of diverse test cases in learning will be studied in the future.

7. CONCLUSIONS

In this paper we present a population-based system for learning heuristics for mapping processes on a network of computers. Our system extends post-game analysis by systematically generating new heuristic methods in order to determine process mappings and test them on sample problems. We have modeled resource scheduling for the learning system as a statistical selection problem under time constraints. Based on some general assumptions, various multistage procedures for selecting the heuristic method with the best population mean are developed, evaluated, and tested. We also present methods for selecting appropriate parameters for the multistage procedures.

Our learning system is able to propose new heuristic methods that consistently outperform existing post-game heuristics obtained by extensive hand tuning. Our experiments also show that different heuristic methods may be required under different objectives and processing conditions. Our learning system provides an automated tool for generating new heuristic methods that can adapt to the target environment.

We have also developed a graphical method for comparing the performance of heuristic methods on different sets of test cases. Our method provides additional insight into selecting the proper heuristic methods for the target application. It is useful for verifying the generality and demonstrating the scalability of the heuristic methods selected. We show that our heuristic methods learned have performance that are scalable and can be generalized to a limited extent to other mapping problems.

REFERENCES

1. Bechhofer, R. E. A single-sample multiple decision procedure for ranking means of normal populations with known variances. *Ann. Math. Statist.* **25**, 1 (Mar. 1954), 16–39.
2. Bechhofer, R. E., Hayter, A. J., and Tamhane, A. C. Designing experiments for selecting the largest normal mean when the variances are known and unequal: Optimal sample size allocation. *J. Statist. Plann. Inference* **28** (1991), 271–289.
3. Devore, J. L. *Probability and Statistics for Engineering and the Sciences*. Brooks/Cole, Monterey, CA, 1982.
4. Gnanadesikan, R. *Methods for Statistical Data Analysis of Multivariate Observations*. Wiley, New York, 1977.
5. Goldberg, D. E., and Holland, J. H. Genetic algorithms and machine learning. *Mach. Learning* **3**, 2/3 (Oct. 1988), 95–100.

6. Grefenstette, J. J., Ramsey, C. L., and Schultz, A. C. Learning sequential decision rules using simulation models and competition. *Mach. Learning* **5** (1990), 355–381.
7. Gupta, S. S., and Panchapakesan, S. Sequential ranking and selection procedures. In Sen, P. K. (Ed.). *Handbook of Sequential Analysis*. Dekker, New York, 1991, pp. 363–380.
8. Hogg, R. V., and Tanis, E. A. *Probability and Statistical Inference*, 3rd ed. Macmillan Publishing Company/Collier Macmillan Publishers, New York, NY/London, England, 1988.
9. Ieumwananonthachai, A., Aizawa, A. N., Schwartz, S. R., Wah, B. W., and Yan, J. C. Intelligent mapping of communicating processes in distributed computing systems. *Proc. Supercomputing 91*. ACM/IEEE, Albuquerque, NM, Nov. 1991, pp. 512–521.
10. Ieumwananonthachai, A., and Wah, B. W. Learning process mapping heuristics under stochastic sampling overheads. *Proc. Computing in Aerospace 8 Conference*. American Institute of Aeronautics and Astronautics, Baltimore, MD, Oct. 1991.
11. Johnson, R. A., and Wichern, D. W. *Applied Multivariate Statistical Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1982.
12. Lowrie, M. B., and Wah, B. W. Learning heuristic functions for numeric optimization problems. *Proc. Computer Software and Applications Conf.* IEEE, Chicago, IL, Oct. 1988, pp. 443–450.
13. Mehra, P., and Wah, B. W. Architectures for strategy learning. In Wah, B., and Ramamoorthy, C. (Eds.). *Computer Architectures for Artificial Intelligence Applications*. Wiley, New York, 1990, pp. 395–468.
14. Stein, C. The selection of the largest of a number of means, abstract. *Ann. Math. Statist.* **19** (1948) 429.
15. Tabachnick, B. G., and Fidell, L. S. *Using Multivariate Statistics*, 2nd ed. Harper and Row, New York, 1989.
16. Tong, Y. L., and Wetzell, D. E. Allocation of observations for selecting the best normal population. In Santner, T. J., and Tamhane, A. C. (Eds.). *Design of Experiments: Ranking and Selection*. Dekker, New York, 1984, pp. 213–224.
17. Wah, B. W., and Kriplani, H. Resource constrained design of artificial neural networks. *Proc. Int'l Joint Conf. on Neural Networks*. IEEE, June 1990, Vol. III, pp. 269–279.
18. Wah, B. W. Population-based learning: A new method for learning from examples under resource constraints, IEEE Transactions on Knowledge and Data Engineering, to appear.
19. Yan, J. C. Post-game analysis—A heuristic resource management framework for concurrent systems. Ph.D. dissertation, Dept. Elec. Eng., Stanford Univ., Dec. 1988.
20. Yan, J. C., and Lundstrom, S. F. The post-game analysis framework—Developing resource management strategies for concurrent systems. *IEEE Trans. Knowledge Data Eng.* **1**, 3 (Sept. 1989).
21. Yan, J. C. New "post-game analysis" heuristics for mapping parallel computations to hypercubes. *Proc. Int'l Conf. on Parallel Processing*. CRC Press, Boca Raton, FL, Aug. 1991, Vol. II, pp. 236–242.
22. Yu, C. F., and Wah, B. W. Learning dominance relations in combinatorial search problems. *IEEE Trans. Software Eng.* **SE-14**, 8 (Aug. 1988), 1155–1175.

Received December 1991; accepted March 27, 1992

ARTHUR IEUMWANANONTHACHAI received his B.S. degree in electrical engineering and computer science from the University of Washington, Seattle, WA, in 1986, and his M.S. degree in computer science from the University of California, Los Angeles, in 1988. Since then, he has been working toward his Ph.D. degree in electrical and computer engineering at University of Illinois, Urbana-Champaign, under the supervision of Professor Benjamin Wah. His research interests include computer networks, distributed systems, computer vision, and machine learning.

AKIKO N. AIZAWA received the B.S. degree in 1985, the M.S. degree in 1987, and the Ph.D. degree in 1990, all in electrical engineering from the University of Tokyo. In 1990, she joined NACSIS (National Center for Science Information System), Japan. She is currently a visiting researcher at the University of Illinois at Urbana–Champaign. Her research interests include communication networks and protocols, distributed systems, database interface, and artificial intelligence.

STEVE R. SCHWARTZ is a Software Engineer at Motorola Cellular in Arlington Heights, IL. He is currently working on systems for the European Digital Cellular standard. He received a BSE in computer engineering from the University of Michigan in 1989, and an MS in electrical engineering from the University of Illinois at Urbana–Champaign in 1991. He is a member of Tau Beta Pi, IEEE, and the ACM.

BENJAMIN W. WAH is a Professor in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign, Urbana, IL. He has published extensively in the areas of computer architecture, parallel processing, artificial intelligence, and computer networks. He is a University Scholar of University of Illinois, an Associate Editor-in-Chief of IEEE Transactions on Knowledge and Data Engineering, member of the Board of Governors of IEEE Computer Society, and a Fellow of the IEEE. During part of 1992, he served as Fujitsu Visiting Chair Professor on Intelligence Engineering at University of Tokyo, Japan.

DR. JERRY C. YAN received his Ph.D. and MSEE from Stanford University. He was awarded the Siemen's Memorial Medal and the Governor's Prize at Imperial College, University of London, UK, where he received a BSEE. Dr. Yan is currently working for Sterling Software as a contractor at NASA Ames Research Center. He is a member of the Institute of Electronic and Electrical Engineers and Association for Computing Machinery. His research interests include parallel processing, performance evaluation, computer architecture, and operating systems.