AUTOMATED DESIGN OF KNOWLEDGE-LEAN HEURISTICS: LEARNING, RESOURCE SCHEDULING, AND GENERALIZATION

BY

ARTHUR IEUMWANANONTHACHAI

B.S.E.E., University of Washington, 1986
B.S., University of Washington, 1986 M.S., University of California at Los Angeles, 1988

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1996

Urbana, Illinois

© Copyright by Arthur Ieumwananonthachai, 1996

AUTOMATED DESIGN OF KNOWLEDGE-LEAN HEURISTICS: LEARNING, RESOURCE SCHEDULING, AND GENERALIZATION

Arthur Ieumwananonthachai, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 1996
Benjamin W. Wah, Advisor

In this thesis we present new methods for the automated design of new heuristics in knowledge-lean applications and for finding heuristics that can be generalized to unlearned test cases. These applications lack domain knowledge for credit assignment; hence, operators for composing new heuristics are generally model free, domain independent, and syntactic in nature. The operators we have used are genetics based; examples of which include mutation and crossover. Learning is based on a generate-and-test paradigm that maintains a pool of competing heuristics, tests them to a limited extent, creates new ones from those that perform well in the past, and prunes poor ones from the pool. We have studied four important issues in learning better heuristics: (a) partitioning of a problem domain into smaller subsets, called *subdomains*, so that performance values within each subdomain can be evaluated statistically, (b) anomalies in performance evaluation within a subdomain, (c) rational scheduling of limited computational resources in testing candidate heuristics in single-objective as well as multi-objective learning, and (d) finding heuristics that can be generalized to unlearned subdomains.

We show experimental results in learning better heuristics for (a) process placement for distributed-memory multicomputers, (b) node decomposition in a branch-and-bound search, (c) generation of test patterns in VLSI circuit testing, (d) VLSI cell placement and routing, and (e) blind equalization.

DEDICATION

To my parents, Ieum and Nantana Ieumwananonthachai

ACKNOWLEDGEMENTS

I would like to thank all of the people who have helped me throughout the course of my graduate study. First, I would like to especially thank my advisor, Professor Benjamin W. Wah, for his advice and guidance, both professionally and personally. Without his help, this research would not have been possible. I would like to express my gratitude for the other committee members, Professors Prithviraj Banerjee and David E. Goldberg. I sincerely thank Professor Pen-Chung Yew for his service on the committee in my preliminary examination.

I would like to thank Dr. Jerry Yan for Post-Game Analysis, Professor Dan Saab for CRIS, Professor Janak Patel and Dr. Elizabeth Rudnik for GATEST, and Professor Carl Sechen for TimberWolf. These are the problem solvers used in our experiments. I also would like to acknowledge the help and guidance that Dr. Jerry Yan has provided during the two summers I spent at NASA Ames Research Center.

Many thanks are due to numerous colleagues who have contributed to the development of TEACHER — Steven Schwartz and Akiko Aizawa, who helped lay the foundation; Lon-Chan Chu for his work on search and generalization; Pankaj Mehra and Yong-Cheng Li for adapting TEACHER to new applications; and Chien-Wei Li, Yao Shu, and Yu Ting for carrying on this work in the future.

I would like to acknowledge the financial support from the following sponsors: National Science Foundation Grants MIP 92-18715 and MIP 88-10584 and National Aeronautics and Space Administration Grants NCC 2-481, NAG 1-613, and NGT 50743 (NASA Graduate Fellowship Program).

I would also like to thank my current and former colleagues — Pankaj Mehra, Lon-Chan Chu, Kumar Ganapathy, Steven Schwartz, Akiko Aizawa, Vijay Karamcheti, Yi Shang, Alan Chang, Chin-Chi Teng, Tanay Karnik, Yong-Cheng Li, Chien-Wei Li, Yao Shu, Yu Ting, and Jeff Monks — for helpful discussions about ideas presented in this thesis. I wish to thank Donna Guzy and Chew-Chin Phua for their support and help.

I would like to thank my Thai friends, past and present, who have helped me cope with the pressures and frustrations of graduate school: Pee Somchai, Pee Add, Pee Tew, Pee Tim, Pee Kung, Pee Eie, Pee Joe, Pee Tee, Moo, Odd, Vee, and my nephew, Ton.

Among my family members, I first wish to thank my parents, whose love and sacrifice have brought me this far. I want to thank my brother, Ike, for many meals and many happy times we have spent together during the few years we were at Illinois. Those were the best times I had here. I also want to thank my sisters, Nantaporn and Ame, for their love and concern. I, therefore, wish to dedicate this thesis to my parents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The design of problem-solving algorithms for many applications generally relies on the expertise of designers and the amount of domain knowledge available. This design is difficult when there is little domain knowledge or when the environment under consideration is different from which the algorithm is applied. In this thesis we study an important problem in the automated design of problem-solving heuristics in knowledge-lean application environments.

Problem solvers using "heuristics" are applied in many applications when a problem cannot be solved optimally. Heuristics are commonsense knowledge that can be used to find suboptimal solutions for a problem without any guarantee on the resulting performance. Consequently, the performance of a problem solver can be affected by the choice of its heuristics.

At present, most heuristics are derived in an ad hoc fashion based on past experience of the designer. Often, the designer must spend a large amount of time to fine tune the heuristics in order to achieve a high level of performance. Since the number of possible heuristics is very large for realistic applications of reasonable complexity, heuristics designed manually may not work well when applied in new problem instances. Further, there is no systematic method to evaluate the effectiveness of heuristics designed manually. To address this problem, we need a well-designed automated system for learning heuristics that can solve application problems with a high level of performance and within a reasonable amount of time.

In the past, there have been many efforts on designing heuristics using machine learning techniques [1–9]; however, most of them [1,2,4–9] assume a world model that relates heuristics and their performance. This type of heuristics is known as "*knowledge-rich*" heuristics. In this research, we focus on learning for a part of an application problem with "*knowledge-lean*" heuristics where no such model exists. We address several shortcomings that exist in current approaches [3,10–12].

The objectives of this research are to (a) develop an automated system for designing knowledge-lean heuristics under resource constraints and (b) apply our system to learn heuristics for real-world applications.

There are a number of problems that are neglected in the development of a learning system for knowledge-lean heuristics. First, there are ambiguities and uncertainties in comparing the performance of different heuristics. In many cases, it is difficult to determine which heuristics provide the best performance. Second, due to the lack of a world model, it is very difficult to generate heuristics with guarantees on their performance. Third, due to the finite amount of computing resources available, it is necessary for the learning system to utilize these resources in an efficient fashion in order to produce as good a heuristic as possible within the given resource constraints. Fourth, different regions of an application domain can have different performance characteristics. It is necessary to identify these regions so that performance can be estimated separately for each region. It is also necessary to have an automated method for determining the true performance of each heuristic over the entire problem domain.

Our research is developed to directly address these problems. Within the scope of this research, we have defined (a) a strategy for evaluating the performance of heuristics, (b) a framework for learning new knowledge-lean heuristics, (c) a real-time resource scheduling strategy to test heuristics according to their partial performance information, and (d) a generalization strategy for determining the true performance of learned heuristics over the problem domain. These results have been implemented in a systematic framework for designing new heuristics for knowledge-lean problem-solving environments. Our system for designing new heuristics can be easily adapted to new applications and problem solvers.

In the remaining parts of this chapter, we first define and characterize what we mean by "applications," "problem solvers," and "heuristics." After defining these terminologies, we present an overview of the objective and issues in the automated design of new heuristics. We also identify the type of applications, problem solvers, and heuristic components in which we are interested. Then, we briefly outline the overall approach we have developed and how it is related to existing work, especially in the genetic algorithm area. Next, we present some examples of the applications and problem solvers targeted by our automated heuristics-design system. We then overview the chapters in this thesis. Finally, we summarize our contributions in this thesis.

1.1 Applications

An *application problem* is a specification of variables, constraints, and objectives, where constraints and objectives are functions of the variables. The variables for an application problem are divided into two classes.

(A) The first class of variables has values that are specified initially before the problem is solved. An assignment of values to these variables is defined as a *problem instance* or a *test case.*

(B) The second class of variables is the remaining unassigned variables that will have their values assigned during the problem-solving process. The assignment of these variables is related to the values specified by the given test case and the constraint and objective functions specified by the application problem. These assignments are referred to in this thesis as the *states of the application environment.* A *solution* in this case is the final assignment of values to all variables so that all performance constraints are satisfied.

A *problem domain* is defined as a collection of test cases that we wish to solve. The number of test cases in a problem domain can be large or small. We are mainly interested in applications with *infinitely many or a large number of test cases in their problem domains*, where it is not feasible to evaluate all test cases in a reasonable amount of time. This situation is common in many interesting applications.

The *quality* of a solution is the value of the objective function based on the values assigned in the solution and is a measure of how good the final assignment of values to the required variables (*i.e.*, the solution) is when the test case is solved.

The quality of a solution can be divided into two classes. First, it can be related to the appropriateness of the solution (*correctness-based*). This appropriateness is related to the satisfiability of constraints or objective functions by the solution. Second, quality can be numerical results based on substituting the solution into the objective function. In this thesis, we focus on applications that have *solutions whose qualities are measured numerically* (quantitative-based).

Some examples of application problems are shown in Table 1.1, along with the definition of problem instance, solution, constraint(s), and objective(s) for each application problem. We observe that the definitions of problem instance, solution, and constraint(s) can range from fairly simple definitions for the vertex-cover problem to highly complicated definitions for the VLSI placement and routing application. A problem instance (or a test case) in the case of VLSI placement and routing is a specification of all cells (including sizes and shapes) and all of the connections among these cells, which correspond to a VLSI circuit specification. A solution in this case is a specification of where each cell is placed on a VLSI chip and how connections among these cells are routed on the physical chip. The constraint in this case involves the timing requirements of each connection and the rule of physics that enforces the routing of connections. More details about these examples will be presented in Section 1.5.

Table 1.2 summarizes possible methods for characterizing an application problem. We defer until Section 1.5 before showing how applications from Table 1.1 are characterized. In this thesis, we focus on application problems that have large domains and have solutions whose qualities are measured numerically.

Table 1.1: Some examples of applications, problem instances, and solutions.

Application	Problem Instance	Solution	Constraint(s)	Objective(s)
Medical diagnosis [13,14]	A set of symptoms	A diagnosis of disease	A valid disease	Correctness of the diagnosis (to be maximized)
Theorems proving [14]	A set of axioms and a theorem to be proved	Indication whether the theorem is correct or incorrect	Only accepted axioms can be used in the process	Correctness of the proof and correctness of the solution (to be maximized)
Test-pattern generation for VLSI circuits [15]	The logic and gates used in a VLSI circuit	A sequence of test patterns to input into the VLSI circuit	Valid inputs to the circuit	Percentage of possible faults detectable by the test patterns (to be maximized)
VLSI placement and routing [16]	A set of cells to be placed and a set of connectivities between cells	Assignment of each cell to a location and layout of the connections	All cell placements and connectivities satisfy certain physical rules and timing requirements	Chip area (to be minimized)
Vertex-cover problem [17]	An undirected graph with certain number of vertices and the connectivities among various vertices in the graph	A subset of vertices	All vertices in the graph are connected to at least one of the vertices in the solution	The number of vertices in the solution (to be minimized)
Process mapping problem [18]	A set of communicating processes and a multicomputer system	A mapping of processes to processor	Each process is assigned to exactly one processor	Completion time of all processes based on the mapping (to be minimized)
Blind equalizer for digital channels [19]	A sequence of data corrupted by an unknown (possibly nonstationary) channel	A sequence of recovered data	Each recovered data value must be valid	Accumulated errors (to be minimized), and S/N ratio (to be maximized)

Table 1.1 (Continued)

Application	Problem Instance	Solution	Constraint(s)	Objective(s)
Load balancing in distributed systems [20]	A set of processors and their jobs and a new incoming job	Decision to place the new job	Place the new job on exactly one processor	Completion time of the new job (to be minimized)
Depth perception using stereo vision [21]	Two or more 2-D images of the same scene and camera information	3-D model of the scene	2-D projections of 3-D model must match input images	Errors in range estimation (to be minimized)

Table 1.2: Characteristics of application problems. (Boldface indicates characteristics of applications targeted by this research.)

Classification	Characteristics
Size of problem domain	• **Large number of test cases** • Small number of test cases
Quality of solution	• Related to correctness of solution • **Numerical values from objective function(s)**

1.2 What Are Problem Solvers and Heuristics

A process of finding a solution for an application is known as a *problem solver.* A problem solver interacts with the application environment (the variables unassigned according to the specification of a test case) to develop a solution for a test case. The problem solver tries to modify the state of the environment so that some desired states can be reached, *i.e.*, the constraint(s) specified in terms of the state of the environment and the test-case specification can be met. Many of these application problems are solved by making a sequence of decisions one after another. The decisions are initiated by the problem solver at decision points and are applied to change the application environment.

A problem solver in general consists of a domain-independent part and a domain-dependent part. The domain-independent part is a general solution method that is applicable across different applications. For example, a divide-and-conquer method is domain independent because it can be applied to many different applications. In contrast, the domain-dependent part is specific for a particular application. For example, the mechanism of partitioning a problem in a divide-and-conquer method is domain dependent.

The domain-independent and domain-dependent parts interact with each other to make decisions during the solution process. The domain-dependent part provides information on the current state to the domain-independent part, which returns a decision according to the information provided. The domain-dependent part then applies the decision to change the state of the application environment.

The *performance* of a problem solver on a test case depends on the *quality* of the solution found by the problem solver for this test case as well as the *cost* (*e.g.*, computation time) in finding the solution (*i.e.*, quality of the problem-solving process). Here, we define *quality* (resp., *cost*) of a solution with respect to an input test case to be one or more measures (based on the objective functions of an application and the solution values found by the problem solver) of how good the final assignment of values to the required variables is (resp., how expensive it is to reach the final assignment) when the test case is solved. The quality and cost of a solution are independent of intermediate states during the problem-solving process.

A *problem solver* can be optimal or heuristic. An optimal problem solver is a realization of an optimal algorithm that solves the problem optimally with respect to certain *objectives*. In contrast, a heuristic problem solver has components (usually domain dependent) that are designed in an ad hoc fashion, leading to possibly suboptimal solutions when applied. When there is no optimal algorithm, the design of effective heuristics is crucial. Without ambiguity, we simply use "problem solvers" in this thesis to refer to "heuristic problem solvers."

In this research, we focus on application domains that do not have abundant domain knowledge in making some of the decisions. Due to the large solution space, optimal decisions cannot be found in a reasonable amount of time. For these applications, their problem solvers must apply heuristics to make decisions in order to reach the desired final state.

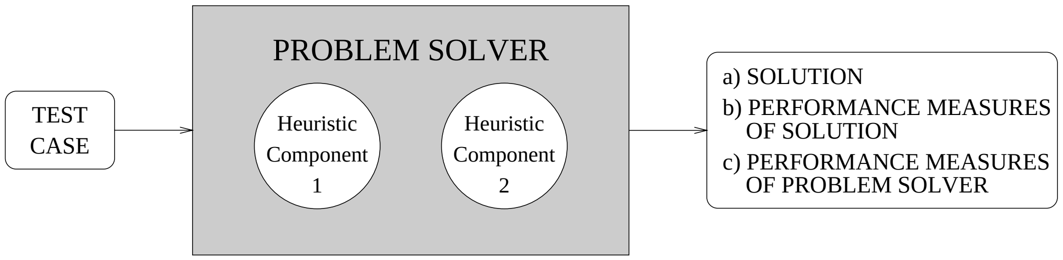

PROBLEM SOLVER
TEST CASE
Heuristic Component 1
Heuristic Component 2
a) SOLUTION
b) PERFORMANCE MEASURES OF SOLUTION
c) PERFORMANCE MEASURES OF PROBLEM SOLVER

Figure 1.1: A heuristic method applied to a test case in a knowledge-lean application domain.

Heuristics are ad hoc components within a problem solver that are used in making decisions, leading to a (suboptimal) solution for a given test case when optimal decisions are not feasible. Heuristics, in general terms, are "rules of thumb" or "commonsense knowledge" used in attempting the solution of a problem [22, 23]. Newell, Shaw, and Simon defined heuristics as "a process that may solve a given problem, but offers no guarantees of doing so [24]." Pearl defined heuristics as "strategies using readily accessible though loosely applicable information to control problem-solving processes in human beings and machines [22]."

In this thesis, we use the term *heuristic method (HM)* to denote a collection of *heuristic decision elements (HDE)* or *heuristic decision rules* applied to solve a target problem. There are many forms and representations of HDEs, including procedures, rules for making decisions, symbolic expressions, and numerical parameters. Figure 1.1 shows the interaction among test case, HM, problem solver, and solution.

The performance of an HM on a test case is the performance of the problem solver using this HM on the test case. Each HM can contribute to the *correctness* (*i.e.*, feasibility) of the solution or the *numerical performance* of the problem solver (quality of the solution found or cost of the problem-solving process). In this thesis, we are interested in developing HMs that affect the numerical performance of the problem solver. We do not attempt to deal with HDEs that may have to be proved correct before they can be used. Examples of these include complex procedures and functions.

A heuristic problem solver can be classified based on the availability of a good *world model for characterizing the relationship between its heuristics and its performance.* A problem solver with such a world model is called *knowledge rich.* When such a model is unknown, the problem solver (and its heuristics) is called *knowledge lean* and can be treated as a black-box process. It is possible for a heuristic problem solver to contain components that are knowledge rich as well as components that are knowledge lean.

There have been many studies in developing new HMs for knowledge-rich components of problem solvers. In this thesis, we have chosen to focus on developing new HMs for knowledge-lean components of problem solvers.

In summary, there are several characteristics that can be used to classify heuristic problem solvers and their HMs. First, a problem solver can be classified as *knowledge rich* or *knowledge lean* based on the availability of a good world model for relating heuristics specification to their performance. Second, an HDE in a problem solver can be classified as *correctness based*, when the choice of the HDE can affect the feasibility or correctness of the solution or *quantitative based*, when the choice of the HDE mainly affects the numerical performance of the solution or the problem-solving process. Third, a problem solver can be classified as *expensive* or *inexpensive* based on the amount of time required to solve a single test case. In this thesis we call a problem solver expensive when it takes more than 1 min to solve a test case on a workstation. We have focused on these solvers because their performance is generally hard to improve. Fourth, the set of possible choices of HMs can be *large* or *small.* We have focused on the case where there are infinitely many or a very large number of possible HMs. Cases with a small number of possible HMs can be tested exhaustively. Table 1.3 presents a summary of all of these characteristics.

Examples of problem solvers and heuristics for applications from Table 1.1 are shown in Table 1.4. As we can see, there are a wide variety of heuristics, both in terms of representations and functionalities. For example, the representations range from simply a numerical value in a process-mapping problem, to a symbolic formula for the branch-and-bound search in a vertex-cover problem, to an if-then rule for the MYCIN medical diagnosis system. More details, including the characteristics of these problem solvers and heuristics, are deferred until Section 1.5.

Table 1.3: Characteristics of problem solvers and HMs. (Boldface indicates characteristics of problem solvers and HMs targeted by this research.)

Classification	Characteristics
World model: model of relationship between HM's specification and its performance	• Knowledge rich — world model exists • **Knowledge lean — no world model**
Execution time of problem solver	• **Expensive — more than 1 min for each test case on a fast workstation** • Inexpensive — less than 1 min for each test case on a fast workstation
Quality of problem solver (cost)	• Related to the correctness of solution • **Related to numerical performance of the solver and the solution**
Size of HM space	• **Large (possibly infinite) number of possible HMs** • Small number of possible HMs

Table 1.4: Example of problem solvers and heuristics for applications in Table 1.1.

Application	Problem Solver	Domain-Dependent Heuristic Component	Heuristics Representation	Example(s)	Cost(s)
Medical diagnosis [13, 14]	MYCIN [13]: a rule-based system	When the conditions of a rule are met by the environment, then the action part of the rule is asserted	Symbolic rules	If (Symp-A) and (Symp-B) Then (Disease-A)	—
Theorems proving [14]	Resolution-based theorems prover [14]	Inference rules, deletion strategies, and ordering of operations	Symbolic rules	If (length(clause) > max) then delete(clause)	—
Test-pattern generation for VLSI circuits [15]	CRIS [15]: Search using genetic algorithm	Controls used in the genetic algorithm: iteration, rejection ratio, sequence depth, control factor, frequency of usage	Numeric values, fitness function	(2, 3, 4, 3.2, 100), H(·)	Execution time of problem solver

Table 1.4 (Continued)

Application	Problem Solver	Domain-Dependent Heuristic Component	Heuristics Representation	Example(s)	Cost(s)
VLSI placement and routing [16]	TimberWolf [16]: Simulated annealing approach	If ((acception ratio) > (threshold)), then reduce temperature to next lower value	Numerical threshold value, cost function, temperature function	0.9, C(·), T(·)	Execution time of problem solver
Vertex-cover problem [17]	Branch-and-bound search [25]	If a node has the smallest decomposition-function value among all active nodes, then expand this node	Symbolic formula	Lower bound + Upper bound of node	Number of nodes expanded in B&B search
Process mapping problem	Post-Game Analysis [18]: rule-based decision making	If (processor utilization / average utilization of all processors) > (threshold), then evict one process	Numeric threshold value	1.10	Time to find final mapping
Blind equalization for digital channels [19]	Gradient descent approach [19]	Objective (error) function for gradient descent	Symbolic formula of the error function	E(·)	Complexity of the error function, Convergence time of filter
Load balancing in distributed systems [20]	SMALL: a rule-based system	If (average WL(·) > (threshold)), then migrate this process	Workload function WL, numeric threshold value	WL(·), 2.0	Overhead of load balancing
Depth perception using stereo vision [21]	Marr and Poggio's iterative algorithm	Gaussian filter for image blurring and edge detection thresholds	(low edge-detection threshold, channel width, high threshold)	(0.6 2.0 5.0)	Time taken to discover 3-D model

Table 1.5: Characteristics of applications, problem solvers, and HMs targeted by our automated system for designing new heuristics.

Component	Characteristics
Application	• Large problem domain • Qualities of solution are numerical values of objective functions
Problem Solver	• Knowledge lean — without world model to relate heuristics specifications to their performance • Relatively expensive problem-solving process (takes more than 1 min of execution time) • Numerical values as cost of the problem-solving process
HM	• Infinitely many or very large number of possible HMs

The characteristics of the targeted applications, problem solvers, and HMs are summarized in Table 1.5. For instance, in Table 1.4, the decomposition function of a branch-and-bound search for the vertex-cover problem fits all of the characteristics presented in Table 1.5. There are infinitely many possible symbolic expressions to be used as the decomposition function. There are also infinitely many problem instances for the vertex-cover problem. The performance measures for solving the vertex-cover problem are the number of vertices in a solution and the number of branch-and-bound (B&B) nodes expanded in finding that solution. Although a small vertex-cover problem with 10-15 vertices can be solved quickly, a larger vertex-cover problem with 30-50 vertices takes more than a few minutes to solve. There is also no model to relate the specification of the decomposition function to the performance for each test case due to the complexity of the interactions in the problem-solving process. More examples are shown in Section 1.5.

1.3 Automated Design and Evaluation of New Heuristics

In this section, we discuss the main focus of this thesis: *the design of an automated system for learning new heuristics used in knowledge-lean problem solvers.* Our goal is to obtain better HM(s) for a target problem solver that produces a better solution for each test case or reduces the cost of problem solving. This goal is very desirable since many important applications require heuristics in their problem solvers (see Table 1.4 for several examples).

Heuristics are usually designed by experts with strong expertise in the target application domain or by automated learning systems using machine-learning techniques. Both methods focus on explaining the relation between heuristics and their performance and on generating "good" heuristics based on observed information or explained relations. In this research, we focus on studying "knowledge-lean" components of the problem solvers where there are no good world models of relationship between heuristics specifications and their performance. Existing work in this area [3,10–12] still has several shortcomings which we plan to address.

Our focus is on developing a systematic strategy for designing knowledge-lean heuristics that are independent of specific applications and problem solvers. We want to be able to easily adapt our heuristics-design system to any new problem solvers and applications.

In this section, we first outline the objective of a heuristics-design process and specify the characteristics of applications and problem solvers targeted by our research. Then, we outline various issues that exist in developing an automated system for designing new heuristics, especially for knowledge-lean heuristics.

1.3.1 Goal and assumptions

A *heuristics-design process* is a process for developing or *learning* new and "better" HMs for a problem solver. The underlining idea behind developing a heuristics-design process is to improve the final solutions derived by its problem solver.

In the general case, the objective of the heuristics-design process is to obtain an HM that leads to the "best" performance over the problem domain, which is usually vague and can be contradictory since we cannot define precisely what "best" performance requires. In this thesis, we define the objective of the heuristics-design process as *finding a new HM that performs "better" than an existing HM with respect to some average objective measures over the entire problem domain.*

For the scope of our research, we have previously enumerated in Sections 1.1 and 1.2 the characteristics of the application/problem solver/HM we like to study. Previously, we have also presented in Section 1.2 an overview of the possible representations of HMs to be developed. Here, we present an important assumption we have made in our research.

The major assumption we have made in our heuristics-design process is that *a problem domain can contain different regions with different statistical performance.* Under this assumption, the performance of each HM over the problem domain cannot be represented by a random subset of test cases. In addition, the performance of an HM on different subsets of test cases can be independent of one another.

It is then necessary to divide the problem domain into subsets called *subdomains* so that performance values within each subdomain can be compared. New HMs can then be designed for a particular subdomain based on a subset of test cases from that subdomain. Later in Chapter 2, we show that all performance values within each subdomain must be *independent and identically distributed (IID).*

The complications due to this assumption can be seen in the next subsection (Section 1.3.2).

1.3.2 Overview of issues

In this subsection, we enumerate all possible issues in the automated design of new HMs for a problem solver. These issues may not be directly related to the type of heuristics-design problem we are interested in. Later on, we focus on how each of these issues is affected by the characteristics of the target problem solvers/HMs we have chosen.

The various issues can be enumerated as follows:

- *Analysis of HMs* – evaluate each HM analytically.

- *Decomposition/Integration of Problem-Solver Components* – related to improving complicated problem solver in a piecewise fashion.

- *Classification of Problem Domain* – defining regions of problem domain where new HMs can be developed independently.
- *Generation of New HMs* – related to developing better HMs based on performance information obtained so far.
- *Evaluation of HMs* – related to comparisons with performance of existing HMs based on minimal evaluations.
- *Generalization of HMs Learned* – related to determining the true performance of each HM over the entire problem domain based on incomplete performance information in the design process.

We ignore the analysis issue in this thesis because analytical evaluation of HMs requires extensive domain knowledge that is not available in our problem solvers. We present an overview of each of the remaining issues in the remaining parts of this subsection.

Decomposition and integration of problem-solver components

When a problem solver has many heuristic components, it may be too complex to design new heuristics for every component simultaneously. For example, the TimberWolf [16] problem solver in Table 1.4 has many heuristic components, including a temperature-control function, a cost function, and many numerical threshold values. These components are not tightly related and may even have different forms (numerical values and symbolic functions). The number of possible combinations of these heuristics is too large to deal with efficiently.

It may be more desirable to improve a problem solver sequentially as follows.

(a) Divide the problem solver into several smaller groupings of heuristic components,

(b) Design new heuristics for each group, with possible interactions between these developments, and

(c) Integrate new heuristics from each group together to form an updated problem solver.

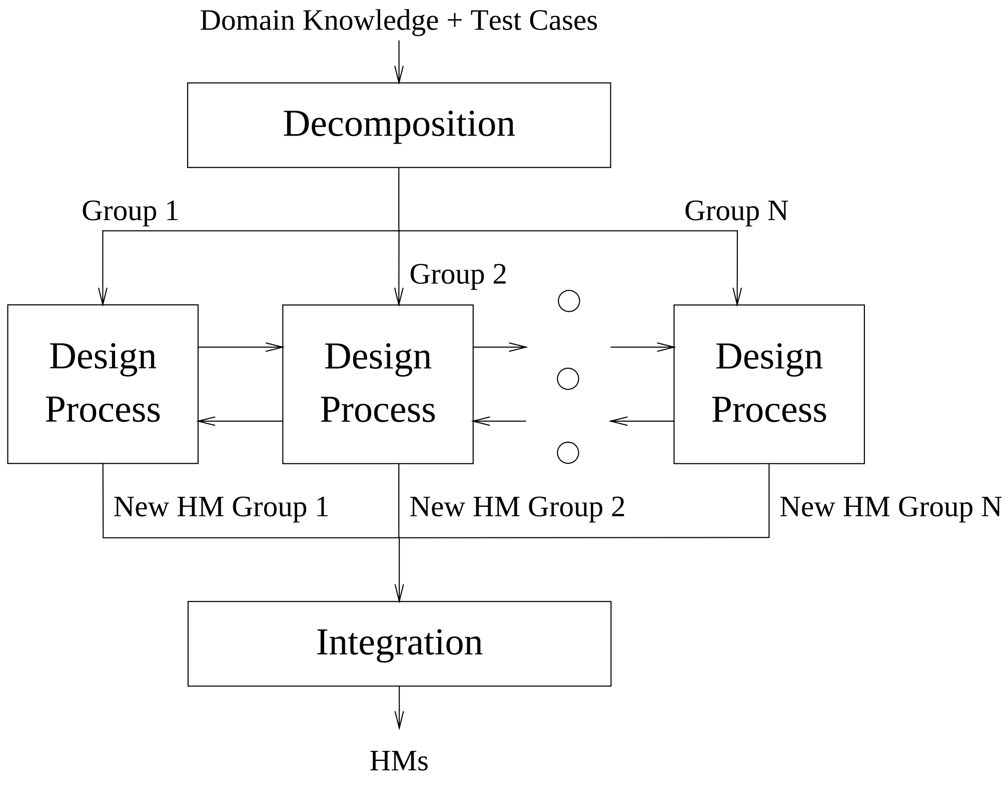


Figure 1.2: Decomposition and integration in a heuristics-design process.

This process is shown in Figure 1.2. Using the TimberWolf system as an example, the temperature-control function can be studied first before the various numerical parameters and the cost function.

There are two difficulties in the decomposition/integration process. First, some domain knowledge is required to decompose the problem solver into smaller groups. Second, the interactions between the development of new heuristics for different groups are very difficult to characterize. The simplest way is to study these groups sequentially; *i.e.*, develop H_A for group A, then develop H_B for group B with H_A in the problem solver.

This issue is not studied in this thesis as it depends on a good system for developing new heuristics for each group. Our current system assumes that any decomposition/integration is performed manually by the users.

Classification of problem domain

This issue deals with the situation when the statistical performance behavior of each HM is different for different regions of the targeted problem domain. In this case, the performance from different regions cannot be combined or compared.

This difficulty affects the selection of test cases to be used during the heuristics-design process where it is necessary to combine the performance of each HM and to compare the performance of different HMs. Obviously, randomly selecting test cases from the problem domain is not a valid approach under this condition. The problem domain must be divided into smaller subsets in which the performance from within each subset can be combined and compared. This partitioning usually requires all performance values within each subset to be *independent and identically distributed (IID).*

New HMs can then be developed for each subset of the problem domain. However, there is no guarantee that these HMs will perform well on other subsets of the problem domain. The generalization issue deals with the necessary condition for an HM to perform well over the entire problem domain. Under certain conditions, it may not be possible for a single HM to perform well over the entire problem domain and may be necessary to partition the problem domain into subspaces.

There has been some work to develop different heuristics under different situations [26]. However, most currently existing heuristics-design systems implicitly assume that performance values from the entire problem domain are IID. Consequently, these systems also assume implicitly that heuristics learned can be generalized to test cases in the entire problem domain.

In Chapter 2, we present more details related to this issue and develop a method for decomposing a problem domain into subdomains.

Generation of new HMs

The methods for generating new and better HMs are dependent on the amount of domain knowledge available. For a knowledge-rich problem solver with a good world model to relate decisions made by each HDE and performance feedbacks, a better HM can be generated based on the model through credit assignments. This approach is the focus of many studies in machine learning [1,2,4–9].

The generation process is much more difficult for knowledge-lean problem solvers where such a world model does not exist and credit assignments cannot be used. Weaker, model-free, domain-independent, and syntactic operators for generating new HMs can still be used in this case. Examples of these operators include genetics-based operators such as crossover, mutation, and hill-climbing.

An existing approach to machine learning that can take advantage of these weak operators is the *genetics-based machine-learning* approach [27,28]. This population-based approach is based on generate-and-test methods that generate new heuristics to be tested by applying operators to existing heuristics that perform well [3,10]. A pool of multiple competing HMs is maintained during the learning process. The new heuristics are potentially good as they are generated based on good ones. Examples of genetics-based machine learning include genetic programming [29] and the Pittsburgh approach to classifier system [3].

This genetics-based machine-learning approach is suitable only for problem solvers whose performance in terms of quality and cost of its solution is measured numerically. It is difficult to develop HMs for problem solvers whose performance is in terms of the correctness of the solutions. In this case, some domain knowledge is required for developing new HMs.

We study this HM generation issue in more detail in Chapter 3.

Evaluation of HMs

One of the major issues in designing new and better HMs is the ability to compare the performance of different HMs. Performance information of an HM is obtained by applying the problem solver using this HM on a set of test cases in a problem domain. We are mainly interested in this issue when performance is in the form of numerical measures.

This evaluation issue is relatively simple when the performance of each HM is deterministic. A single evaluation of each HM on one test case is sufficient to determine the exact performance of that HM.

In this thesis, we are interested in two types of application domains: (a) those with a large number of test cases and possibly an infinite number of deterministic HMs for solving them and (b) those with a small number of test cases, but the HMs concerned have a nondeterministic component, such as a random initialization point, that allows different results to be generated for each test case. In both types, the performance of an HM is nondeterministic, requiring multiple evaluations of the HM on different test cases or multiple evaluations of the HM on the same test case. Consequently, we must define valid statistical metrics for comparing two HMs without exhaustively testing all test cases using these HMs (we use sample-mean values in this thesis). This statistical comparison requires identifying subsets of test cases whose collective behavior on an HM can be evaluated statistically (see the issue on classification).

We present in Chapters 2 and 5 issues in performance evaluation of heuristics.

An important issue in implementing a heuristics-design system is the scheduling of finite computational resources for testing a possibly infinite set of test cases and infinitely many variations of HMs, which entails apportioning computational resources to tests so that the best HM is found when resources are expended. The problem is especially difficult when tests are expensive and nondeterministic. We study in Chapter 4 the scheduling of computational time in heuristics learning.

Generalization of HMs learned

When the problem domain is very large and only a small subset can be covered during the heuristics-design process, it is necessary to generalize the performance of HMs learned to test cases not studied in the design process. We want to develop HMs that perform well for the entire problem domain, not just for the test cases used during the design process.

Generalization is difficult when HMs do not perform consistently or have different performance distributions across different sets of test cases (see the issue on classification). Chapter 5 examines issues in generalization under this condition.

Existing machine-learning systems usually assume that the test cases in their test database are representative of the entire problem domain. In addition, it is implicitly assumed that performance values of all test cases are IID. In this case, the performance over the test database can represent the true performance over the entire problem domain. However, this approach does not work when there are different regions within the problem domain with entirely different performance behavior.

1.3.3 Links to previous work

In this subsection, we summarize existing work that studies various issues related to the automated design of knowledge-lean heuristics.

From the previous subsection, we observe that most existing work has been focused on the issue of generating new and better HMs (such as [1, 6, 7, 27]) with some related work in evaluation and generalization. In most cases, these two issues are studied independently of the HM generation issue. The issues on decomposition and integration and classification have been mostly ignored. This result is related to the fact that these issues appear only in more complex applications and problem solvers. In addition, it is usually necessary to address the other issues before these two issues can be dealt with effectively. This necessity is particularly true for the decomposition-and-integration issue.

In this thesis, we attempt to address some of the issues that have been ignored previously and provide a framework for integrating the solutions for each of these issues together. We have focused on classification and generalization with special attention on dealing with different statistical behavior in different regions of the problem domain. We have also attempted to provide a systematic solution to the evaluation issue. In the generation issue, we simply base our solution on existing methods since there have been extensive studies in this area.

The only issue we have neglected in this research is the decomposition-and-integration issue. This issue will be studied in the future.

1.4 General Overview of Our Heuristics-Design Approach

Given the objective, scope, and issues defined in the heuristics-design problem, we present in this subsection our overall approach. Our approach is built upon existing work in the area of machine learning, in particular, genetics-based learning. We have developed metalevel controls on existing learning methods in order to obtain better and more efficient heuristics.

Specifically, we have identified two major difficulties in learning a single general HM across many different test cases of an application. First, heuristics generally do not have the same performance statistically across test cases of a different nature, making it difficult to evaluate performance using statistical methods. Second, when tests are expensive to carry out and computational resources are limited, one needs to schedule tests to identify good heuristics and eliminate bad ones. Traditional methods generally assume that tests are inexpensive and that heuristics learned can be generalized. An example of such an existing approach is genetics-based machine learning.

In this thesis, we have used genetics-based machine learning as the underlying method for developing good HMs. To cope with the issues on generalization and resource scheduling, we have developed additional metalevel controls, resulting in an approach similar to a *metalevel genetic algorithm* (GA) [27]. This approach has been studied in the past [30] and has been found to work well in experimental prototypes [31–33].

We have developed two metalevel controls to address these issues. (See Figure 1.3.) First, we have chosen to partition the problem domain into smaller subsets called *subdomains* in which performance values of test cases in a subdomain are IID. Our metalevel strategy here is to (a) select several subdomains, (b) apply genetics-based machine learning to each selected subdomain, and (c) select the "best" HM across all tested subdomains. Second, we have developed resource scheduling strategies for controlling actions in each genetics-based learning process.

An alternative approach to selecting one common HM across all subdomains is to have multiple HMs and to decide probabilistically, when given a new test case, which HM to

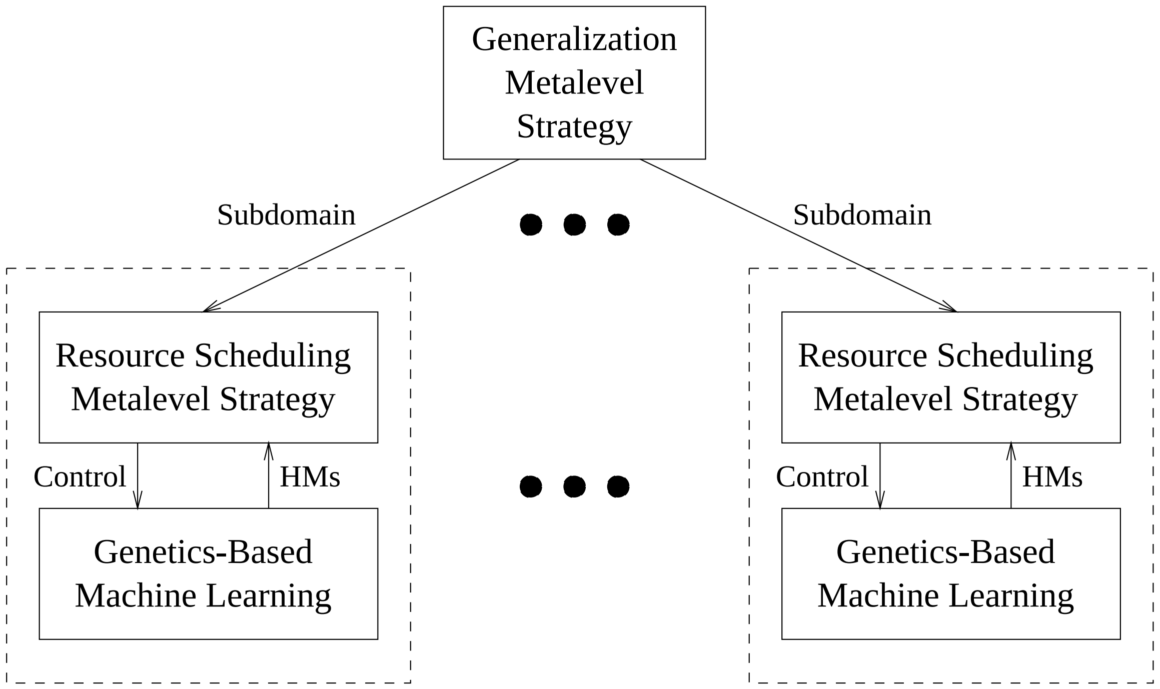

Generalization Metalevel Strategy
Subdomain
Subdomain
Resource Scheduling Metalevel Strategy
Control
HMs
Genetics-Based Machine Learning
Resource Scheduling Metalevel Strategy
Control
HMs
Genetics-Based Machine Learning

Figure 1.3: General overview of the metalevel genetic-algorithm approach used in this thesis.

apply [34]. This approach is not suitable for applications studied since relative performance of HMs will likely vary from one subdomain to another.

In Chapter 3, we present an in-depth study of our strategy and its relationship to existing work. To determine whether one HM is better than another HM, especially when they may have different statistical behavior across test cases in the problem domain, we present in Chapter 2 issues and techniques in comparing the performance of two HMs. Using these as building blocks for our learning process, we then present in Chapters 4 and 5 details of our metalevel strategies.

1.5 Example Applications and Problem Solvers

In this section, we examine several applications and their heuristic problem solvers from Table 1.4 in more detail. First, Table 1.6 shows the characteristics of heuristic problem

Table 1.6: Characteristics of heuristic problem solvers from Table 1.4.

Problem Solver	Size of Problem Domain	Performance Type	Time of Execution	World Model Availability	Size of HM Space
MYCIN for medical diagnosis	Large	Numerical (Percentage of correct diagnosis)	Inexpensive	Knowledge-Rich	Large
Resolution-based Theorems Proving	Large	Correctness	Expensive	Knowledge-Rich	Large
CRIS for generating VLSI test sequences	Large	Numerical	Expensive (especially for large circuits)	Knowledge-Lean	Large
TimberWolf for VLSI placement and routing	Large	Numerical	Expensive (especially for large circuits)	Knowledge-Lean	Large
Branch-and-bound search for vertex-cover problem	Large	Numerical	Expensive (less so for small problems)	Knowledge-Lean	Large
PGA for process-mapping	Large	Numerical	Expensive	Knowledge-Lean	Large
Blind equalization using gradient descent	Large	Numerical	Expensive (less expensive than other applications)	Knowledge-Lean	Large
SMALL for load balancing in distributed systems	Large	Numerical	Inexpensive	Knowledge-Lean	Large
Depth perception in stereo vision	Large	Numerical	Expensive	Knowledge-Lean	Large

solvers and applications from Table 1.4. Our focus is on applications and problem solvers that fit all of the characteristics that we have chosen.

From this table, we see that MYCIN and the resolution-based theorems proving systems do not fit our conditions. First, MYCIN, a rule-based system for medical diagnosis, has a good world model for relating the correctness (quality) of its diagnosis (solution) for a given set of symptoms (test case) to the rules used in selecting the diagnosis (heuristics).

Second, resolution-based theorems proving uses a set of rules (heuristics) to attempt to resolve the given set of axioms with a new theorem (a test case). The correctness (quality) of the decision whether the new theorem is correct or incorrect (solution) is an important measure of this problem solver. Consequently, heuristics used in this system must be semantically sound in order to avoid incorrect decisions. Such correctness-based problem solvers are not targeted in this research. In addition, there is a good world model available for knowledge-intensive improvement of this problem solver.

In the rest of this section, we explore in more detail several applications that fit our conditions. These are the problem solvers that we will improve using our system for designing new heuristics. For all problem solvers presented here, there are infinitely many HMs that can be selected. There is also no model for relating performance information to the specification of each HM. This condition is especially true for CRIS (a genetic-algorithm package for generating test patterns in VLSI circuit testing) where we have only information on the possible ranges of values used in CRIS and the results received after applying CRIS on a circuit.

1.5.1 CRIS and GATEST

The first two problem solvers we have focused on are two genetic-algorithm packages (CRIS [15] and GATEST [35]) for generating test patterns in VLSI circuit testing. Both packages use a domain-independent genetic algorithm [28] that continuously evolves test patterns by analyzing mutated vectors on their ability to identify (or cover) more faults in a circuit. There are many possible domain-dependent HMs; however, in our experiments, we chose the domain-dependent HMs for CRIS as a set of seven numeric parameters and for GATEST [35], one of the four fitness functions for computing the performance of test patterns.

A test case for these applications is a circuit to generate test patterns for. There are many possible circuits that can be solved by these problem solvers. In addition, both problem solvers have nondeterministic components that lead to different performance results.

In terms of performance, the main performance measure for both problem solvers is the percentage of faults covered by the discovered test patterns (solution to this application). The amount of CPU time used by both problem solvers is also important. GATEST requires more than several minutes to solve each circuit, whereas CRIS takes less time but still can require several minutes to solve larger circuits.

The results of our experiments for these two problem solvers extend the performance results we have found earlier for CRIS [36] and show the ability of our learning and generalization procedures to result in higher fault coverages than the original packages.

1.5.2 TimberWolf

The second problem solver we consider is TimberWolf (version 6) [16, 37], a software package based on simulated annealing for placing and routing a set of VLSI circuit components. There are many heuristic components in TimberWolf, including a temperature-control function, a cost function, and several numerical parameters.

A test case for this application is a set of VLSI circuit components to be mapped to a physical layout. A layout is a solution in this case. The placement of these components (cells and wires) must meet the timing requirements and the requirements of the law of physics. The quality of a solution is in terms of the chip area of the final layout. The cost of finding such a solution in terms of execution time must also be considered. A reasonably large circuit (*primary*1) can take from 5 to 15 min to solve.

For this problem solver, we illustrate that, by tuning a set of ten numeric parameters in TimberWolf, we can reduce the area of the chip as well as the time needed to find the layout.

Results on extending our results on TimberWolf's cost and temperature-control functions will be studied in the future.

1.5.3 Branch-and-bound search

The third problem solver is a software package WISE [38] that implements a branch-and-bound search to find optimal solutions for combinatorial optimization problems. We consider three combinatorial optimization problems in this thesis (vertex cover, asymmetric traveling salesman, and knapsack packing). In this case, the branch-and-bound search is domain independent, and we chose the decomposition HM as the domain-dependent heuristics. The decomposition HM is used to pick an attribute to decompose a subproblem in a search tree into descendents. The HM is represented as a symbolic formula of parameters that can be obtained in the search tree.

For instance, in a vertex-cover problem, the goal is to find the minimum number of nodes of a graph so that each edge is emanating from one of the covered nodes. In this case, a subproblem represents a set of nodes in the graph to cover partially the edges in the graph, and the decomposition HM picks the next node to be included in the covered set.

The performance measures of a branch-and-bound search are the quality of the solution found and the number of nodes that have been expanded during the problem-solving process. In this thesis, we assume that the branch-and-bound search will stop only after an optimal solution is found. Consequently, all HMs have the same quality for the same test case, and the objective is to minimize the cost (number of nodes expanded).

Because target applications are NP-complete [17], the amount of time required to solve a test case can increase exponentially with the problem size. It is very common to take several hours to find a solution for a single test case. In this thesis, we assume that relatively small test cases are used in the design process and that performance of HMs learned can generalize to larger test cases.

1.5.4 Post-game analysis

The next problem solver is the post-game analysis (PGA) system [18,39,40], a simulation-based method for mapping a set of communicating processes on a multicomputer system. This system collects an execution trace, consisting of actual execution times in between communications and amounts of data sent between processes, and uses them in a simulation system to find the actual completion time of a specific mapping. It then applies heuristics to propose a new mapping, evaluating the effectiveness of the new mapping through the simulation system. This iterative refinement is repeated until no further improvement is possible.

A test case for this application is a set of communication processes and a multicomputer system. A solution in this case is a mapping between the processes and the processors. A process must be placed on exactly one processor. The quality of a solution is in terms of the completion time of the communicating processes based on the final mapping. The cost of finding this mapping in terms of execution time is also important. A typical test case we use takes more than one minute of execution time on a Sun SparcStation 10/30. There is a trade-off between the quality and cost for this particular problem solver. Ideally, a good HM should find a mapping with low completion time while using minimal execution time. Unfortunately, an HM that uses less execution time is also likely to find a worse mapping.

There are four components of the heuristics used in PGA: (a) proposal-generation heuristics, (b) priority-assessment heuristics, (c) transformation-generation heuristics, and (d) feasibility heuristics. These heuristics are represented as expressions that combine values collected during program execution and are applied to make decisions.

1.5.5 Blind equalization

The final problem solver is a gradient descent algorithm to find a set of weights of an FIR filter [19]. The heuristic method involved is the cost function defined in the weight space for the descent algorithm. The cost function is represented by a symbolic expression. In our experiments, the cost function is defined in terms of the weights of the filter and the current output of the filter.

In this application, we define a test case as multiple random sequences of data of fixed length passing through a fixed channel and a fixed-length blind equalizer with a fixed set of random initial weights. A solution is the sequence of output data recovered from a sequence of input data corrupted in transmission.

The main performance measure in this problem solver is the number of accumulated errors (incorrect values for the recovered output sequence) for a sequence of input data corrupted in transmission. There are also other measures of interest, such as the convergence time (amount of time required before there are no more errors) and the complexity of the filter.

1.6 Outline of Thesis

This section gives a brief overview and organization of the different chapters in this thesis. The thesis is divided into three parts. In the first part, consisting of Chapter 2, a systematic examination of the issues involved in comparing the performance of two HMs is presented. In the second part, consisting of Chapters 3, 4, and 5, a systematic framework for designing knowledge-lean heuristics, TEACHER (an acronym for TEchniques for the Automated Creation of HEuRistics) [41], is described. In the third part, consisting of Chapter 6, experimental results from applications of TEACHER, our automated system for designing knowledge-lean heuristics, to develop new heuristics for several real-world applications are shown. Figure 1.4 presents an overview of this thesis.

Before a process for designing new heuristics can be developed, we must have a good understanding of how to determine if one HM is better than another HM. Hence, in Chapter 2, we first study and analyze the problem of comparing the performance of two HMs. The first issue in Chapter 2 deals with normalization of performance. Next, we present a method for dealing with multiple objective performance measures that treats each objective in an independent fashion. We then deal with partitioning the problem domain into smaller subsets called *subdomains*. This partitioning is necessary in the case where performance values from different regions of the problem domain have different behavior and must be

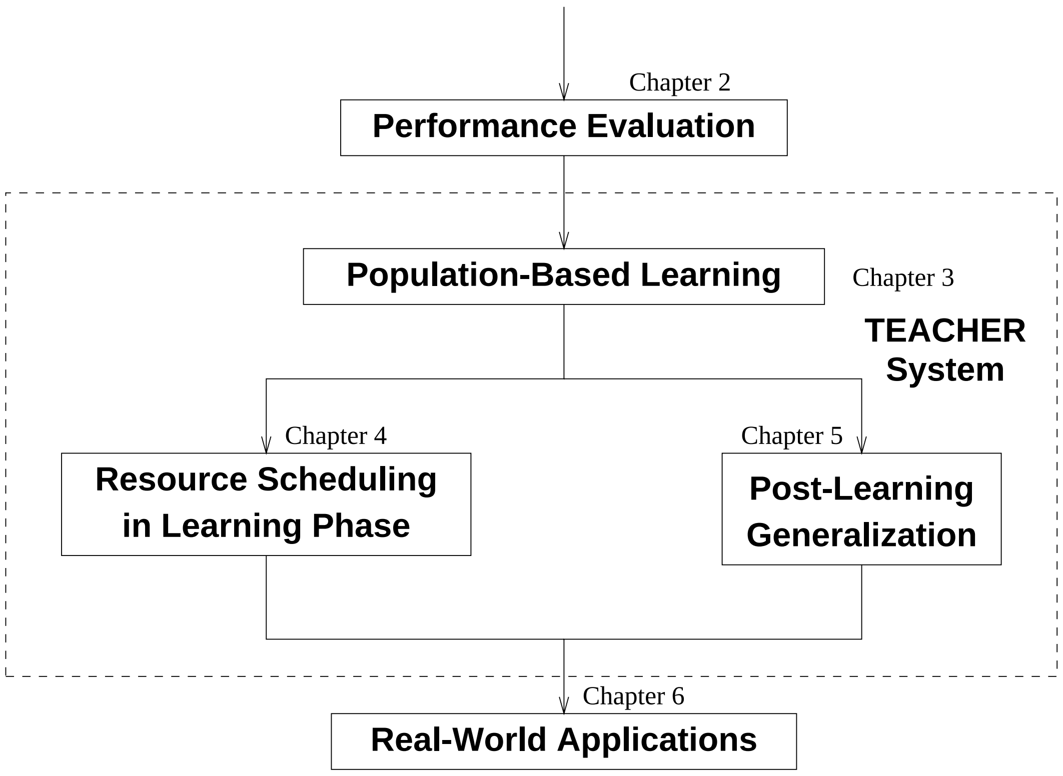


Figure 1.4: Overview and organization of this thesis.

dealt with separately. The performance-evaluation problem is then separated into two parts: statistical performance evaluation within each subdomain where performance values are IID and statistical generalization across subdomains where performance values must be dealt with separately and independently. In this chapter, we observe potential difficulties in the classification of a problem domain and in statistical performance generalization across the entire problem domain.

Chapter 3 presents a review of existing work in the automated design of knowledge-lean heuristics and an overview of our heuristics-design process that systematically deals with the issues of problem domain classification, heuristics generation, performance evaluation, and statistical performance generalization. Our system that implements this automated heuristics-design process is known as TEACHER [41].

We then proceed to study the components in TEACHER. Chapter 4 presents various strategies that we have developed for scheduling resources during each learning experiment. Chapter 5 presents our unified strategy for statistical generalization to extend each HM's performance over the entire problem domain.

In Chapter 4, we study two problems in resource scheduling during each learning experiment: sample allocation for allocating resources among active HMs and duration scheduling for deciding on a proper time to generate a new set of HMs. We present a dynamic sample-allocation strategy called *nonparametric minimum-risk* that does not require any knowledge about the distributions of performance values. We also present a dynamic duration-scheduling strategy called *DMDS* for dealing with situations in which constraints from multi-objective optimization can prune off most new HMs.

In Chapter 5, we study the *statistical generalization process* for estimating the performance of a given set of HMs over the entire problem domain, based on each HM's incomplete performance information obtained during the heuristics-design process. Our primary objective is to provide better performance than the incumbent HM. Our strategy is divided into two parts. First, the performance within each subdomain is estimated and compared with the incumbent HM. Second, the performance over the entire problem domain of each HM is estimated, based on the worst-case performance of that HM over the selected subset of subdomains used in the heuristics-design process.

Chapter 6 presents various results we have collected after applying our heuristics-design system to various real-world applications. The target problem solvers include (a) process mapping using PGA, (b) branch-and-bound search to solve several combinatorial optimization problems, (c) CRIS and GATEST for generating test patterns for VLSI circuits, (d) TimberWolf for placement and routing of VLSI circuits, and (e) blind equalization using gradient descents. Our results consistently show the effectiveness of our heuristics-design system in finding new HMs that improve in performance over existing HMs.

Finally, Chapter 7 summarizes the results we have developed in this thesis and presents some possible future directions to extend and enhance this work.

1.7 Contributions

The following are the main contributions of this thesis:

- *Identification of Issues in Automated Development of New Knowledge-Lean Heuristics (Chapters 1 and 3).* We have identified five key issues that must be addressed in the development of an automated system for designing knowledge-lean heuristics. These issues are decomposition and integration of problem-solver components, partitioning of a problem domain, generation of new heuristics, performance evaluation of heuristics, and statistical generalization of learned heuristics.

- *Development and Implementation of an Automated System for Designing Knowledge-Lean Heuristics* [36, 41] *(Chapter 3).* We have developed a systematic framework for designing new knowledge-lean heuristics. This framework has been developed to specifically deal with various key issues that we have identified. TEACHER is an implementation of our framework that has been applied to develop new heuristics for several applications. Currently, we have not extended our system to deal with the issue of decomposition and integration of problem-solver components. This extension will be carried out in the future.

- *Strategy for Performance Evaluation of Heuristics (Chapters 2 and 5).* We have identified various problems that must be addressed in the performance evaluation of heuristics and a systematic approach to address these issues. Our main contribution is the partitioning of a problem domain into smaller subsets, called *subdomains*, so that performance values within each subset can be evaluated statistically. The criteria under which the performance of an HM can be statistically estimated based on a small number of test cases is the *independent and identically distributed (IID)* property of all performance values. Performance evaluation can then be divided into two parts: statistical performance evaluation within each subdomain and statistical generalization across multiple subdomains.

  To improve statistical performance evaluation within a subdomain, normalization is applied to provide a consistent basis for comparing two HMs, and multi-objective optimization can be dealt with by transforming all but one objective into constraints. The transformation of multi-objective optimization allows each objective to be dealt with independently.

- *Strategy for Statistical Generalization of Learned HMs (Chapter 5).* When a problem domain contains multiple subdomains, performance from different subdomains must be dealt with separately and independently. The performance of each HM over the entire problem domain can then be estimated based on their *worst* performance among the set of subdomains used within the design process. This approach requires the performance values from each subdomain to be normalized with respect to an incumbent HM.

- *Resource Scheduling Strategies for Learning Under Resource Constraints* [36, 42, 43] *(Chapter 4).* We have developed a *nonparametric minimum-risk* strategy for allocating resources among active HMs without requiring any knowledge about performance distributions. We have also developed a *DMDS* strategy for dynamic scheduling of time to generate a new set of HMs. This strategy is designed to deal with the conditions in which constraints due to multi-objective optimization can eliminate most or all HMs generated.

- *Application of Learned Problem Solvers in Real-World Applications (Chapter 6).* We have applied our automated system for designing knowledge-lean heuristics to several real-world applications. Our results consistently show the effectiveness of our heuristics-design system for providing new HMs that improve over existing HMs.

2. PERFORMANCE EVALUATION OF TWO HEURISTICS

In this chapter, we study the process of performance evaluation of heuristics. The primary objective of this process is to determine, based on a fixed set of HMs and a given problem domain, the HM that outperforms other HMs with high probability over a given problem domain. We focus on the special case when there are only two HMs to evaluate and defer to Chapter 5 to address the general case of more than two HMs.

Our systematic approach to performance evaluation of two HMs has been developed to deal with two key issues: (a) ambiguities due to multiple conflicting performance objectives and (b) a large (possibly infinite) number of performance values. There are six steps in our approach to performance evaluation: (1) normalize performance values to obtain relative difference in performance between the two given HMs for each test case, (2) transform the original multiple-objective problem by applying constraints on all but one or all objectives and choose only one objective to optimize, (3) partition the problem domain into smaller problem subdomains when necessary so that each problem subdomain can satisfy the IID (independent and identically distributed) property, (4) use statistical estimation (which requires the IID property on performance values as a necessary requisite) to predict true performance over each problem subdomain based on a subset of performance values, (5) choose the HM that is better based on sample mean and probability of win (measure of certainty of the ordering of sample means) in all subdomains as the best HM for the problem domain, and (6) validate the result of previous steps by examining aspects of HM performance not studied in the previous steps.

2.1 Introduction

Before exploring the issues in performance evaluation of heuristics, we first review in this section some of the definitions defined in Chapter 1. We then define the objective in performance evaluation of two HMs and present the related issues. An overview of the strategies for addressing these issues is then presented in the final portion of this section.

2.1.1 Background

We have defined in Chapter 1 an *application problem* as a specification of variables, constraints, and objectives, where constraints and objectives are functions of the variables. The variables for an application problem are divided into two classes. The first class of variables has values that are specified initially before the problem is solved. An assignment of values to these variables is defined as a *problem instance* or a *test case*. The second class of variables is the remaining unassigned variables that will have their values assigned during the problem-solving process. A *solution* in this case is the final assignment of values to all variables in the second class so that the values of all constraint functions are satisfied based on the problem-instance specification and this assignment. The *quality* of a solution is the values of the objective functions based on the values assigned by the solution. A *problem domain* is defined as a collection of test cases that we wish to solve.

A process of finding a solution for an application is known as a *problem solver*. *Heuristics* are ad hoc components within a problem solver that are used in making decisions, leading to a (suboptimal) solution for a given test case when optimal decisions are not feasible. In this thesis, we have defined a *heuristic method* or *HM* as a collection of heuristics within a problem solver. Figure 1.1 shows the interaction among test case, HM, problem solver, and solution.

The *performance* of an HM on a test case depends on the *quality* of the solution found by the HM for this test case as well as the *cost* (*e.g.*, computation time) in finding the solution (*i.e.*, quality of the problem-solving process). Here, we define *quality* (resp., *cost*) of a solution with respect to an input test case to be one or more measures (based on the objective functions of an application and the solution values found by the problem solver) of how good the final assignment of values to the required variables is (resp., how expensive it is to reach the final assignment) when the test case is solved. The quality and cost of a solution are independent of intermediate states traversed during the problem-solving process.

In this thesis, we focus on applications that have solutions whose qualities are measured numerically and that have infinitely many or a large number of test cases in their problem domains. We have also focused on problem solvers whose qualities (*i.e.*, cost) are measured numerically. The actual numerical result(s) in evaluating an HM on a single test case is called *raw performance measure(s).* Problem solvers studied in this research can also have inherent uncertainties in their quality and cost, *i.e.*, have different solution qualities or solver costs on the same test case.

Table 2.1 shows several examples of performance measures and raw performance measures based on applications discussed in Chapter 1. For example, the main performance measure in test-pattern generation for VLSI circuits is the percentage of faults covered by a test pattern found by a problem solver (such as CRIS [15] or GATEST [35]). The objective of each HM is to maximize this measure of solution quality. In this application, the uncertainty in a problem solver can cause the raw performance measure by an HM to vary even for the same VLSI circuit.

2.1.2 Performance-evaluation objective

Our objective in performance evaluation of a pair of HMs is *to determine whether one HM is probabilistically better than or worse than the other HM over each test case in a given problem domain.* One of these HMs can be designated as the *baseline HM* in this discussion. Our goal is to determine whether the other HM is better, worse, or indeterminate with respect to the baseline HM. An important assumption under this objective is that *all test cases are equally important.*

To interpret the above objective in a more formal and mathematical form, we can designate HM_B as the baseline HM and HM_X as the other HM. Without loss of generality, we assume that there are n objective performance measures, $J_1, J_2, \ldots, J_n$, to be *maximized.*

Table 2.1: Example performance measures for several example applications. (Q refers to a quality measure, C refers to a cost measure, *Min* means that the objective is to minimize a measure, and *Max* means that the objective is to maximize a measure.)

Application	Performance Measure			
	Measure	Type	Goal	Example
Test-pattern generation for VLSI circuits	Percentage of fault coverage	Q	Max	79%, 75%
VLSI placement and routing	Area of layout	Q	Min	3431143
	Execution time	C	Min	532.5 s
Branch-and-bound search for solving a vertex-cover problem	Number of vertices required	Q	Min	5 vertices
	Number of nodes in B&B search	C	Min	153 nodes
Process mapping problem	Completion time of mapping found	Q	Min	1431.3 s
	Time to find mapping	C	Min	20.1 s
Blind equalizer for digital channels	Convergence time	Q	Min	431 bits
	Total number of accumulated errors	Q	Min	89 bits
	Signal-to-noise ratio of converged state	Q	Min	-8.43 dB
Load balancing in distributed systems	Completion time of incoming job	Q	Min	154.3 s
Stereo vision for depth perception	Error in range estimation	Q	Min	143.3 in
	Execution time	C	Min	53.2 s

We also assume that there are k test cases, $t_1, t_2, ..., t_k$, where k can be infinite. We use the symbol $P_{a,c}^{(b)}$ to denote the performance value of HM_a for objective performance measure J_b on test case t_c. When there is randomness within a problem solver, the performance of this HM on a single test case can be represented by the *average* or *mean* performance over multiple applications of the HM on this test case. Table 2.2 summarizes all performance values for the two HMs.

Based on Table 2.2, we observe that there are a large number of performance values in comparing two HMs. Since we are mainly interested in the relative performance of one HM over another on each test case, we can transform each performance value for the baseline HM, HM_B, into a fix constant, BL. This BL value is dependent only on the choice of the

Table 2.2: Raw performance values of two HMs: HM_B (the baseline HM) and HM_X.

Performance Measure		Test Case				
		t_1	t_2	$\cdots$	t_{k-1}	t_k
HM_B	J_1	$P^{(1)}_{B,1}$	$P^{(1)}_{B,2}$	$\cdots$	$P^{(1)}_{B,k-1}$	$P^{(1)}_{B,k}$
	J_2	$P^{(2)}_{B,1}$	$P^{(2)}_{B,2}$	$\cdots$	$P^{(2)}_{B,k-1}$	$P^{(2)}_{B,k}$
	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
	J_{n-1}	$P^{(n-1)}_{B,1}$	$P^{(n-1)}_{B,2}$	$\cdots$	$P^{(n-1)}_{B,k-1}$	$P^{(n-1)}_{B,k}$
	J_n	$P^{(n)}_{B,1}$	$P^{(n)}_{B,2}$	$\cdots$	$P^{(n)}_{B,k-1}$	$P^{(n)}_{B,k}$
HM_X	J_1	$P^{(1)}_{X,1}$	$P^{(1)}_{X,2}$	$\cdots$	$P^{(1)}_{X,k-1}$	$P^{(1)}_{X,k}$
	J_2	$P^{(2)}_{X,1}$	$P^{(2)}_{X,2}$	$\cdots$	$P^{(2)}_{X,k-1}$	$P^{(2)}_{X,k}$
	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
	J_{n-1}	$P^{(n-1)}_{X,1}$	$P^{(n-1)}_{X,2}$	$\cdots$	$P^{(n-1)}_{X,k-1}$	$P^{(n-1)}_{X,k}$
	J_n	$P^{(n)}_{X,1}$	$P^{(n)}_{X,2}$	$\cdots$	$P^{(n)}_{X,k-1}$	$P^{(n)}_{X,k}$

transformation method and is independent of the choice of the baseline HM. The result of the same transformation on the performance value of the other HM, HM_X can be denoted by $S^{(i)}_{X,j}$ for objective measure J_i and test case t_j (*i.e.*, corresponding to $P^{(i)}_{X,j}$). This transformation is known as *normalization* and will be discussed in more detail in Section 2.2. Table 2.3 shows a summary of all performance values after normalization. Note that normalization not only provides information about the relative performance of the two HMs but also reduces the number of performance values by a factor of two.

Based on the notations we have presented above, there are many ways to interpret the stated objective. In our research, we have further assumed that *the objective is to maximize the average performance (or population mean) across the problem domain.* This assumption implies that the magnitude of the performance for each test case is important in evaluating the performance of an HM. A major reason for choosing this assumption is that it is commonly used in evaluating HMs and problem solvers. Another advantage of this assumption is that the average metric has some nice properties that can be exploited in the performance-evaluation process.

Table 2.3: Normalized performance values for HM_B, the baseline HM, and HM_X, the HM to be compared against the baseline HM.

Performance Measure		Test Case				
		t_1	t_2	$\cdots$	t_{k-1}	t_k
HM_B	J_1	BL		$\cdots$		BL
	J_2					
	$\vdots$	$\vdots$				$\vdots$
	J_{n-1}					
	J_n	BL		$\cdots$		BL
HM_X	J_1	$S_{X,1}^{(1)}$	$S_{X,2}^{(1)}$	$\cdots$	$S_{X,k-1}^{(1)}$	$S_{X,k}^{(1)}$
	J_2	$S_{X,1}^{(2)}$	$S_{X,2}^{(2)}$	$\cdots$	$S_{X,k-1}^{(2)}$	$S_{X,k}^{(2)}$
	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
	J_{n-1}	$S_{X,1}^{(n-1)}$	$S_{X,2}^{(n-1)}$	$\cdots$	$S_{X,k-1}^{(n-1)}$	$S_{X,k}^{(n-1)}$
	J_n	$S_{X,1}^{(n)}$	$S_{X,2}^{(n)}$	$\cdots$	$S_{X,k-1}^{(n)}$	$S_{X,k}^{(n)}$

Each of the interesting properties of the average metric [44, 45] is dependent on the normalized performance values satisfying certain conditions. The first major property of the average metric is that the true average can be statistically predicted based on incomplete information when certain conditions are met. More details about this important property will be shown later in Section 2.4.2. The second property is that the *average performance* (or population mean) for performance measure J_i can approximate the *median* value for J_i when normalized performance values of each HM are symmetric [44, 46, 47].

The *median value* [44, 45] for a set of numbers is the value in which half of the numbers in the set are below it and the other half of numbers are above it. Since the median value of HM_B is fixed at BL, the median value of HM_X can indicate which HM has better normalized performance values more often. If the median of $S_{X,j}^{(i)}$ over all test cases t_j is greater than BL, then HM_X has better performance than HM_B (BL) for measure J_i in more than 50% of the test cases. This median value leads to a way to compare two HMs in which *the magnitude of the difference between their performance on a given test case is not important and only the relative ordering of the performance of each test case is important.*

Using the average performance to compare HMs, the ordering of two HMs can be determined for one objective measure. With multiple objective measures, however, there can be cases where the ordering of the two HMs cannot be determined. Let $\bar{S}_a^{(b)}$ represent the average of performance values from performance measure J_b for HM_a over k test cases. HM_X is considered better when $\bar{S}_X^{(i)} > BL$ for all measures J_i, $1 \leq i \leq n$. HM_X is considered worse when $\bar{S}_X^{(i)} < BL$ for all measures J_i. In all other cases, the ordering between the HMs cannot be determined.

In summary, under the following assumptions:

- all test cases are equally important,
- relative performance in the form of normalized performance measure is used, and
- the average metric is the criteria for comparing performance,

our objective in performance evaluation of a pair of HMs can be interpreted as follows. *The better HM is the one that has better <u>average normalized performance</u> over the problem domain for every objective performance measure.* The ordering of the two HMs cannot be determined when each HM is better than the other over a subset of performance measures.

2.1.3 Performance-evaluation issues and strategy

Based on the performance-evaluation objective presented in the previous subsection, there are two key issues that must be addressed in the performance-evaluation process [36, 48]:

1. *Multiple objective performance measures* that are inconsistent with each other (leading to indeterminate ordering of HMs), and

2. *Variation in performance values* across different test cases or even on one test case (due to uncertainty in the problem solver). This variation can lead to many (and possibly infinite) performance values for each HM to be collected.

To address these two key issues, we have developed a systematic, step-by-step strategy for performance evaluation. In the remaining sections of this chapter, we present and discuss each of the steps in our performance-evaluation process. In summary, the overall performance-evaluation strategy is as follows:

1. We first *normalize* the performance values in order to (a) eliminate dependence on absolute values in a test case and (b) simplify the process of determining relative ordering of performance values for each test case. In Section 2.2, we define this normalization process, discuss some important properties of normalization, and present several normalization methods.

2. To deal with multiple performance objectives, we propose in Section 2.3 a strategy that either constrains all objectives or all but one objective.

3. To deal with infinitely many or a large number of performance values, we use the sample-mean value to predict the true performance of each HM over the entire problem domain based on a subset of performance values. Note that performance results based on *statistical estimation* [47] generally require all performance values to be *independent and identically distributed* (IID) [47]. In particular, statistical methods must be applied to test this property. These issues are discussed in Section 2.4.

4. If the normalized performance values of an HM are not IID over the problem domain, we must partition the domain into subsets which we call *problem subdomains*. In Section 2.5 we define this concept and discuss the implications of such partitioning. Specifically, the procedure involves performance evaluation within a subdomain (Section 2.5.2) and performance evaluation across subdomains (Section 2.5.3).

5. Since we may not be able to evaluate each HM on all test cases, we must validate the performance of HMs tested. The results of these validations can either confirm or reject conclusions reached in the performance-evaluation process. For instance, performance of HMs on subdomains that are not used in the evaluation process must be examined to see if consistent conclusions can be reached (Section 2.6.1). Second, a more thorough examination of trade-offs among multiple-objective measures can be depicted graphically (Section 2.6.2).

Figure 2.1 summarizes and relates these steps together.

The performance-evaluation process studied in this chapter is a special case of the more general problem of selecting a single HM from a fixed set of HMs. The necessary extensions of the steps presented above to the general case with more than two HMs are presented in Section 3.6.4 and Chapter 5.

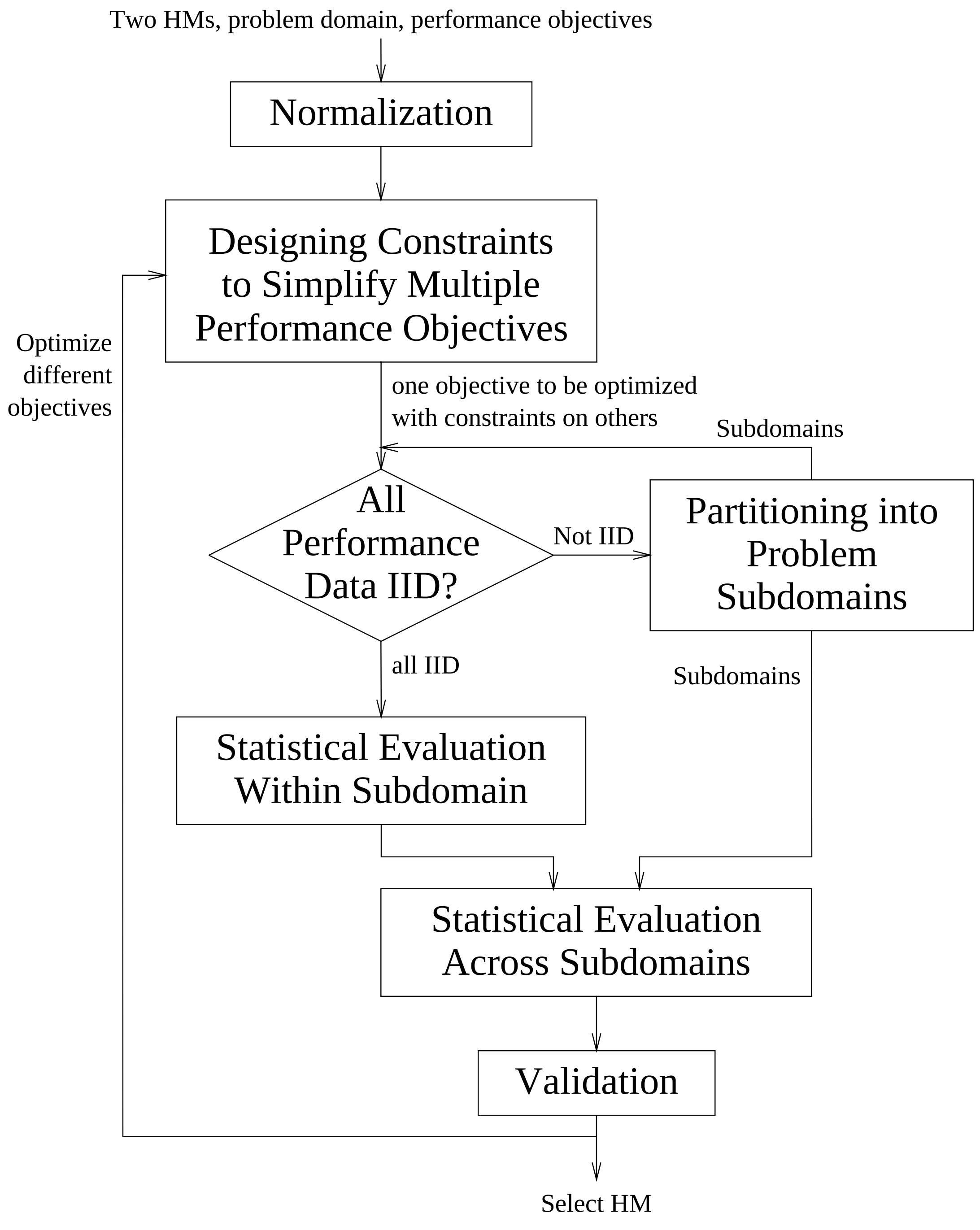
2.2 Normalization

In this section, we explore issues of *normalization* in transforming performance values in order to compare the performance of different HMs on each test case. First, we define normalization along with some examples of the process. Second, we explore several interesting characteristics of possible normalization methods. Finally, we present several symmetric-normalization methods that are useful in the evaluation process. In this section, we use the notations defined in Section 2.1.2 as well as some additional notations.

2.2.1 Definition and examples

Normalization of raw performance measures involves choosing a *baseline HM* (HM_B) and a test case and using the performance of the baseline HM as a reference point. The value resulting from normalizing a raw performance measure is called a *normalized performance measure*. The normalized performance $S_{a,j}^{(i)}$ for HM_a and objective measure J_i on test case t_j is a function of the raw performance of both HM_B and HM_a on the same test case and objective measure. In other words, $S_{a,j}^{(i)} = f(P_{a,j}^{(i)}, P_{B,j}^{(i)})$.

The simplest and most common normalization method on an objective measure is to find the ratio between the baseline performance and the raw performance. In this thesis, we refer to this normalization method as the *improvement ratio*. Using the notations in Section 2.1.2, we define $S_{a,j}^{+(i)}$, the improvement ratio of HM_a for objective measure J_i (to be maximized)



Two HMs, problem domain, performance objectives
Normalization
Designing Constraints to Simplify Multiple Performance Objectives
Optimize different objectives
one objective to be optimized with constraints on others
Subdomains
All Performance Data IID?
Not IID
Partitioning into Problem Subdomains
all IID
Subdomains
Statistical Evaluation Within Subdomain
Statistical Evaluation Across Subdomains
Validation
Select HM

Figure 2.1: The process of performance evaluation of two HMs.

on test case t_j, as follows:

$$S^{+(i)}_{a,j} = \frac{P^{(i)}_{a,j}}{P^{(i)}_{B,j}} , \tag{2.1}$$

where $P^{(i)}_{a,j}$ is the corresponding raw performance measure for HM_a, and HM_B is the baseline HM. When the goal is to minimize $P^{(i)}_{a,j}$ (such as cost or time), then the improvement ratio $S^{+(i)}_{a,j}$ is defined as $P^{(i)}_{B,j}/P^{(i)}_{a,j}$, which is sometimes known as the *speedup* measure. BL, the normalized baseline value, is always one in this normalization method regardless of the choice of the baseline HM.

This normalization process produces two main results.

(1) Since the normalized baseline performance of each test case is designed to be a fixed value, BL, it is easier to compare the performance of two HMs on a test case using a normalized performance measure. For a given objective measure and a test case, HM_X is better than HM_B when the normalized performance measure of HM_X is better than BL. The exact value of BL is dependent on the choice of the normalization method (BL is one for the improvement-ratio method).

(2) Normalization reduces the difference in *magnitude* of raw performance measures. It may lead to less variation in the normalized measures. (See Examples 2.1 and 2.2 later in this section.)

We would like to point out that normalization is appropriate in dealing with one objective measure at a time. Since the performance values from different objective measures have different meaning, it is necessary to normalize each of these measures independently. When there are multiple measures, each of them should be normalized separately using the same baseline HM. (See Table 2.3 and Example 2.7 for some examples.) This also means that each objective measure should be normalized with respect to the same measure (on the same test case) of the baseline HM.

Another point is that normalization has to be slightly modified to deal with randomness within a problem solver; *i.e.*, each HM can have different performance values even for the same test case. In this case, the performance of an HM on a test case can be represented

Table 2.4: Variations in performance values of two HMs in generating test patterns to test sequential VLSI circuits using CRIS [15] as the problem solver. The variations are due to different circuits and randomness of the problem solver.

Circuit ID	HM	Fault Coverages (%)					
		Random Seeds used in HM				Max.	Avg.
		61801	98052	15213	55414		
s298	101	78.2	74.7	79.9	81.8	81.8	78.7
	535	83.4	84.1	83.4	82.8	84.1	83.4
s444	101	60.3	13.9	11.2	60.5	60.5	36.5
	535	81.9	86.3	86.3	83.3	86.3	84.5
s1196	101	93.2	94.4	94.9	92.1	94.9	93.7
	535	93.2	92.5	93.6	92.4	93.6	92.9

by the *average* of a measure over different applications of the HM on the test case, which is demonstrated in the following example.

Example 2.1 [CRIS] Table 2.4 shows the fault coverages of two HMs in generating test patterns to test sequential VLSI circuits [15, 36]. Each HM is evaluated four times using different random seeds on three different circuits. The table shows that there are distinct differences in performance values for different circuits. The randomness in the problem solver produces variations in performance values that seem to belong to a common statistical distribution.

Using performance data in Table 2.4, Table 2.5 shows the normalized fault coverages using HM_{535} as the baseline HM. Each raw performance value is normalized using the improvement ratio with the average performance of HM_{535} on the same circuit as the baseline performance.

In this case, normalization simply changes the magnitude of performance for each circuit and does not affect differences in performance between the two HMs or variations in performance for the same circuit. However, normalization makes it easier to determine that HM_{535} is better for circuits $s444$ and $s298$ but slightly worse than HM_{101} for circuit $s1196$. ■

Table 2.5: Normalized fault coverages for CRIS [15] (normalized using the improvement ratio with respect to the average performance of HM_{535} on a circuit).

Circuit ID	HM	Normalized Fault Coverages					
		Random Seeds used in HM				Max.	Avg.
		61801	98052	15213	55414		
s298	101	0.94	0.90	0.96	0.98	0.98	0.94
	535	1.00	1.01	1.00	0.99	1.01	**1.00**
s444	101	0.71	0.16	0.13	0.72	0.72	0.43
	535	0.97	1.02	1.02	0.99	1.02	**1.00**
s1196	101	1.00	1.02	1.02	0.99	1.02	1.01
	535	1.00	1.00	1.01	0.99	1.01	**1.00**

Example 2.2 [VC] The second example is based on applying a branch-and-bound (B&B) search [49,50] to solve a vertex-cover (VC) problem [17]. The undirected graph representing each problem instance of VC is characterized by

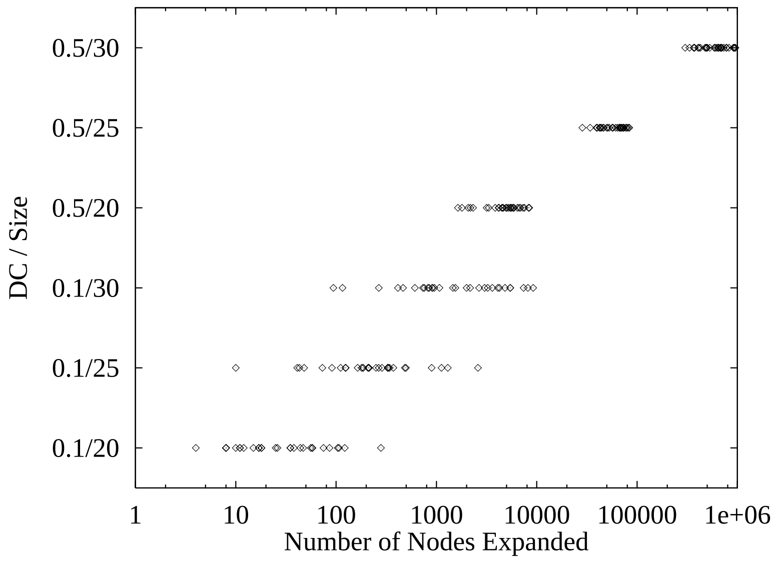
(1) the number of vertices in the graph (problem size),

(2) the degree of connectivity (*DC*) that measures the probability that an edge exists between two vertices, and

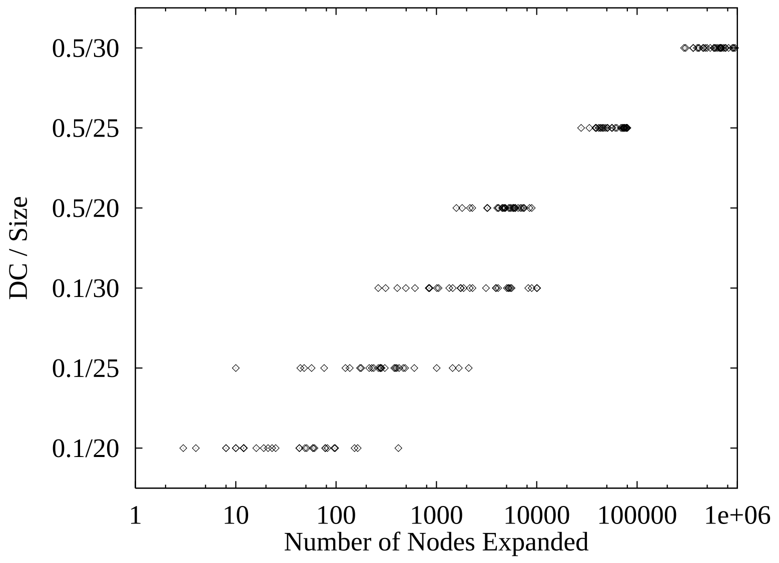
(3) the random seed used in generating a test case.

In this example, we group all test cases with the same problem size and the same DC but with different random seeds together. Figure 2.2 shows the scatter plots of two HMs that relate the number of nodes expanded with respect to DC and problem size. There are six groups of results based on two different DCs, three different problem sizes, and 30 different random seeds in each group.

From this figure, we observe that any changes in DC or problem size will lead to a large variation in the number of nodes expanded. Moreover, variations in the random seed upon which the test case is generated can lead to large (as high as two orders of magnitude) differences in the number of nodes expanded.



(a) HM_{271}



(b) HM_1

Figure 2.2: Variations in raw performance values in applying a B&B search to solve different instances of the vertex-cover problem. (In this example, performance values are not normalized.)

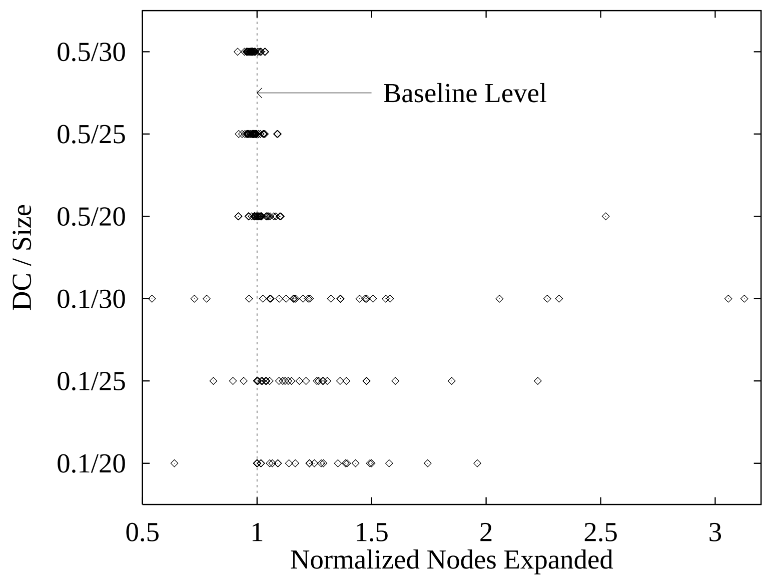


Figure 2.3: Normalized performance of HM_{271} in applying a B&B search to solve different instances of the vertex-cover problem (HM_1 is the baseline HM).

Using the performance data above, Figure 2.3 shows a scatter plot of the normalized performance for HM_{271} when HM_1 (the HM used in the existing VC application) is used as the baseline HM.

HM_{271} has fairly similar performance behavior for the same DC but with different problem sizes and random seeds. There are more variations among different problem sizes when DC = 0.1 due to some extreme values (outliers). However, there are significant differences between performance behavior for different DCs. This example shows that normalization can eliminate or reduce variations in performance with respect to two problem characteristics (problem size and random seed) but cannot deal with variations with respect to one characteristic (DC).

Based on this figure, we can also conclude that HM_{271} performs better than HM_1 for most test cases with DC = 0.1. Both HMs seem to perform similarly for test cases with DC = 0.5. Note that it would be difficult to determine this information from the raw performance values in Figure 2.2. ■

2.2.2 Characteristics of normalization methods

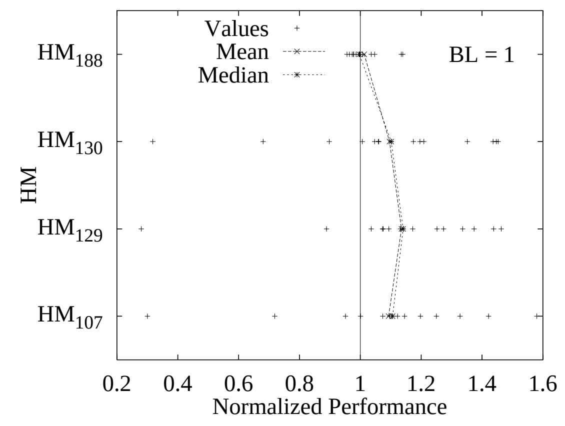
Using the normalization process defined in the last subsection, we can now explore alternative normalization methods. The improvement ratio defined earlier is just one of many possible normalization methods. In this section, we examine the similarities and differences among various normalization methods. Better understanding of these characteristics can help select appropriate normalization methods under various situations.

First, we have made some assumptions about the normalization methods under consideration based on the following notations: $P_{a,j}^{(i)}$ and $S_{a,j}^{(i)}$ represent, respectively, the raw and normalized performance measures of measure J_i for HM_a on test case t_j, and HM_B represents the baseline HM.

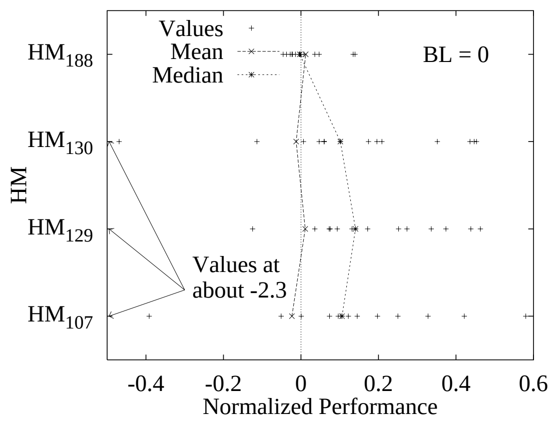
- Normalized performance $S_{a,j}^{(i)}$ is a function of only the raw performance values $P_{a,j}^{(i)}$ and $P_{B,j}^{(i)}$ and does not contain any constant values. Hence, it eliminates methods such as $S_{a,j}^{(i)} = (P_{a,j}^{(i)} - 300)/(P_{B,j}^{(i)} - 300)$ from consideration.
- For a given test case t_j and measure J_i, the HM that has the better raw performance value will have a larger normalized performance value. For example, if $P_{a,j}^{(i)}$ is better than $P_{B,j}^{(i)}$, then $S_{a,j}^{(i)} > BL(= S_{B,j}^{(i)})$. In other words, the objective is always to maximize the normalized performance.

These assumptions are made to eliminate simple variations of the same normalization method from consideration.

Based on these assumptions, the *ordering* of normalized performance values of two HMs on a given test case and a given measure will be identical regardless of the choice of the normalization method. Consequently, the orderings of the *medians* of normalized performance values obtained using different normalization methods are *always consistent* since these orderings are dependent only on the number of times that each HM is better. Moreover, the choice of a normalization method only impacts the *magnitude* of the normalized performance values.



(a) Method A



(b) Method B

Figure 2.4: Comparison of normalization results based on two different normalization methods for four pairs of HMs (VC problem) with HM_1 as the baseline HM.

We demonstrate the effects of different normalization methods in the following example.

Example 2.3 In this example, we attempt to compare four different HMs (HM_{107}, HM_{129}, HM_{130}, and HM_{188}) against the same baseline HM, HM_1, with two different normalization methods (called Method A and Method B in this example). These HMs are for solving VC problems using a B&B search similar to the ones in Example 2.2. Each HM is compared against HM_1 over the same set of 15 test cases.

In Figure 2.4, we show the results of the two different normalization methods for comparing these four HMs against HM_1. This figure shows that the ordering of the mean (average) values (when compare with BL, performance of HM_1) for HM_{107} and HM_{130} are different under different normalization methods. In both cases, HM_1 is considered better under Method B but worse under Method A. This result is mostly due to different emphasis on extremely bad performance values by different normalization methods. However, the ordering of the median values are consistent for all HMs. ■

From Example 2.3, we can observe that the *average* of normalized performance values *can have different orderings* since different normalization methods have different effects on the magnitude of normalized performance values. This anomaly is known as *performance anomalies due to different normalization methods* [36, 48, 51]. In fact, certain normalization methods can order a pair of HMs differently depending on the HM chosen as the baseline. This anomaly is known as *anomalies due to the baseline HM* [36, 48, 51]. Consider the following example.

Example 2.4 Suppose the improvement ratios of HM_1 over (baseline) HM_2 on two test cases are 10 and 0.1, respectively. Then the average improvement ratio is 5.05, which is better than the baseline level of 1, and HM_1 is considered better than HM_2

When HM_1 is used as the baseline, HM_2 also has improvement ratios of 0.1 and 10, respectively, for these two test cases. Consequently, the average improvement ratio is also 5.05, and HM_2 is considered better than HM_1.

In short, the ordering of two HMs is dependent on the choice of the baseline HM when we evaluate these HMs using improvement ratios. ■

We now explore some conditions for selecting normalization methods that would lead to a more consistent ordering of the average normalized performance values. We have identified two conditions that fit under this criteria.

Condition (A): *The ordering of two HMs (HM_1 and HM_2) based on the average normalized measure is independent of the choice of the baseline HM.*

In other words, if HM_1 is considered better than HM_2 for measure J_i when HM_2 is the baseline HM, then HM_1 is also better than HM_2 when HM_1 is the baseline HM instead. Normalization methods that satisfy this condition will not have anomalies with respect to the choice of the baseline HM.

Certain normalization methods can violate this condition if they overemphasize or de-emphasize certain ranges of relative performance. For instance, the conventional improvement-ratio (speedup) measure from Example 2.4 is biased against degradation over the baseline HM (as degradations are in the range between 0 and 1, whereas improvements are in the range between 1 and infinity).

Based on our study, we have found a sufficient condition for a normalization method to satisfy Condition (A). This condition is named the *symmetric-normalization condition* and can be stated as follows:

Symmetric-Normalization Condition: When raw performance values of HM_1 and HM_2 on test case t_a are a reversal of their values on another test case t_b (*i.e.*, $P^{(i)}_{1,a} = P^{(i)}_{2,b}$ and $P^{(i)}_{2,a} = P^{(i)}_{1,b}$), the normalized performance values for test case t_a are related to the normalized performance values for test case t_b as follows:

$$\begin{aligned} S^{(i)}_{1,a} &= 2 * BL - S^{(i)}_{1,b} , \\ S^{(i)}_{2,a} &= 2 * BL - S^{(i)}_{2,b} , \end{aligned} \tag{2.2}$$

regardless of the choice of the baseline HM. When HM_2 is the baseline HM, the second equation is equivalent to $BL = 2 * BL - BL = BL$. ∎

Next, we must prove that the symmetric condition is sufficient for a normalization method to satisfy Condition (A).

Proof. Let $S1^{(i)}_{2,j}$ denote the normalized performance value $S^{(i)}_{2,j}$ when HM_1 is the baseline HM. Similarly, let $S2^{(i)}_{1,j}$ be the normalized performance value $S^{(i)}_{1,j}$ when HM_2 is the baseline HM.

The first step is to note that when $P^{(i)}_{1,a} = P^{(i)}_{2,b}$ and $P^{(i)}_{2,a} = P^{(i)}_{1,b}$, we have the following equations based on the definition of our normalization process:

$$\begin{aligned} S1^{(i)}_{2,b} &= S2^{(i)}_{1,a} , \\ S1^{(i)}_{2,a} &= S2^{(i)}_{1,b} . \end{aligned} \tag{2.3}$$

Using Eqs. (2.2) and (2.3), we can conclude that

$$S1_{2,a}^{(i)} = 2 * BL - S1_{2,b}^{(i)} = 2 * BL - S2_{1,a}^{(i)} . \tag{2.4}$$

This equation can be applied to any test case. Note that $S2_{2,j}^{(i)} = BL = S1_{1,j}^{(i)} = 2*BL - S1_{1,j}^{(i)}$.

We can then compute $\overline{S1_2^{(i)}}$ and $\overline{S2_1^{(i)}}$, the average of normalized performance values for HM_2 with HM_1 as the baseline and for HM_1 with HM_1 as the baseline, respectively:

$$\begin{aligned} \overline{S1_2^{(i)}} &= \frac{1}{N}\sum_{j=1}^{N} S1_{2,j}^{(i)} , \\ \overline{S2_1^{(i)}} &= \frac{1}{N}\sum_{j=1}^{N} S2_{1,j}^{(i)} , \end{aligned} \tag{2.5}$$

where N is the number of test cases. We can then use Eq. (2.4) to simplify the above equation:

$$\begin{aligned} \overline{S1_2^{(i)}} &= \frac{1}{N}\sum_{j=1}^{N} \left(2 * BL - S2_{1,j}^{(i)}\right) \\ &= 2 * BL - \frac{1}{N}\sum_{j=1}^{N} S2_{1,j}^{(i)} \\ &= 2 * BL - \overline{S2_1^{(i)}} . \end{aligned} \tag{2.6}$$

Hence, if $\overline{S1_2^{(i)}} \geq BL = \overline{S1_1^{(i)}}$, then $\overline{S2_1^{(i)}} \leq 2*BL - BL = BL = \overline{S2_2^{(i)}}$. In this case, HM_2 is considered better regardless of the choice of the baseline HM. Similarly, if $\overline{S1_2^{(i)}} \leq BL$, then $\overline{S2_1^{(i)}} \geq BL$.

Based on these steps, we can conclude that if a normalization method satisfies the symmetric-normalization condition, then Condition (A) will be reached. ■

One consequence of this *symmetric-normalization condition* is that the range of normalized values that shows improvement over the baseline HM is the same size as the range of normalized values that shows degradation from the baseline HM. Normalization methods that satisfy this condition are defined as *symmetric-normalization methods.*

An existing normalization method that satisfies the symmetric-normalization condition is EWN (Equal Weight Normalization) [25]. This method normalizes all performance values between 0 and 1 with the performance of the baseline HM at 0.5. It computes the normalized performance $S_{a,j}^{EWN(i)}$ for objective measure J_i of HM_a on test case t_j using the following equation (assuming that the objective is to maximize $P_{a,j}^{(i)}$):

$$S_{a,j}^{EWN(i)} = \frac{P_{a,j}^{(i)}}{P_{a,j}^{(i)} + P_{B,j}^{(i)}}, \tag{2.7}$$

where HM_B is the baseline HM.

Example 2.5 Using the data in Example 2.4, the EWN of HM_1 with HM_2 as the baseline HM for the two test cases is 0.909 and 0.091, respectively. The average EWN is 0.5 which is the same as BL. Hence, both HMs are considered equivalent. The same result is achieved when HM_1 is used as the baseline HM. ■

Condition (B): *The averages of normalized performance values of two given HMs (HM_1 and HM_2) are approximately the same as their median values.*

We have already shown earlier in this section that orderings of HMs based on the medians of normalized performance values are always consistent. Consequently, when this condition is satisfied, the orderings of HMs based on the average normalized performance values would also be consistent and be identical to the ordering based on the medians. This condition would eliminate anomalies due to the choice of normalization method.

We have previously stated in Section 2.1.2 that the average metric can be used to estimate the median metric when normalized performance values are symmetric. So, for a given measure J_i and using HM_2 as the baseline HM, a sufficient condition for Condition (B) would be for the distribution of normalized performance values of HM_1 to be symmetric.

One simple measure indicating whether a set of data is symmetric is the *skewness* [52] of the distribution ($\gamma = E([X - \mu]^3)/\sigma^3$). This value is 0 when the data set is symmetric. Other more sophisticated tests for the symmetric condition of each data set also exist [53].

From Figure 2.4 in Example 2.3, we can see that the results of Method A are much closer to being symmetric than those of Method B. The normalized performance using Method A is still not truly symmetric, as the absolute value of its skewness varies from 0.96 to 1.58 (still smaller than for Method B which is above 2 in most cases). In this case, the mean values for Method A are also much closer to the median values than those of Method B.

This condition is much harder to satisfy than the symmetric-normalization condition. It is dependent not only on the normalization method but also on the raw performance values of HM_1 and HM_2. It should be noted that normalization methods that satisfy Condition (A) will more likely satisfy Condition (B).

In the next subsection, we present several examples of symmetric-normalization methods and discuss their differences. One of these symmetric-normalization methods should be applied when Condition (A) and/or Condition (B) are desired.

2.2.3 Symmetric-normalization methods

In the previous subsection, we have defined normalization methods that can satisfy the *symmetric-normalization condition* as *symmetric-normalization methods*. In this subsection, we present several symmetric-normalization methods. To simplify understanding, we describe all of these methods in terms of the improvement-ratio measure, S^+ (called speedup in the special case). This description is possible because all of these methods are simply monotone transformations of the improvement-ratio measure.

Symmetric improvement [36,48,54] is the normalization method we have developed specifically to satisfy the symmetric-normalization condition. We have chosen to use this normalization method as the first and most detailed example of a symmetric-normalization method. We want the symmetric-improvement measure to treat the improvement of HM_1 over HM_2 in exactly the same way as the improvement of HM_2 over HM_1, regardless of the choice of the baseline HM. The improvement ratio, S^+, treats these two ranges of relative performance differently with more focus on improvement of the non-baseline HM over the baseline HM.

We define the symmetric-improvement measure, S^{sym+}, as follows:

$$S^{sym+} = \begin{cases} S^+ - 1 & \text{if } S^+ \geq 1 \\ 1 - \dfrac{1}{S^+} & \text{if } 0 \leq S^+ < 1 \ , \end{cases} \tag{2.8}$$

where S^+ is the improvement ratio of the new HM with respect to the original baseline HM. Note that we assume that the choice of the baseline HM is the same for all normalization methods discussed in this section.

Equation (2.8) dictates that improvements and degradations carry the same weight: improvements are in the range from zero to infinity, and degradations are in the range from zero to negative infinity. The baseline level BL is 0 for this normalization method.

Next, we need to illustrate that this symmetric-improvement measure satisfies the symmetric-normalization condition. First, we assume that the raw performance values of HM_1 and HM_2 on test cases t_a and t_b are exchanged, *i.e.*, $P_{1,a}^{(i)} = P_{2,b}^{(i)}$ and $P_{2,a}^{(i)} = P_{1,b}^{(i)}$. Next, without loss of generality, we assume that HM_2 is the baseline HM.

Based on the definition of improvement ratio in Eq. (2.1),

$$S_{1,b}^{+(i)} = \frac{1}{S_{1,a}^{+(i)}} \ . \tag{2.9}$$

Note that the improvement ratio does not satisfy Eq. (2.2).

Now there are three possible cases for the value of $S_{1,a}^{+(i)}$: greater than 1, less than 1, or equal to 1 ($= BL$). We now consider the symmetric-improvement values under each of these cases. Since BL is 0, we want $S_{1,b}^{sym+(i)} = -S_{1,a}^{sym+(i)}$ in order to satisfy the symmetric-normalization condition (Eq. (2.2)).

(1) $S_{1,a}^{+(i)} > 1$, which means that $S_{1,b}^{+(i)} < 1$ based on Eq. (2.9). In this case, $S_{1,b}^{sym+(i)} = 1 - 1/S_{1,b}^{+(i)}$. We can compute $S_{1,a}^{sym+(i)}$ as follows:

$$S_{1,a}^{sym+(i)} = S_{1,a}^{+(i)} - 1 = \frac{1}{S_{1,b}^{+(i)}} - 1 = (1 - S_{1,b}^{sym+(i)}) - 1 = -S_{1,b}^{sym+(i)} \ . \tag{2.10}$$

(2) $S_{1,a}^{+(i)} < 1$, which means that $S_{1,b}^{+(i)} > 1$. In this case, $S_{1,b}^{sym+(i)} = S_{1,b}^{+(i)} - 1$. We can compute $S_{1,a}^{sym+(i)}$ as follows:

$$S_{1,a}^{sym+(i)} = 1 - \frac{1}{S_{1,a}^{+(i)}}j = 1 - S_{1,b}^{+(i)} = 1 - (S_{1,b}^{sym+(i)} + 1) = -S_{1,b}^{sym+(i)} . \quad (2.11)$$

(3) $S_{1,a}^{+(i)} = 1 = S_{1,b}^{+(i)} = BL$. In this case, $S_{1,a}^{sym+(i)} = S_{1,b}^{sym+(i)} = BL = 0$.

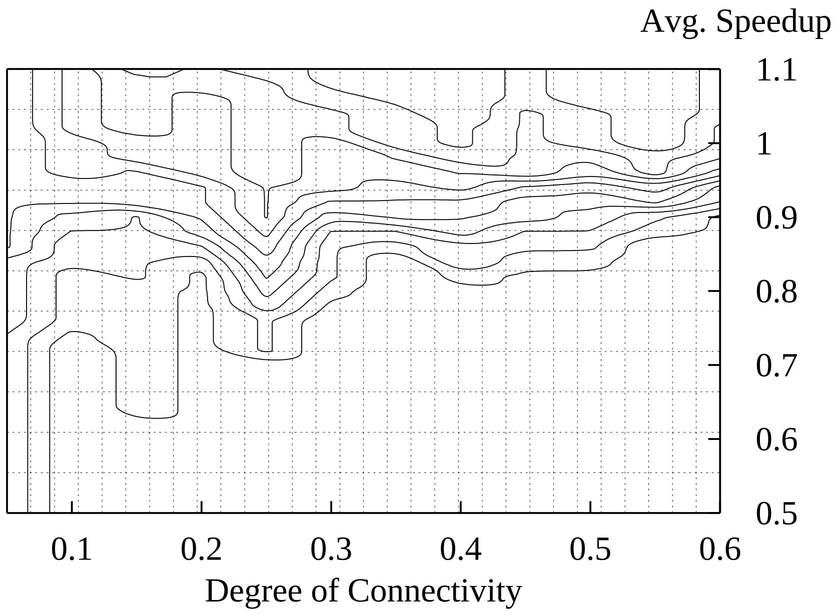
We can see that the symmetric-normalization condition is satisfied by the symmetric-improvement measure for all possible cases of $S_{1,a}^{+(i)}$ value. So this normalization method is a symmetric-normalization method. Consequently, it eliminates the anomalous condition in which the choice of the better HM can be dependent on the choice of the baseline HM.

As an example, the average symmetric improvement is zero ($= BL$) for the performance data in Example 2.4 regardless of the choice of the baseline HM, thereby avoiding the anomaly in which the average improvement ratio of both HMs is greater than BL, independent of the choice of the baseline HM.

To illustrate the difference between improvement ratios and symmetric improvements, we show in Figure 2.5 the distributions of speedups (improvement ratios) as well as symmetric speedups (symmetric improvements) of an HM to solve the vertex-cover problem with different DCs. From this figure, we observe that the contours for speedups are bunched closer to the baseline value (*i.e.*, 1 for speedups and 0 for symmetric speedups) than the contours of the symmetric speedups, which represents the compression of ranges by the speedup measure.

After exploring the symmetric-improvement method, we now proceed to present some other symmetric-normalization methods. For these methods (including the symmetric-improvement method), we mainly examine their characteristics, with special focus on the differences between various symmetric-normalization methods.

- *Symmetric improvement* (S^{sym+}). As previously stated, the range of improvement is from 0 to infinity, and the range of degradation is from 0 to negative infinity. BL is 0 for this normalization method.



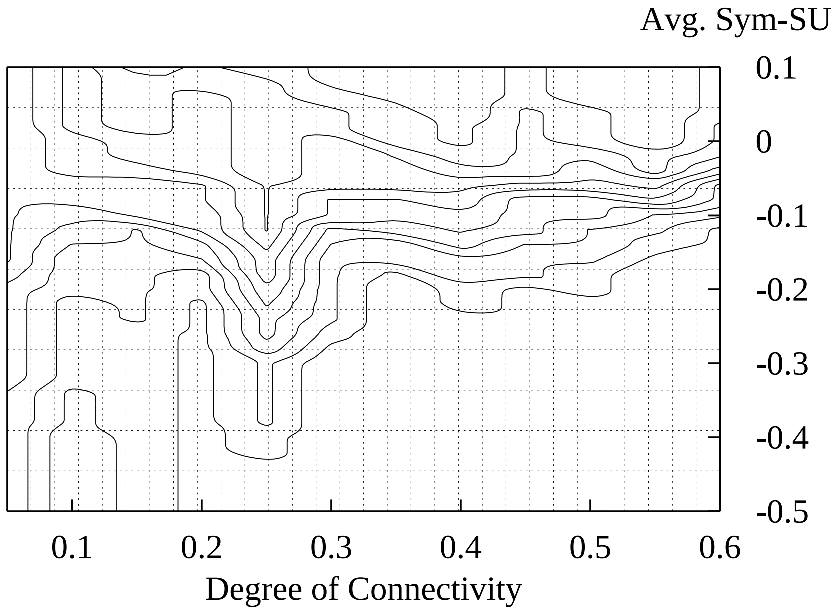


Figure 2.5: Contour plots showing the distribution of normalized performance values of one HM on 15 test cases for solving the vertex-cover problem (using speedups and symmetric speedups).

When $S^+ > 1$ (the new HM has better performance than the baseline HM), symmetric improvement, S^{sym+}, has exactly the same behavior as improvement ratio S^+ (except shifted down by 1 in each value). When $S^+ < 1$ (the new HM has worse performance than the baseline HM), this method simply creates a symmetric range of degradations. The magnitude of this degradation is exactly the same as the magnitude of the improvement if the performance values of the two HMs on that test case are reversed.

- *Delta improvement* where $S^\Delta = S^+ - 1/S^+$. The range of improvements is from 0 to infinity, whereas the range of degradations is from 0 to negative infinity. BL (the performance of the baseline HM) in this case is always 0.

  Delta improvement, S^Δ, is similar to symmetric improvement except that there is a slight expansion in regions where performance of the new HM is similar to performance of the baseline HM (S^+ close to 1).

- *Log-improvement* where $S^{log+} = \log(S^+)$. The range of improvements is from 0 to infinity, whereas the range of degradations is from 0 to negative infinity. BL in this case is always 0.

  Log-improvement, S^{log+}, puts less weight (*i.e.*, compress) on extreme values (for both improvements and degradations). It is similar to symmetric improvement when S^+ is close to 1.

- *EWN* where $S^{EWN} = S^+/(S^+ + 1)$ (see also Eq. (2.7)). The range of improvements is from 0 to 0.5, whereas the range of degradations is from 0.5 to 1. BL in this case is always 0.5.

  EWN, S^{EWN}, compresses the entire range of possible performance values to be between 0 and 1.

The difference between various symmetric-normalization methods is in the compression or decompression of different regions of improvements and degradations. This difference is illustrated by plotting the corresponding values of different symmetric-normalization methods discussed above with respect to each possible value of symmetric-improvement measure shown in Figure 2.6.

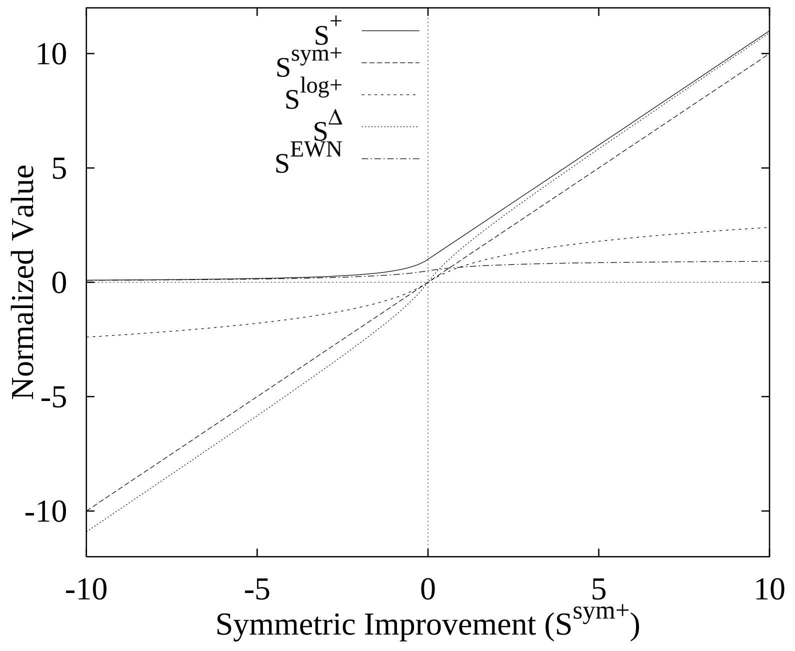


Figure 2.6: Plots of symmetric-improvement (S^{sym+}) values using different symmetric-normalization methods.

From this figure, we also observe that these symmetric-normalization methods and the improvement-ratio method are related through some monotone transformations. When two normalized values of one normalization method (say S^{+}) are ordered in a certain way (say $S_{a,j}^{+(i)} > S_{b,k}^{+(i)}$), then the corresponding normalized values of the other normalization methods (such as S^{sym+}) will have identical ordering ($S_{a,j}^{sym+(i)} > S_{b,k}^{sym+(i)}$).

These symmetric-normalization methods can differ only in the ordering of the average performance while maintaining the same ordering of performance on individual test cases. The choice of the proper symmetric-normalization method depends on user requirements and weights that users want to put on different ranges of performance. For example, if each HM can have performance values that vary widely, compressing extreme values may be desirable. An example method in this case is the log-improvement measure, S^{log+}.

2.3 Performance Evaluation with Multiple Objectives

In designing new HMs, the performance of an HM may be evaluated by multiple objectives (such as quality and cost). Of course, we would like to find HMs with improved quality and reduced cost. In an ideal case, the better HM (among the two HMs) would have better average normalized performance for every objective performance measure. However, this may not always be possible, as improved quality is often associated with increased cost. In general, one HM may be better than another on a few measures but worse on others.

In this section, we study some approaches to cope with the case when one HM is better than the baseline on a subset of the objectives, but worse on others. This problem is generally known as a *multi-objective optimization problem* [55]. The most obvious approach is to let users simply choose subjectively the HM that suits them better based on all performance information available. However, we are interested in developing a systematic approach to identify trade-offs among HMs before presenting these trade-offs to users. In addition, we like to extend the approach to the general case with more than two HMs.

There are two possible methods for resolving potential conflicts when two HMs are ordered differently based on different objective measures.

<u>Using Single Parametric Function.</u> This approach optimizes a *single* parametric function of all objective measures. Different trade-offs among the objectives are obtained by adjusting tunable parameters of the parametric function. This is the approach we have used originally [42, 43]. Our experiences have shown that anomalies may happen with this approach [51].

- The performance of an HM (as defined by the parametric objective function) may change drastically when minor changes are made on the parameters of the objective function.

  In fact, it is possible to show that one HM is better than another by finding a new parametric objective function of performance measures.

- It is difficult to translate a desirable level of trade-off into a corresponding set of parameters for the parametric objective function. We have seen similar difficulties in

Table 2.6: Inconsistent performance of PGA HMs with respect to multiple objective performance measures. ($E[c_{B,j}^{norm}] = BL^c = 0.5$ and $E[q_{B,j}^{norm}] = BL^q = 2.0$.)

Quality Measure	Test Case j				Average
	1	2	3	4	
$t_{B,j}$	1399.45	1426.59	1452.65	1343.02	1405.43
$c_{B,j}$	30.41	51.12	11.90	56.21	37.41
$t_{1,j}$	1489.26	1402.56	1415.82	1534.30	1460.48
$c_{1,j}$	9.21	8.41	12.98	2.03	8.16
$q_{1,j}^{norm}$	1.94	2.02	2.03	1.88	1.96
$c_{1,j}^{norm}$	0.77	0.86	0.48	0.97	0.77

the goal attainment method [55] to determine a good combination of parameters so that the HM with the best trade-offs can be found.

In general, given raw performance of several HMs, it is usually difficult to predict which HM will have the highest performance under a parametric performance function.

These difficulties have been observed when one attempts to use a single parametric function with adjustable parameters to evaluate the performance of HMs for a process placement application [42]. The following example illustrates these difficulties.

Example 2.6 [PGA] To demonstrate multi-objective optimization, we have chosen to use post-game analysis (PGA) [18, 39] (see Section 1.5) as a running example. PGA use HMs to map a collection of communicating processes on a network of computers. The objectives of each PGA HM are (i) to minimize the time for finding a mapping and (ii) to minimize the completion time of the mapping found.

For PGA HMs in Table 2.6, the cost $c_{i,j}$ is the time taken to find a mapping for test case j using HM_i, and $t_{i,j}$ is the corresponding completion time of the mapping found. The quality measure for test case j using HM_i, $q_{i,j}$, is the reciprocal of $t_{i,j}$. These performance measures have been normalized as follows:

$$c_{i,j}^{norm} = \frac{c_{B,j}}{c_{i,j} + c_{B,j}},$$

$$q_{i,j}^{norm} = \frac{t_{i,j} + t_{B,j}}{t_{i,j}}, \tag{2.12}$$

where HM_B is the original baseline HM for PGA [39]. These normalization equations are related to the EWN method presented in Section 2.2.2 ($c_{i,j}^{norm} = S_{i,j}^{EWN(c)}$ and $q_{i,j}^{norm} = 1/(1 - S_{i,j}^{EWN(t)})$). Based on the above definitions, $E[c_{B,j}^{norm}] = BL^c = 0.5$ and $E[q_{B,j}^{norm}] = BL^q = 2.0$.

Ideally, we want the HM selected to have high $E[c_{i,j}^{norm}]$ and high $E[q_{i,j}^{norm}]$. In this example, however, HM_B has higher $E[q_{i,j}^{norm}]$ ($E[q_{B,j}^{norm}] = 2.0 > 1.96 = E[q_{1,j}^{norm}]$), but HM_1 has higher $E[c_{i,j}^{norm}]$ ($E[c_{1,j}^{norm}] = 0.77$ as compared to $E[c_{B,j}^{norm}]$ of 0.5). Hence, we cannot find an HM that is better on all objective criterias, and some trade-offs between the two objective measures must be made.

We have previously [42] defined a family of parametric functions with an adjustable parameter T_{scale} as follows:

$$Q_{i,j}(T_{scale}) = \frac{q_{i,j}^{norm}}{0.5 + (0.5 - c_{i,j}^{norm})/T_{scale}}. \tag{2.13}$$

From Eq. (2.13) and the data in Table 2.6, suppose we want HM_1 to have the best performance. It is difficult to predict T_{scale} that will lead to this conclusion. Only by experimenting with various T_{scale} values (as shown in Table 2.7) can we find the proper T_{scale} (between 0 and 29 in this case). Table 2.7 also indicates that changing T_{scale} from 29 to 30 changes the ordering between HM_1 and HM_B. This fact cannot be predicted ahead of time. ■

Transformation of Objectives into Constraints. [36,48] To avoid inconsistencies in ordering HMs due to multiple objectives, we must evaluate the two HMs based on each individual performance measure and not combine these measures into a single parametric function [51]. This new approach applies constraints on all but one or on all objective measures and chooses

Table 2.7: Average performance of two PGA HMs (from Table 2.6) based on a single parametric performance function (from Eq. (2.13)) under different values of adjustable parameter T_{scale}.

T_{scale}	Average $Q_{i,j}(T_{scale})$	
	HM_1	HM_B
10	4.154	4.000
25	4.015	4.000
29	4.002	4.000
30	3.999	4.000
45	3.976	4.000
50	3.971	4.000

one objective to optimize (usually the remaining unconstrained objective). In this case, the better HM is the one that satisfies all constraints and has the best objective value in the measure to be optimized.

First, *all but one* or *all* objective measures should be constrained. Based on the assumptions in Section 2.2.2, we know that the objective is to maximize each normalized performance measure. For each constrained measure, the original objective of maximizing the *average* normalized performance measure can be transformed by putting a constraint on the *average* normalized performance measure instead. Each constraint level should reflect the (user selected) desirable performance level for that measure. The goal is then to optimize a single objective measure. When all but one measure are constrained (leaving one unconstrained measure), then the optimization objective is the single unconstrained measure.

An HM is considered unacceptable when one of its performance constraints is violated [36, 48]. If only one HM remains, then that HM is the better HM. If both HMs remain, then the HM with the best optimization objective is the better HM. If both HMs violate the constraints, then neither HM is acceptable. Based on this decision process, there can still be cases where the two HMs cannot be ordered. However, such cases should be less frequent as compared to the case of optimizing all the objective measures simultaneously. Notice that this method preserves the correct ordering when one HM is better for all objective measures.

Table 2.8: Examples in applying different constraints (on $E[c_{i,j}^{norm}]$ and $E[q_{i,j}^{norm}]$) on a multi-objective problem (PGA) using performance data from Table 2.6. ($BL^c = 0.5 = E[c_{B,j}^{norm}]$, $BL^q = 2.0 = E[q_{B,j}^{norm}]$, $\times^c$ and $\times^q$ denote elimination due to constraints on $E[c_{i,j}^{norm}]$ and $E[q_{i,j}^{norm}]$, respectively, and $\surd$ denotes an acceptable HM.) The optimization objective is $E[q_{i,j}^{norm}]$.

Case	Constraint Level		HM		
ID	$E[c_{i,j}^{norm}]$	$E[q_{i,j}^{norm}]$	HM_B	HM_1	HM Choose
1	$1.2BL^c = 0.6$	–	$\times^c$	$\surd$	HM_1
2	$0.8BL^c = 0.4$	–	$\surd$	$\surd$	HM_B
3	$1.6BL^c = 0.8$	–	$\times^c$	$\times^c$	none
4	$1.2BL^c = 0.6$	$BL^q = 2$	$\times^c$	$\times^q$	none
5	$1.2BL^c = 0.6$	$0.9\ BL^q = 1.8$	$\times^c$	$\surd$	HM_1
6	$0.9BL^c = 0.45$	$0.9\ BL^q = 1.8$	$\surd$	$\surd$	HM_B
7	$0.9BL^c = 0.45$	$BL^q = 2$	$\surd$	$\times^q$	HM_B

We demonstrate this approach in Example 2.7 using the performance values in Example 2.6.

Example 2.7 [PGA] Because our original objective is to optimize the normalized performance, the constraint is also on the normalized performance measure. Since the normalized performance of the baseline HM is always fixed at BL, the constraint level should be specified relative to BL. Since the constraints are relative to BL, we always know ahead of time whether the baseline HM, HM_B, can satisfy a given set of constraints. The exact value of BL is dependent on the normalization method. In this example, $BL^c = 0.5$ and $BL^q = 2.0$.

Table 2.8 shows several possible constraints that can be applied to the problem in Example 2.6 and the HM chosen based on each set of constraints.

The optimization objective in all cases is to maximize the average normalized quality $E[q_{i,j}^{norm}]$. With one acceptable HM, this HM is always considered the better HM under that set of constraints. With two acceptable HMs, the HM with the higher $E[q_{i,j}^{norm}]$ is considered the better HM. In this example, $E[q_{B,j}^{norm}] > E[q_{1,j}^{norm}]$, so HM_B is always considered the better

HM when both HMs are not pruned. With no acceptable HM, neither HM is considered better.

The first three cases (Cases 1-3) have constraints on only the average normalized cost $E[c_{i,j}^{norm}]$. These cases show the various possibilities with one acceptable HM, two acceptable HMs, and no acceptable HM. Since HM_1 has a higher average normalized cost, HM_B would always be pruned before HM_1 under this single-constraint situation. In this situation, it is easy to construct the appropriate constraint levels for any desirable conclusions.

The last four cases (Cases 4-7) have constraints on both $E[c_{i,j}^{norm}]$ and $E[q_{i,j}^{norm}]$. The different combinations of constraint levels lead to different sets of acceptable HMs (with 0, 1, or 2 acceptable HMs). Since there is more than one constraint, the set of acceptable HMs can vary more due to the interaction between different constraints.

From this example, we observe that it is easy to determine the conditions in which one HM is considered better than the other HM. ■

When more detailed trade-offs among different objective measures are desired, it is necessary to study the distributions and relationships between performance measures of each selected HM. To inspect this information for the case with two performance measures, we have proposed a multidimensional graphical representation of performance values, representing each performance measure in a separate axis (see Section 2.6.2). Two HMs are, therefore, compared based on their relative positions in this multidimensional plot.

For the general case with more than two HMs under consideration, it is necessary that the constraint levels are constant for all HMs. Consequently, the normalized performance values of all HMs for the constrained measures should be normalized against a *fixed* baseline HM. After eliminating all HMs that fail the fixed constraints, the remaining HMs can be compared based on the single objective measure. See Sections 3.6.4 and 4.4.2 and Chapter 5 for more detail.

2.4 Statistical Estimation of HM Performance

After studying issues in multiple-objective optimization, we can proceed to deal with the case in which there are too many test cases in the problem domain for each HM to be tested completely.

For accurate performance evaluation of an HM over an entire problem domain, we must obtain performance information for every test case. Based on the second issue in Section 2.1.3, this is impractical for most applications of interest due to the large problem domain or randomness within the problem solver. In this section, we first present the causes and examples of this large (and possibly infinite) number of performance values of each HM. We then discuss the potential of using statistical averages to evaluate each HM based on a subset of performance values. However, statistical estimation of performance values gives meaningful results only when the performance values satisfy a certain statistical property (independent and identically distributed — IID). Consequently, we present some existing methods for evaluating this IID property on each set of data.

2.4.1 Performance variations and their causes

In this subsection, we present an overview of the causes of different performance values for the same HM and some examples of variations in performance. First of all, different objective performance measures (such as cost and quality) will lead to different performance values. We have already presented in Section 2.3 a method that allows each objective measure to be dealt with independently. In this subsection, we are concerned with only one objective measure at a time.

For a given performance measure, there are two possible causes in variations in performance values of an HM in a problem domain.

1. *Internal characteristics of test cases.* Different test cases with different problem characteristics (such as problem size, internal structure, and random initiations) usually lead to different performance behavior. Problem characteristics and how they affect performance values are application and problem-solver dependent.

2. *Randomness inherent in the target problem solver.* For some problem solvers, there is inherent randomness in the problem-solving process. Hence, different performance levels can result when a single HM is evaluated on the same test case multiple times. For most problem solvers of this type, the randomness can be represented through the value of their initial random seed.

To demonstrate that these variations lead to a large number of distinct performance values that must be collected, we present some real-world data from several applications.

Example 2.8 [VC] In Example 2.2, we observe that any changes in DC, problem size, or even different random initiations will lead to a different (raw and normalized) number of nodes expanded (Cause 1).

A general problem domain of this application containing all possible VC problem instances would encompass a large number of DCs, problem sizes, and random problem instances. This large number of problem instances means that each HM will produce a very large (and probably an infinite) number of performance values. ■

Example 2.9 [CRIS] Based on the performance data in Example 2.1, we observe variations in performance values for the same circuit by the same HM. These variations are caused by the randomness in the problem solver (Cause 2). Since all existing sequential VLSI circuits can be considered to belong to the problem domain of this application and since there can be infinitely many variations in performance values even for the same circuit, each HM will produce infinitely many performance values over the problem domain. Gathering all of these performance values is practically impossible. ■

Based on Examples 2.8 and 2.9, we would expect that most applications will have a large number of different performance values from each HM over their problem domains. This large number of performance values on each HM is difficult to collect completely within a reasonable amount of time. Although after normalization all performance values of the baseline HM are fixed at BL, the normalized performance values of the other HM, HM_X, are still uncertain as shown in previous examples. This uncertainty is the reason why we need to use statistical estimation to predict the performance of each HM over the problem domain, based on only a relatively small subset of performance values.

2.4.2 What is statistical estimation

Statistical estimations [44–47, 56] are methods for performance evaluation based on analyzing statistical properties of performance data. When certain conditions are met, statistical estimation can be applied to a collection of data, which is a subset of a larger pool, and the result can predict (represent) the result for the larger pool.

For cases in which we are interested, the population mean (our performance objectives from Section 2.1.2) of a set of data can be statistically estimated based on the sample mean (sample average) of a subset of this data set. Another related statistical estimator is the sample standard deviation, which is an unbiased estimator of the standard deviation, a measure of the spread of performance values. These statistical estimations are possible only when all values in the data set are *independent and identically distributed (IID)* [46, 47].

Example 2.10 [VC] Based on the VC data in Example 2.2, we first renormalized the data using our symmetric-improvement method (with HM_1 still as the baseline HM) instead of the original improvement-ratio method. The population mean and population standard deviation of these new normalized measures can then be computed for each combination of DC and problem size. Table 2.9 shows these true performance indicators over each complete data set (of size 30) along with their statistical estimators (sample means and sample standard deviations) when different subsets of test cases are tested (5, 10, 15, and 20).

Based on this table, we observe that these estimators are good approximations of the true performance level in most cases. As the size of the available performance values increases, these estimators usually move closer to the true performance level. It is important to note that this is just one example of statistical estimation based on one fixed sequence of drawing performance values from each data set. Fluctuations can still happen, and one or two extreme performance values can still bring the estimators away from the true performance level, especially when the number of available performance values is small. For instance, observe the behavior of the data set with DC=0.5 and problem size of 25. ■

Table 2.9: Example of statistical estimation (sample mean and sample standard deviation) on normalized performance of HM_{271} for a vertex-cover problem. The symmetric-improvement method is used in normalization ($BL = 0$) with HM_1 as the baseline HM.

Number of Test Cases	Estimation Measure	Data Group with Given DC/Size			
		0.1/25	0.1/30	0.5/25	0.5/30
30	μ	0.219	0.507	−0.011	−0.021
	σ	0.295	0.824	0.039	0.027
5	$\hat{\mu}$	0.312	0.565	0.013	−0.024
	$\hat{\sigma}$	0.339	0.443	0.041	0.047
10	$\hat{\mu}$	0.214	0.461	0.012	−0.018
	$\hat{\sigma}$	0.258	0.491	0.051	0.039
15	$\hat{\mu}$	0.162	0.486	0.003	−0.019
	$\hat{\sigma}$	0.245	0.610	0.046	0.033
20	$\hat{\mu}$	0.162	0.494	−0.003	−0.020
	$\hat{\sigma}$	0.232	0.686	0.042	0.031

The accuracies of these estimates are dependent on both the spread of performance values (which can be estimated by the standard deviation or the variance) and the number of performance values in the subset. The accuracies increase when the spread is smaller or when more performance values are used in computing these estimates. For the average metric, there are certain statistical properties that allow the uncertainty of this estimation to be calculated under certain conditions. We will take advantage of these properties in Section 2.5.

When requirements on statistical properties of performance data are satisfied (the IID property in our case), statistical estimation can then be applied to predict the true performance of each HM over the target problem domain from a smaller subset of performance data (see Section 2.5.2). Unfortunately, the IID property may not be met throughout a problem domain. We will discuss this issue in the next subsection.

2.4.3 Test of statistical properties

In the previous subsection, we have shown that the performance of an HM can be evaluated based on a subset of performance values. However, the result of statistical estimation is meaningful only when the necessary statistical properties (IID in this case) are met by the performance values.

In this subsection, we formally define this IID property and present some statistical tests [46,47] to evaluate when this property is satisfied or violated. We are mainly interested in statistical tests that are not dependent on the exact distribution of data, *i.e.*, *nonparametric* statistical tests [47, 57]. We also present some examples on how real performance values behave in relationship to the IID property.

The statistical tests presented here are applicable when only one objective performance measure is considered. We can, however, test the IID property of each one of the multiple objective measures independently. The IID property for the case of multiple objective performance measures will hold when every individual objective measure is IID. This multiple-objective IID property is possible because our method for dealing with multiple objective measures (see Section 2.3) treats each of the objective measures (as constraints) independently.

Definition: A set of performance values is defined as having the *independent and identically distributed* (*IID*) property when it satisfies both the *identical-distribution* property and the *independence* property.

The *identical-distribution* property of a set of performance values means that each performance value in the set belongs to the same distribution.

The *independence* property means that performance values within the set are independent of each other. ■

Test of the Identical-Distribution Property: One method for testing the identical-distribution property is the Kolmogorov-Smirnov two-sample test [57, 58]. This variation of the Kolmogorov-Smirnov goodness-of-fit test [56, 57] compares sample *cdfs* (cumulative

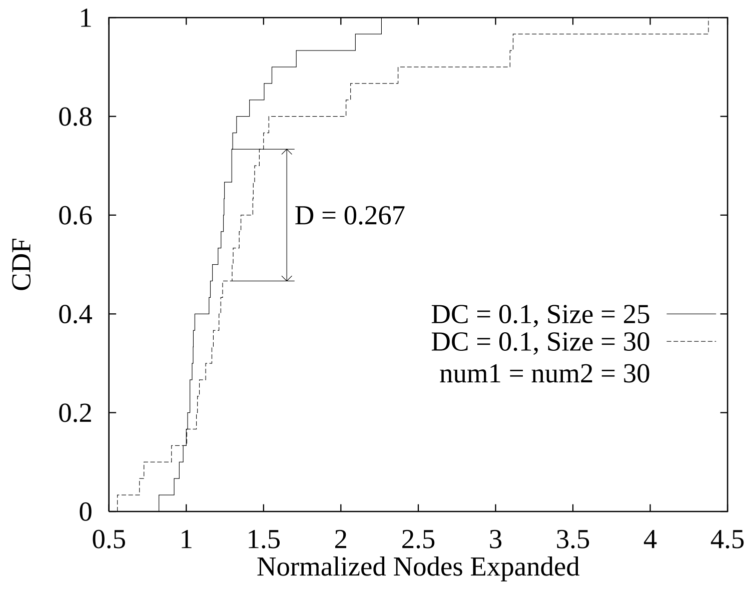


Figure 2.7: Example of the Kolmogorov-Smirnov two-sample test on the VC problem.

distribution functions [45]) of two sets of data. When the vertical difference between two sample cdfs (D) is large, then it is unlikely that the two sets of data belong to the same distribution. An example application of this test is shown in Figure 2.7.

For our purpose, we check for the identical-distribution property by dividing performance data into different subsets. We can then apply the Kolmogorov-Smirnov two-sample test on each pair of subsets. If the Kolmogorov-Smirnov test statistic D, the maximum difference between the two sample cdfs, is larger than the critical value $D^{\alpha}_{n_1,n_2}$ for data sets of size n_1 and n_2 and confidence level α, then the two sets of data are considered to be from different distributions. See Example 2.11 later in this section.

There are other statistical tests that can be applied, such as the Mann-Whitney test [46] and the Wald-Wolfowitz two-sample runs test [57]. ■

Although the independence property is likely to hold when performance values are drawn from randomly selected test cases (or randomness of the problem solver), we still need to verify that this property is satisfied.

The testing of independence is a difficult, if not impossible task [46]. We currently cannot guarantee that all performance values are independent. However, it is possible to evaluate the randomness of *a given sequence of test cases*, which is a necessary condition for the independence property.

Test of Randomness: There are many existing nonparametric methods for testing randomness of a sequence of data. These include tests based on runs-above-and-below the median [46, 47], runs-up-and-down [47, 57], and serial-correlation statistics tests [47, 59]. Other tests exist when data have specific distributions, such as the frequency test and the sequential test for uniform distribution [59].

There are four main alternative hypotheses to the null hypothesis of randomness [47]: (1) presence of shifts in the average level, (2) presence of trends in the average level, (3) presence of cyclic movements in the average level, and (4) specific parametric alternatives. The specific parametric alternatives are vague and ignored in this chapter. The various existing tests of randomness have different degrees of effectiveness against the first three causes for violations of randomness. In this section, we present several possible statistical tests that are selected to make sure that all three alternative hypotheses are covered. Other alternatives can be found in [47].

For each randomness test, we use the notation $X(1), X(2), ..., X(n)$ to denote a given sequence of n data elements.

(A) *Total Number of Runs Up and Down* [47, 57]. For a given sequence of data, we can consider the differences between consecutive values in the sequence. Replace a difference between consecutive values, $X(i+1) - X(i)$, by the symbol "+" if it is positive and by the symbol "−" if it is negative. Runs of the symbols in this sequence (length $n-1$) are referred to as runs up and down. A test statistic in this case is U, which is equal to the total number of runs (up or down). For a sequence to be considered random with significance level α, the test statistic U must satisfy $U_L^\alpha \leq U \leq U_U^\alpha$. This test is most suitable for detecting trends or cycles.

(B) *Total Number of Runs Above and Below the Median* [46,47]. A *run* in this case consists of a set of consecutive values either all greater than or all less than the median value. In this case, the number of values above the median is the same as the number of values below the median, *i.e.*, $N_1 = N_2$ ($n/2$ when n is even or $(n-1)/2$ if n is odd). A test statistic in this case is T_{median}, which is the total number of runs. For a sequence to be considered random with significance level α, the test statistic T_{median} must satisfy $T_L^\alpha \leq T_{median} \leq T_U^\alpha$, where T_L^α and T_U^α are dependent on N_1 and N_2. This test is most suitable for detecting shifts or trends.

(C) *Total Number of Runs Above and Below the Mean* [57]. This is similar to (B) except that the values are compared against the mean value rather than the median value. In this case, the number of values above the mean, N_1, is *not* necessarily the same as the number of values below the mean, N_2. A test statistic in this case is T_{mean} = total number of runs. For a sequence to be considered random with significance level α, the test statistic T_{mean} must satisfy $T_L^\alpha \leq T_{mean} \leq T_U^\alpha$. This test is also most suitable for detecting shifts or trends.

(D) *Kendall's Rank Correlation Coefficient Test* [47]. For every pair of data in a sequence $X(u)$ and $X(v)$, where $u < v$, let H_{uv} be $+1$ if $X(u) > X(v)$ and -1 otherwise. A test statistic in this case is $H = \sum_{u>v} \sum_{v=1}^{n} H_{uv}$. For a sequence to be considered random with significance level α, the test statistic H must satisfy $|H| \leq H_\alpha$. This test is most effective for detecting trends and shifts.

(E) *Circular Serial Correlation Coefficient Test* [47]. A test statistic in this case for a particular gap value k is $C_k = \sum_{i=1}^{n} X(i)X(i+k)/n$, where $X(i+k) = X(i+k-n)$ when $i+k > n$. For a sequence to be considered random with significance level α, the test statistic C_k must satisfy $|C_k| \leq C_\alpha$. The critical value C_α is dependent on the distribution of values within the given sequence. This test is most effective for detecting trends and cycles. ■

Table 2.10 summarizes the various randomness tests presented above.

We now show an example in evaluating the IID property on some real-world data using the statistical tests presented above.

Example 2.11 [VC] Using the VC data in Example 2.2, we observe that, for the VC problem, normalized performance measures on test cases with different problem sizes and/or random seeds but with the same DC seem to exhibit similar behavior. However, it is likely that test cases with different DCs have different performance distributions.

Table 2.10: Summary of statistical randomness test methods.

Statistical Test	Test Statistic	Nonrandomness Cause		
		Shift	Trend	Cycle
Total Number of Runs Up and Down	U		X	X
Total Number of Runs Above and Below Median	T_{median}	X	X	
Total Number of Runs Above and Below Mean	T_{mean}	X	X	
Kendall's Rank Correlation Coefficient Test	H	X	X	
Circular Serial Correlation Test (gap size = k)	C_k		X	X

To confirm this observation, Table 2.11 illustrates the identical-distribution test for different problem sizes and for different DCs using HM_{271}. First, this table shows that the performance for the same problem size and the same DC is likely to belong to the same distribution. This table also shows that the performance for different problem sizes but the same DC seems to belong to the same distribution. However, the performance for different DCs seems to belong to different distributions.

Next, we test for randomness in each sequence of performance values for HM_{271}. This randomness property is a prerequisite to the independence property. The results from various randomness tests are shown in Table 2.12. All sets of data pass all tests in this case and should be accepted as random and independent.

It should be noted that even if a set of data fails a few of these tests, a more detailed examination should be performed before dismissing the data as nonrandom. There are reasonable chances (at least α) that a nonrandom sequence of data is obtained even though the underlying data are random. As an example, if we move the first data in the set with DC=0.5 and problem size=25 to be the last data, then $H = -133$, which would fail to fall within the critical region for Kendall's rank correlation coefficient test.

Table 2.11: Example of the Kolmogorov-Smirnov two-sample test, with $\alpha = 0.1$ significance level, of normalized nodes expanded in a branch-and-bound search using a specific decomposition HM (HM_{271}) to solve a vertex-cover problem.

Degree of Connectivity	Graph Size	Number of Samples	D_n	D_n^α
0.1	25	15:15	0.267	0.467
0.1	30	15:15	0.200	0.467
0.5	25	15:15	0.467	0.467
0.5	30	15:15	0.267	0.467
0.1	25:30	30	0.267	0.316
0.5	25:30	30	0.233	0.316
0.1:0.5	25	30	**0.667**	0.316
0.1:0.5	30	30	**0.833**	0.316

Table 2.12: Example of randomness tests, with two-tail $\alpha = 0.1$ significance level, of normalized nodes expanded in a branch-and-bound search using a specific decomposition HM (HM_{271}) to solve a vertex-cover problem (30 values from each set of data).

Randomness Test		Thresholds		Data Group with Given DC/Size			
		Min.	Max.	0.1/25	0.1/30	0.5/25	0.5/30
Number runs up/down		15	24	17	21	18	22
Number of runs above/below median		11	21	11	19	13	21
Number of runs above/below mean		—		11	15	15	19
		Thresholds		10/21	8/17	10/21	10/21
Kendall's correlation		-92	92	-37	-21	-79	-27
Circular serial correlation	$k = 1$	—		0.02	-0.13	0.00	0.00
	$k = 2$			-0.01	0.01	0.00	0.00
	$k = 3$			-0.01	-0.01	0.00	-0.00
		Thresholds		-0.03/0.03	-0.17/0.17	-0.02/0.02	-0.02/0.02

Based on the two tables above, we can tentatively conclude, based on available data, that the performance of HM_{271} on VC problems with the same DC (with different problem sizes and different random seeds) satisfies the IID property, and statistical estimation methods can be applied. However, the IID property definitely does not exist when VC problems with different DCs are considered, and statistical estimation methods cannot be used.

When the target problem domain contains only problems with the same DC but may have different problem sizes and random seeds, then statistical estimation can be applied on this problem domain. However, a more likely scenario is for the target problem domain to contain all VC problems with all possible combinations of DCs, problem sizes, and random seeds. In this case, it is not possible to use statistical estimation over the entire problem domain. Some strategies must then be developed to overcome this problem. We will present our strategy in the next section. ■

Based on these statistical tests for the randomness and the identical-distribution properties, we can identify when statistical aggregate measures can be applied to evaluate a given set of performance data. When these two properties are satisfied, the given set of data is likely to be IID.

Results of Example 2.11 show that (a) there are subsets of performance data that can be evaluated statistically, but (b) there are also characteristics related to the test cases, such as different DCs for the VC problem, that can cause violations of the IID property. With these statistical tests, it is easier to identify exactly the test-case characteristics (and randomness of a problem solver) that can be varied without violating the IID property and the characteristics that will violate the IID property.

Since statistical estimation is a necessary step for effective performance evaluation, we must

(a) reduce variation in performance behavior as much as possible through transformation (or mapping) of performance values, and

(b) if the first step cannot eliminate violations of the IID property, partition the problem domain into smaller subsets so that statistical estimation can be applied on each subset.

Normalization (see Section 2.2) is already a good method to reduce the variation in performance behavior (see Example 2.2 for example). We can then proceed to the second step in dealing with non-IID performance data in the next section (Section 2.5).

One important note is that a monotone transformation of performance data cannot affect their IID property. This fact is important since most normalization methods are related to one another through some monotone transformations. See Section 2.2.3 for some examples. Consequently, the IID property of a set of normalized performance values is not affected by the choice of the normalization method when chosen from among methods (such as symmetric-normalization methods) that are related by monotone transformations.

2.5 Problem Subdomains

Based on information in the previous section, it is apparent that applying statistical estimation over the entire problem domain is not feasible for many applications. When statistical estimation methods cannot be applied, performance evaluation of HMs is difficult. A logical approach would then be to partition the problem domain into smaller subsets so that statistical estimation can be applied within each subset [36,48,54,60,61].

In this section, we explore the idea of defining subsets of the problem domain called "problem subdomains" so that the IID property is satisfied for each subset [54]. We then proceed to discuss the consequences of partitioning the problem domain. Based on the concept of problem subdomains, the performance-evaluation process can be divided into two separate phases: (a) performance evaluation within a single problem subdomain and (b) performance evaluation across multiple problem subdomains. Finally, we discuss the issues involved in each of these phases.

2.5.1 Definition

In this subsection, we explore the partitioning of the problem domain into subsets that can satisfy the IID property [54]. We also explore how to determine the boundary of each subset and outline the consequences of this partitioning on performance evaluation.

Definition: A *problem subdomain* is defined in this thesis as a subset of the problem domain whose performance values satisfy the IID property over the entire subset. ■

As previously stated, this IID property is a sufficient condition for using statistical estimation to estimate the average metric. Consequently, the performance of an HM over a problem subdomain can be estimated through statistical estimation over a subset of performance data from that subdomain. The next step, then, is to determine which test cases belong to a particular subdomain.

As previously stated in Section 2.4.1, there are two causes of performance variations for a given objective performance measure. In this thesis, we assume that different performance values from the same test case, *i.e.*, variations due to randomness of the problem solver (Cause 2), are IID (see Example 2.13). Obviously, if performance values from the same test case are not IID, then performance evaluation based on statistical estimation methods is not meaningful. The only alternative in this case is to evaluate HMs based on exhaustive testing of each HM.

Consequently, the partitioning of the problem domain into problem subdomains is dependent on the application-specific knowledge about test-case attributes (characteristics) (Cause 1) and how they affect the IID property of performance values of target HMs. The minimum problem subdomain must contain a single test case since performance variations due to randomness of the problem solver (if it exists) are assumed to be IID.

We must know the attributes of an application in order to classify its test cases and a set of decision rules to identify the subdomain to which a test case belongs. Statistical tests from the previous subsection can be used to determine (a) whether a given set of performance data are IID and (b) whether different sets of IID performance values (based on test cases with different attributes) belong to the same problem subdomain.

In some applications, it may be difficult to determine the subdomain to which a test case belongs. This difficulty can happen because the available attributes may not be well-defined or may be too numerous to be useful. When we do not know the attributes to classify test cases into subdomains, we can treat each test case as a subdomain by itself. This treatment works well when the problem solver has some random components. By using multiple applications of the problem solver, we can obtain statistically valid performance values of the HM on a test case (see Example 2.13). The situation is even simpler when there is no inherent randomness in the problem solver and when the performance within each subdomain is constant (no estimation required).

To illustrate the concept and methods for identifying problem subdomains, we continue the use of the vertex-cover and the test-pattern-generation problems as running examples.

Example 2.12 [VC] We have previously stated in Example 2.2 that, based on available domain knowledge, there are three main attributes for a VC test case: DC, problem size, and the random seed. From Example 2.11, we observe that normalized performance measures on test cases with different problem sizes and/or random seeds, but with the same DC, seem to be IID, while test cases with different DCs lead to different performance distributions. Similar results are obtained for different HMs on the same set of test cases.

Consequently, we can treat the degree of connectivity (DC) as an attribute to classify graphs into problem subdomains. Each problem subdomain then contains all graphs with the same DC and can have different problem sizes and different random seeds. ■

Example 2.13 [CRIS] So far, we have used two main criteria to study performance behavior of the test-pattern-generation application: circuit ID and random seed for the problem solver. To formally evaluate and test the performance of this application, we have collected additional performance data for the HMs and circuits studied in Example 2.1. Results on testing for identical distributions and randomness on the extended data sets (30 values in each set) are shown in Tables 2.13 and 2.14, respectively.

Table 2.13 shows the maximum differences between two sets of data (D_n) and the maximum acceptable differences for a significance level of 0.1 (D_n^α). Two sets of data can be considered identical if $D_n < D_n^\alpha$. Table 2.14 shows the various statistics for randomness tests along with the maximum and minimum acceptable values. The input data are considered random when their test statistics are between the minimum and the maximum acceptable values. When these maximum and minimum thresholds are the same independent of input

Table 2.13: Kolmogorov-Smirnov two-sample test, with $\alpha = 0.1$ significance level, of normalized fault coverages for the application on test-pattern generation.

HM ID	Circuit ID	Number of Samples	D_n	D_n^α
HM_{101}	s298	15:15	0.267	0.467
	s444	15:15	0.267	0.467
	s1196	15:15	0.200	0.467
	s298:s444	30	**1.000**	0.316
	s298:s1196	30	**0.633**	0.316
	s444:s1196	30	**0.874**	0.316

Table 2.14: Results of randomness tests, with two-tail $\alpha = 0.1$ significance level, of normalized fault coverages in test-pattern generation based on HM_{101} (30 values from each set of data).

Randomness Test		Thresholds		Performance Values from Circuit		
		Min.	Max.	s298	s444	s1196
Number runs up/down		15	24	20	15	17
Number of runs above/below median		11	21	16	11	14
Number of runs above/below mean		—		12	14	13
		Thresholds		10/20	8/16	10/21
Kendall's correlation		-92	92	-81	-77	41
Circular serial correlation	$k = 1$	—		0.00	-0.01	0.00
	$k = 2$			0.00	-0.01	-0.00
	$k = 3$			-0.00	-0.00	0.00
		Thresholds		-0.02/0.02	-0.02/0.02	-0.02/0.02

data, they are shown in the column titled "thresholds." When these thresholds are dependent on input data (in circular serial correlation tests and number of runs above/below mean test), they are shown in a separate row.

The results in these tables indicate that performance values from the same circuit with different random seeds are IID. Since performance from different circuits is likely to be different, we cannot automatically group all of the circuits into a single problem subdomain. There are many attributes that can be used to characterize each circuit (such as length of the longest path and the maximum number of fan-in's and fan-out's). However, we have few circuits for evaluation, and discrepancies in various attributes among available circuits are large. In addition, based on the available domain knowledge, none of these attributes is a clear winner in characterizing the circuits.

Since this particular problem solver (CRIS) has inherent randomness that leads to IID performance for a given circuit, we have chosen each circuit as an independent problem subdomain in this case. ■

After defining problem subdomains and methods to determine their boundaries, we need to discuss the effects of partitioning the problem domain into subdomains on the performance-evaluation process. With subdomain partitioning, the performance-evaluation process can now be divided into two phases: (1) performance evaluation within a single problem subdomain and (2) performance evaluation across multiple problem subdomains. We will address each of these phases separately.

2.5.2 Performance evaluation within subdomains

By the definition of a problem subdomain, the average performance within a problem subdomain can be estimated statistically based on a small subset of performance values within the problem subdomain. Two HMs can then be compared based on their sample means over a subset of normalized performance values with respect to each objective measure.

We have previously identified and addressed two of the major issues that can influence the result of performance evaluation within a problem subdomain:

(1) the normalization method used (Section 2.2), and

(2) the handling of multiple objective performance measures (Section 2.3).

The sample-mean values can then be used in place of the true average values for (a) testing constraints violation (imposed to deal with multiple objectives) and for (b) ordering the HMs based on the single optimization objective.

The main issue remaining that can affect the result of performance evaluation within a problem subdomain is the inaccuracies in the estimated sample-mean values. Since a sample mean is just an estimate of the true average value, the decisions based on a sample-mean value (such as ordering or pruning due to constraints) can be incorrect. For example, the sample means for DC/size equal to 0.5/25 with 5, 10, and 15 test cases from Example 2.10 (see Table 2.9) can mislead us to believe that HM_{271} is better than the baseline HM (HM_1) for this data set. The true performance over the entire 30 test cases indicates that HM_1 is actually better than HM_{271} for this set of data.

We first consider the issue of uncertainty in ordering two HMs based on their (estimated) sample means on the single optimization objective. Consider two HMs, HM_B and HM_X, with HM_B as the baseline HM for normalization and J_i as the optimization objective. We denote the sample mean and true average normalized performance of HM_X for measure J_i over subdomain π as $\hat{\mu}_{X,i,\pi}$ and $\mu_{X,i,\pi}$, respectively. Based on $\hat{\mu}_{X,i,\pi}$ and BL (true average performance for HM_B), we can find a tentative order between HM_X and HM_B for subdomain π. For example, if $\hat{\mu}_{X,i,\pi} > BL$, then we consider HM_X to be better than HM_B. The issue is the degree of certainty that HM_X will perform better than HM_B when more performance data are collected.

As noted in Section 2.4.2, the accuracy of statistical estimates of the true mean depends on the number of performance values used in the estimation (n) and the spread of performance values that can be measured by the (sample) standard deviation (σ or $\hat{\sigma}$). As a result, the true ordering of HMs is affected not only by the statistical estimate of their average performance but also by these two factors as well. Under certain conditions, uncertainty about the accuracy of the sample mean can be estimated.

To deal with certainty in ordering two HMs, we have developed a measure called *probability of win*, P_{win}, that takes into consideration the value of the statistical estimate, the spread of normalized performance, and the number of performance values used in the estimation. This probability ranges from 0 to 1: the closer P_{win} is to 1.0, the more certain that HM_X has better true average normalized performance than the baseline HM, HM_B. The closer P_{win} is to 0.0, the more certain that HM_X has worse true average normalized performance than the baseline HM, HM_B. When P_{win} is 0.5 (probably when $\hat{\mu}_{X,i,\pi} = BL$), HM_X can be better or worse than HM_B with equal probability.

P_{win} is defined as the probability that the true mean performance of HM_X (with respect to one performance measure) is better than the true mean performance of the baseline HM, HM_B, (fixed at BL) for one subdomain [54] (see Figure 2.8):

$$P_{win}(X, J_i, \pi) \;=\; P(\mu_{X,i,\pi} > BL \mid \hat{\mu}_{X,i,\pi}, \hat{\sigma}_{X,i,\pi}, n_{X,\pi}), \tag{2.14}$$

where $\mu_{X,i,\pi}$, $\hat{\mu}_{X,i,\pi}$, $\hat{\sigma}_{X,i,\pi}$, and $n_{X,\pi}$ are, respectively, the actual average normalized performance, the sample mean of normalized performance, the sample standard deviation of symmetric normalized performance, and the number of samples that HM_X has been tested for subdomain π and measure J_i. For simplicity, indices π and i are not shown in the rest of this subsection.

By the Central Limit Theorem [45],

$$P(\hat{\mu}_X \mid \mu_X, \sigma_X, n_X) \approx \mathcal{N}\left(\mu_X, \frac{\sigma_X^2}{n_X}\right) \;, \tag{2.15}$$

when n_X is large and where $\mathcal{N}(a, b)$ is the normal distribution function with mean a and standard deviation $\sqrt{b}$ [45]. Using this assumption, the following variable

$$T = \frac{\hat{\mu}_X - \mu_X}{\sqrt{\hat{\sigma}_X^2 / n_X}} \tag{2.16}$$

is approximately Student's t-distributed with $n_X - 1$ degrees of freedom [62]. Therefore,

$$\begin{aligned} P_{win}(X) \quad &= P(\mu_X > BL \mid \hat{\mu}_X, \hat{\sigma}_X, n_X) \\ &= P\left(T < \frac{\hat{\mu}_X - BL}{\sqrt{\hat{\sigma}_X^2 / n_X}}\right) = F_t\left(n_X - 1, \frac{\hat{\mu}_X - BL}{\sqrt{\hat{\sigma}_X^2 / n_X}}\right) \;, \end{aligned} \tag{2.17}$$

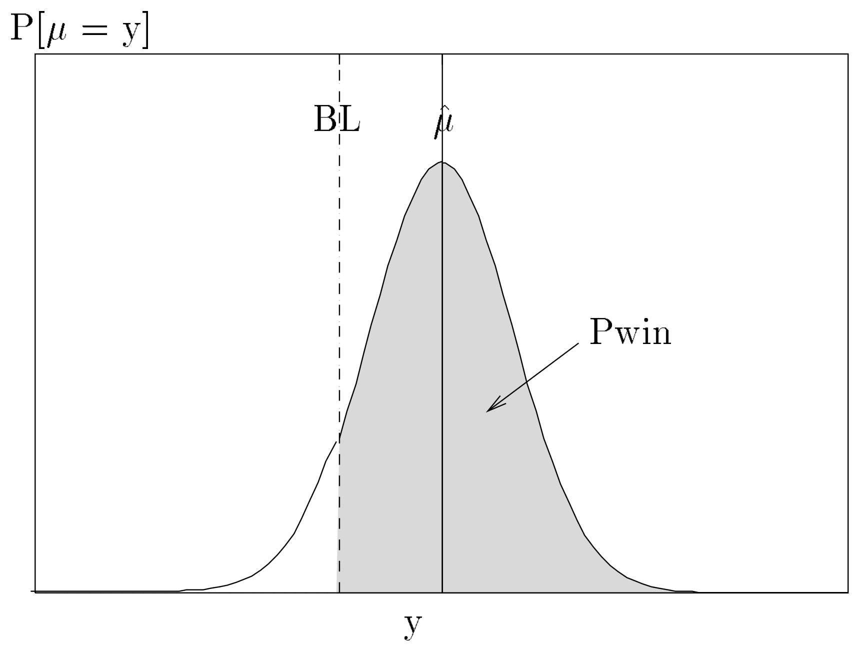


Figure 2.8: Diagram demonstrating the meaning of probability of win.

where $F_t(\nu, x)$ is the cdf of Student's t-distribution with ν degrees of freedom. When $n \to \infty$, we have

$$P(\mu_X > BL) \approx \Phi\left(\frac{\hat{\mu}_X - BL}{\sqrt{\hat{\sigma}_X^2 / n_X}}\right) , \tag{2.18}$$

where Φ is the standard cumulative normal distribution function [44].

Since $P_{win}(X) > 0.5$ when $\hat{\mu}_{X,i,\pi} > BL$ and $P_{win}(X) < 0.5$ when $\hat{\mu}_{X,i,\pi} < BL$, the decision of which HM is better for subdomain π can be based on comparing $P_{win}(X)$ with 0.5. Further, probability of win carries additional information on the certainty level of the ordering of the two HMs. A bigger deviation from 0.5 means a higher certainty level on the ordering. There is more certainty on the ordering of the two HMs when $P_{win}(X)$ is outside the range $0.5 \pm \Delta$ where $\Delta > 0$. When $P_{win}(X)$ is between $0.5 \pm \Delta$, we can consider the ordering between the two HMs to be indeterminate or uncertain. The higher Δ leads to more certainty in the final ordering but is also likely to lead to more indeterminate ordering.

Table 2.15: Examples of probability of win of HM_{271} with respect to baseline performance ($BL = 0$) for the vertex-cover problem based on data from Table 2.9.

Number of Test Cases	Degree of Connectivity/# of Nodes			
	0.1/25	0.1/30	0.5/25	0.5/30
30	1.000	0.999	0.067	0.000
5	0.946	0.977	0.741	0.159
10	0.986	0.992	0.762	0.089
15	0.989	0.996	0.598	0.021
20	0.997	0.998	0.376	0.005

Example 2.14 [VC] Based on the sample-mean and sample-standard-deviation values from Example 2.10 (see Table 2.9), we can compute the probability of win for each possible case using Eq. (2.17). The resulting probability-of-win values are shown in Table 2.15.

From this table, we can observe that the probability-of-win values in most cases (except for DC=0.5 and problem size of 25) demonstrate a very high degree of certainty (close to 0 or 1). For these cases, the orderings based on the sample mean (and P_{win}) also happen to be correct.

For DC=0.5 and problem size of 25, however, the probability-of-win values indicate a lower degree of certainty (closer to 0.5). This lower degree of certainty indicates a high probability that the sample-mean values will give an incorrect ordering. In actuality, the ordering based on the sample means for this data set with 5, 10, and 15 test cases is incorrect. For $\Delta \approx 0.265$, these incorrect orderings would be rejected due to an insufficient degree of certainty, while the orderings from other data sets (which are correct) would still be accepted. However, the (correct) ordering for DC=0.5 and problem size of 25 with 20 test cases would also be rejected under this criteria. ■

In the multiple-objective case, a similar uncertainty is involved in deciding for each constrained performance measure whether the true average is larger than the given constraint level [36]. The *probability of satisfying a given set of constraints*, P_{ok}, can be used to estimate the likelihood that the constraint is satisfied, based on given performance values [36]. The computation of P_{ok} follows the same reasoning as the reasoning for P_{win} above.

For each constrained measure, J_a, the probability that the true mean of that normalized measure is higher than the minimum constraint level, θ_a, for HM_X in subdomain π can be computed as follows:

$$P(\mu_{X,a,\pi} \geq \theta_a) \;=\; F_t\left(n_{X,\pi} - 1, \frac{\hat{\mu}_{X,a,\pi} - \theta_a}{\sqrt{\hat{\sigma}^2_{X,a,\pi}/n_{X,\pi}}}\right) \;, \tag{2.19}$$

where $F_t(\nu, x)$ is the cdf of Student's t-distribution with ν degrees of freedom.

When there are multiple constrained measures, the probability that all constraints (θ_j for $j = 1, \ldots, k$) are satisfied is equal to $P(\mu_{X,1,\pi} \geq \theta_1 \cap \ldots \cap \mu_{X,k,\pi} \geq \theta_k)$. Based on probability theory, we know that

$$\prod_j P(\mu_{X,j,\pi} \geq \theta_j) \leq P_{ok} = P\left(\mu_{X,1,\pi} \geq \theta_1 \cap \ldots \cap \mu_{X,k,\pi} \geq \theta_k\right) \leq \min_j P\left(\mu_{X,j,\pi} \geq \theta_j\right). \tag{2.20}$$

Hence, we use $\min_j P(\mu_{X,j,\pi} \geq \theta_j)$ as an approximation to P_{ok}.

Similar to the case with P_{win}, an HM is considered to satisfy a given set of constraints when $P_{ok} > 0.5$. However, when P_{ok} is in the vicinity of $0.5 \pm \Delta$ for some $\Delta > 0$, there are higher degrees of uncertainty about whether the given HM actually satisfies all constraints. In our current situation where no further data will be collected, it is justifiable to simply accept all HMs with $P_{ok} \geq 0.5$ and eliminate all HMs with $P_{ok} < 0.5$. In other situations in which insufficient testing is performed (such as during a learning experiment), it is desirable to eliminate only HMs that are very certain to violate one or more constraints (see Section 4.4.2). In this case, $P_{ok} < 0.5 - \Delta$ for significant values of Δ, such as 0.25, should be used as the criteria to eliminate HMs from further consideration.

2.5.3 Performance evaluation across subdomains

In this section, we deal with the situation in which a target problem domain contains more than one problem subdomain. Since our principal objective in performance evaluation is still *to compare the performance of two HMs across the entire problem domain*, this phase of performance evaluation is very important. For a given set of problem subdomains, we can estimate the performance of each HM within each problem subdomain in the set. Based on this performance, we can evaluate and compare HMs over a given set of problem subdomains.

Unfortunately, because performance across different problem subdomains is not IID, we cannot use statistical estimation methods to generalize/predict performance based on a set of problem subdomains to subdomains not evaluated. This is a major reason why we should *minimize the number of problem subdomains within a problem domain* whenever possible, *i.e.*, maximize the size of each problem subdomain.

For performance evaluation over the entire problem domain based on a selected set of problem subdomains to be effective and meaningful, these selected subdomains must be *representative of the entire problem domain.* This representation implies that for any subdomains within the given problem domain, the two HMs' behavior will (theoretically) be similar to one of the selected subdomains. This representation cannot be guaranteed without exhaustive testing.

For a selected set of representative subdomains, we can evaluate performance of the two HMs over the problem domain by treating the performance of each subdomain independently. For one HM to be considered better for the entire problem domain, it must be considered better in every selected subdomain. This condition means that for every subdomain the selected HM must satisfy given constraints and, in the subdomain where both HMs satisfy the constraints, must have better (sample) average normalized performance with respect to the optimization objective.

The general performance-evaluation process across multiple subdomains based on the above criteria can be divided into two steps.

(1) Eliminate HMs that violate the given constraints in one or more subdomains ($P_{ok} < 0.5$ in one or more selected subdomain(s)).

(2) When no HMs or one HM remains, no further decision is necessary. With all HMs pruned (no HMs remaining), neither HM is considered better for the target problem

domain. With one HM remaining, the remaining HM is considered better in the given problem domain.

When both HMs satisfy all of the constraints, they must be ordered based on their average normalized performance on the optimization objective. The better HM is the one with the higher average normalized performance in every selected subdomain. For HM_X to be considered better, $P_{win}(X, J_i, \pi) > 0.5 + \Delta$ for all π. For HM_B to be considered better, $P_{win}(X, J_i, \pi) < 0.5 - \Delta$ for all π. The condition $\Delta > 0$ can be applied to account for a higher degree of certainty in the ordering in each subdomain.

The difficulty arises when each HM has better average performance for different subsets of problem subdomains. In this case, the ordering between the two HMs is considered indeterminate and is similar to the case in which both HMs are eliminated due to constraints.

When both HMs satisfy all of the constraints and have better average performance on different subsets of selected subdomains, users must select the HM to apply. We cannot simply combine the average performance values from different subdomains since they come from different distributions. Some possible heuristics that can be used include average P_{win} across all subdomains or best/worst case P_{win}. In Chapter 5, we go into more detail about some possible heuristics for ordering HMs when there is no dominant HM.

We demonstrate our process for performance evaluation across multiple subdomains in the following example.

Example 2.15 [VC] We continue to use the vertex-cover problem (see Example 2.2) as a running example. We want to compare one selected HM (HM_{271}) with respect to the existing HM used in solving VC problems (HM_1).

Based on Example 2.12, each subdomain contains test cases with different problem sizes (number of nodes in the graph) but with the same DC (degree of connectivity). We have selected six subdomains with DCs ranging from 0.1 to 0.6 to represent various regions within

Table 2.16: Example of performance evaluation across multiple subdomains in the vertex-cover problem. HM_{271} is compared against the original and baseline HM, HM_1. Performance of the baseline HM, HM_1, is equal to $BL = 0$ for all subdomains. In each subdomain, 30 test cases have been evaluated.

Performance Measure	Degree of Connectivity (DC)					
	0.1	0.2	0.3	0.4	0.5	0.6
$\hat{\mu}_{271}^{sym+}$	0.283	0.068	0.060	0.049	−0.000	0.028
$\hat{\sigma}_{271}^{sym+}$	0.571	0.209	0.124	0.200	0.078	0.067
$P_{win}(271, \pi)$	0.994	0.957	0.994	0.905	0.500	0.985

the problem domain. The performance of each HM is normalized using our symmetric-improvement method with HM_1 as the baseline (performance of $HM_1 = BL = 0$). Table 2.16 shows the normalized performance of HM_{271} for different DCs. In this example, we collect the performance of both HMs on 30 random test cases in each subdomain.

Since there is only one objective and no constraint, both HMs are acceptable and must be compared based on the optimization objective (normalized number of search nodes expanded). From this table, we observe that, in all subdomains except the one with DC=0.5, HM_{271} has better estimated performance (sample mean, $\hat{\mu}$) than HM_1 with a significantly high certainty level (high probability of win). When DC=0.5, the performance of HM_{271} is very similar to that of HM_1, which indicates high uncertainty about whether HM_{271} is better than HM_1 for this subdomain.

Based on our performance-evaluation criteria, if $\Delta = 0$, then HM_{271} can be selected as the best HM across the problem domain. For any $\Delta > 0$, then HM_{271} cannot be considered better than HM_1 due to the subdomain with DC=0.5. In the latter case, most heuristics and users would still prefer HM_{271} over HM_1 based on the available performance information.

■

2.6 Validation

In previous sections, we have presented our approach to performance evaluation. In this section, we provide some methods for validating our performance-evaluation approach. This validation process is necessary to make sure that the results and conclusions reached are correct and meaningful. In this section, we first outline a performance-evaluation process that, based on statistical estimation of a small subset of subdomains, provides meaningful results for the entire problem domain. Next, we present a method for a more thorough examination of trade-offs among multiple objectives. Our current graphical method is applicable for the case with two objectives.

2.6.1 Validation based on unseen test cases

In the previous section, we have shown that the HM which performs consistently better in all selected subdomains will likely perform better for other subdomains in the target problem domain. This situation is based on the assumption (a) that the selected subdomains are representative of behavior within the target problem domain and (b) that the sample mean is indicative of the true performance of each HM within a problem subdomain.

In this section, we present a validation process for checking the performance of two HMs on test cases not seen during performance evaluation: both from old and new subdomains. If the HMs can be generalized, then their performance on new test cases should be consistent with the conclusions drawn during performance evaluation. This consistency means that:

(1) for subdomains tested during the design process, the sample means of each HM on new performance values in each subdomain should be similar to previous sample means (for performance values obtained during the design process) for the same subdomain, and

(2) for all subdomains, the ordering of HMs based on sample means in each of these subdomains should be similar to the orderings found during the performance-evaluation process.

Table 2.17: Validation of performance on previously evaluated subdomains for the vertex-cover problem (see Table 2.16). Performance results are based on evaluating 15 new test cases in each subdomain ($\mu_1^{sym+} = BL = 0$).

Performance Measure	Degree of Connectivity					
	0.1	0.2	0.3	0.4	0.5	0.6
$\hat{\mu}_{271}^{sym+}$	0.638	0.078	0.020	-0.004	-0.000	-0.010
$\hat{\sigma}_{271}^{sym+}$	1.361	0.189	0.112	0.064	0.027	0.020
$P_{win}(271, \pi)$	0.955	0.934	0.750	0.406	0.500	0.037

Example 2.16 [VC] In this example, we verify the performance of the two HMs for the vertex-cover problem discussed in Example 2.15. We first re-evaluate both HMs on subdomains used in Example 2.15. The performance results based on 15 additional test cases are shown in Table 2.17.

From this table, we notice that performance values for DC=0.4 and DC=0.6 are not consistent with performance results found during performance evaluation. Performance results for DC=0.2, 0.3, and 0.5 are very close to the previous performance. Performance for DC=0.1 is different and exhibits a higher degree of variation due to some outliers in the new set of test cases. The ordering of the two HMs is consistent with previous orderings for DC=0.1, DC=0.2, DC=0.3, and DC=0.5. Results for DC=0.4 and DC=0.6 produce different orderings although their sample-mean values have changed by less than 0.055. We notice that this performance (DC=0.4 and DC=0.6) does not deviate significantly from the previous performance when DC=0.5.

Table 2.18 shows the performance of HM_{271} when normalized with respect to HM_1 for 6 new subdomains. These performance values are largely consistent with performance values from subdomains used during our evaluation process and are very similar to performance values in subdomains with similar DCs. HM_{271} consistently performs better than HM_1 except in the subdomain with DC=0.55. However, HM_{271} has a lower degree of certainty (probability of win) in some of these new subdomains than in Example 2.15. For DC=0.55, the performance is not too far out of line as compared to that of DC=0.5.

Table 2.18: Validation of performance on subdomains not used during performance evaluation for a vertex-cover problem (see Table 2.16). Performance results are based on evaluating 30 test cases in each subdomain ($\mu_1^{sym+} = BL = 0$).

Performance Measure	Degree of Connectivity					
	0.05	0.15	0.25	0.35	0.45	0.55
$\hat{\mu}_{271}^{sym+}$	0.218	0.031	0.054	0.017	0.016	−0.011
$\hat{\sigma}_{271}^{sym+}$	3.178	0.209	0.179	0.057	0.071	0.034
$P_{win}(271, \pi)$	0.645	0.609	0.945	0.943	0.886	0.043

Based on this information, we see that there can be misleading conclusions for individual subdomains (such as for DC=0.4 and DC=0.6) even at a high degree of certainty. In addition, if no uncertainty factor is taken into consideration ($\Delta = 0$) during the performance-evaluation process, the final evaluation result may be incorrect, as HM_{271} is not better than HM_1 in all subdomains. HM_{271} is better than HM_1 in several subdomains but is also slightly worse than HM_1 in some subdomains, sometimes with a high degree of certainty (low P_{win} for DC=0.55 and for the new set of validation data with DC=0.6). However, reasonable conclusions can still be reached in most cases by taking the degree of certainty into consideration ($\Delta > 0$). For example, the conclusion reached during the evaluation process with $\Delta > 0$ is consistent with all validation performance results shown in this example. ■

2.6.2 Graphical comparison of HM performance

In Section 2.3, we have proposed methods to address trade-offs among multiple objective measures by treating each objective independently and by applying constraints on the *average* normalized performance of each constrained measure. In short, we have only considered trade-offs among average objective measures in each subdomain independently. However, the average performance by itself does not provide a complete picture on the effectiveness of HMs and their trade-offs among different objective measures. Thorough understanding of the trade-offs provided by different HMs requires more detailed examination than just their average performance values [36, 42, 51].

In this section, we present a graphical method for presenting multiple sets of distributions for cases with two objective performance measures. This method allows a more thorough examination of trade-offs between two objective measures for one or more HMs. Methods for dealing with more than two objective measures are left for future research.

We propose using graphical plots of distributions of performance values for different objective measures and different problem subdomains as a tool for detailed evaluation of performance and trade-offs. The simplest graphical method is a scatter plot of performance with respect to each test case. However, with multiple subdomains and multiple test cases for each subdomain, a scatter plot may contain too many data points that obscure the overall trade-off, which is especially true when comparing performance of different HMs. Hence, additional information related to the distribution of performance values of an HM in a subdomain must be provided within each graphical plot.

The overall process for obtaining a plot of performance trade-offs based on their distributions in each subdomain can be enumerated using the following steps [42, 51]:

Step 1: The first step in plotting performance trade-offs is to compute/find the values of each performance measure for each test case with respect to different HMs. If necessary, normalization should be applied to make resulting performance values IID. Although a scatter plot can be achieved this way, additional steps are required to obtain a more comprehensive view of the performance of HMs.

Step 2: We must eliminate from the performance data any *clearly* anomalous data points that might unduly influence the statistical result we want to observe. We are mainly interested in *univariate outliers*, *i.e.*, the few extreme cost or quality values that do not fit with other data points. These can be detected simply by computing the standardized score $((x-\mu)\,/\,\sigma)$ of the (normalized) performance values for the given HM. Any data point with standardized scores in excess of a threshold such as 8.00, *i.e.*, a point more than 8 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 points with an unusual combination of cost and quality values, can also be detected by computing the Mahalanobis distance for each data point. A data point whose Mahalanobis distance has less than a 0.001 probability of happening can be considered an outlier [63]. The application of this method requires that the joint distribution is a multivariate normal distribution; hence, if normality is not true, then the next step must first be applied before computing the distance. This step may not be necessary if performance data after eliminating univariate outliers appear to be reasonable.

Step 3: Next, we determine the joint distribution between the two (normalized) measures for a given HM. The most likely form of the joint distribution is a bivariate normal distribution.

To validate that this assumption applies to a data set, we must check that the marginal distribution of each (normalized) measure is normal. There are many methods for evaluating univariate normality; examples include computing the values of skewness and kurtosis [63], applying the goodness-of-fit test, such as the Kolmogorov-Smirnov [56] and Geary tests [44], and applying the Shapiro-Wilk test [64].

Next, we must evaluate bivariate normality of the joint distribution. A method we can use is the chi-square plot or gamma plot [65], 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. Other methods for checking multivariate normality can be found in [64].

If the performance values do not have a normal distribution, then a distribution has to be assumed first so that the values can be transformed into another set that can be tested for normality. Note that Steps 2 and 3 may need to be repeated a number of times in order to assure that the data points have a bivariate normal distribution and to remove outliers from the data points.

Step 4: Finally, we obtain a 90% constant probability density contour of the bivariate normal distribution representing the joint distribution of the two (normalized) measures. To do so, we first compute the mean vector, $\mathbf{M}$, and the variance-covariance matrix, Σ, for the distribution. The mean vector represents the centroid of the ellipse that represents this contour.

We then derive the axes for the ellipse that represents this contour from the eigenvalues λ_1, λ_2 and eigenvectors $\mathbf{e}_1, \mathbf{e}_2$ of the covariance matrix, Σ. The axes of the ellipse are represented as 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 [65]. The notation $\chi_\nu^2(\alpha)$ represents the critical value at level α of the chi-square distribution with ν degrees of freedom [44, 65]. These two vectors represent the direction and length of the axes for the ellipse with respect to the centroid of data set, $\mathbf{M}$. The resulting contour encompasses the area where 90 percent 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 into one for the original data points using the inverse transformation.

Example 2.17 [PGA] To demonstrate our performance trade-off plots, we have chosen to use the post-game analysis problem solver discussed in Example 2.6 as an example. As mentioned previously, there are two performance objectives for each PGA HMs: minimize the cost (time for finding a mapping) and maximize the quality of the mapping found (by minimizing the time for finding a mapping).

In this case, each of the performance measures is normalized based on the improvement-ratio method ($BL = 1$). Based on notations in Example 2.6, we can express these two normalized measures as follows:

$$\begin{aligned} c_{i,j}^{norm} &= \frac{c_{i,j}}{c_{B,j}}, \\ q_{i,j}^{norm} &= \frac{t_{B,j}}{t_{i,j}}, \end{aligned} \tag{2.21}$$

where the objective is to maximize $q_{i,j}^{norm}$ and minimize $c_{i,j}^{norm}$.

Figure 2.9 shows the performance-tradeoff plot between normalized cost and quality obtained using the steps described above for comparing a new PGA HM with respect to the original baseline PGA HM (HM_B) [39]. Note that these two normalized measures ($c_{i,j}^{norm}$ and $q_{i,j}^{norm}$) were verified to have bivariate normal distributions without requiring any transformations. In this example, there are no outliers. From this figure, we see that the

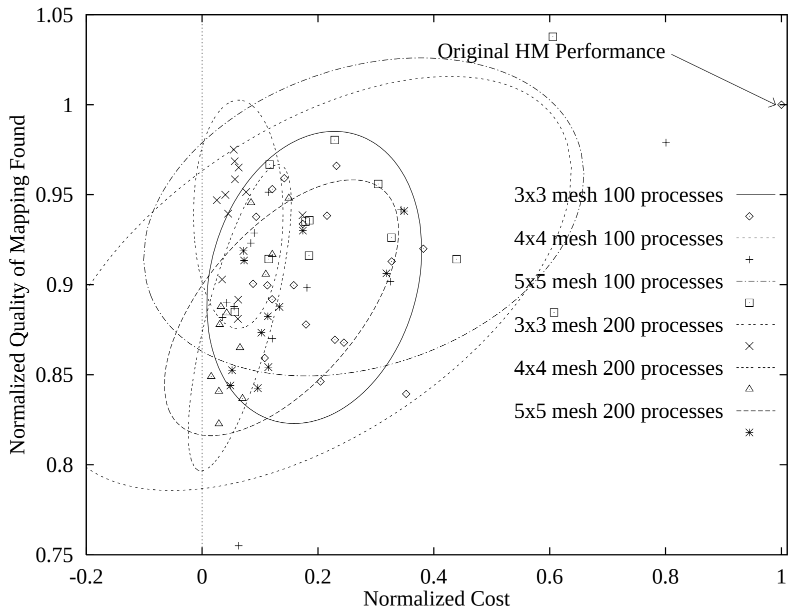


Figure 2.9: Plot of performance trade-offs between normalized cost and quality of two PGA HMs (HM_B and HM_1) over multiple problem subdomains.

new HM has much lower cost with slightly lower quality than the baseline HM for all test cases. ■

2.7 Summary

In this chapter, we have explored the process of performance evaluation of two HMs over a problem domain. The two HMs case is a special case of the more general performance-evaluation problem. Our performance-evaluation objective in this research is to determine the HM that has *better average normalized performance over the given problem domain for every objective performance measure.* We have identified two key issues in the performance-evaluation process that must be addressed:

(1) multiple objective performance measures with unknown trade-offs, and

(2) a large (possibly infinite) number of performance values for each HM due to variations in performance and a large (possibly infinite) number of test cases in the problem domain.

We have developed a systematic performance-evaluation process that can deal with these issues. There are six steps in our process for performance evaluation of two HMs.

1. *Normalize* the performance values to (a) obtain relative performance between the two given HMs with respect to each test case and (b) reduce differences in magnitude of performance values with respect to different test cases.

   To avoid anomalies (inconsistent orderings) due to the choice of the baseline HM and the choice of normalization method, we have developed normalization methods that satisfy the *symmetric-normalization condition.* These *symmetric-normalization methods* can eliminate anomalies due to the choice of the baseline HM and can potentially reduce anomalies due to the normalization method chosen.

2. When there are multiple objective measures, *apply constraints on all but one or on all objectives measures.* The goal is to find the best HM among the HMs that satisfy all of the constraints based on a single objective (usually the single unconstrained measure when not all measures are constrained).

   The original goal of finding one better HM under all objectives may be difficult to fulfill in most cases due to trade-offs that exist among these objectives. Hence, it may not be possible to determine which HM is better all of the time.

   We want to avoid formulating trade-offs among objective measures in terms of a single parametric function because it is difficult to determine appropriate parameters to result in desired trade-offs.

3. Partition the problem domain into *subdomains.* Normalized performance values within each subdomain must be *independent and identically distributed (IID).*

   We have presented several statistical tests for evaluating when a given set of data is likely to satisfy the IID property. These tests can be used to identify test cases that can be grouped into subdomains.

Within each subdomain in which performance values are IID, *statistical estimation* can be applied to predict the true performance level (population mean) based on a subset of performance data.

4. Evaluate the performance of each HM within each subdomain. Due to the IID property in each subdomain, the performance of each HM within a given subdomain can be estimated based on a subset of test cases. The sample-mean values can be used to decide (a) whether an HM satisfies or violates a given set of constraints and (b) when an HM has higher performance for the single optimization objective.

   However, there is uncertainty involved in using the estimated sample means to decide which HM is better. Probabilistic analysis can be used to determine the degree of certainty involved in using the sample-mean values in any decision-making process. This probabilistic analysis results in P_{ok} for checking constraint satisfaction and P_{win} for ordering the average performance with respect to the single optimization objective. These probabilistic measures can be applied to increase the degree of confidence in identifying the better HM within each subdomain.

5. Evaluate performance over the entire problem domain. Since performance results from different subdomains are likely to come from different distributions and have different statistical properties, they should not be combined. Our method is to treat each subdomain independently.

   The best HM over the entire problem domain must be better in all subdomains, taking into account uncertainty about various statistical estimates (based on the method in the previous step). If neither HM is better in all subdomains, then the ordering of the two given HMs cannot be determined. In this case, users have to pick one HM based on their application requirements.

6. Verify performance-evaluation results by (a) making sure that additional performance values are consistent with the results already collected and (b) using additional distribution information (other than the average, P_{win}, and P_{ok}) to get a better view of trade-offs among different objective measures.

Later in this thesis, we extend our performance-evaluation process to handle the general case with more than two HMs. Section 3.6.4 presents an extension to performance evaluation within each subdomain to deal with the case with more than two HMs in the learning process. Chapter 5 presents an overview of performance evaluation over the entire problem domain with more than two HMs.

3. HEURISTICS-DESIGN PROCESS

In this chapter, we study issues in designing new heuristics used in problem solving. Our study has been focused on developing knowledge-lean heuristics with nondeterministic performance with an objective of finding, under resource constraints, a new HM that performs "better" than existing HMs.

There are five potential issues in a heuristics-design process: (1) *decomposition* of a problem solver into smaller components and *integration* of new HMs designed for each component together, (2) *classification* of a problem domain into subdomains so that statistical performance evaluation can be performed, (3) *generation* of new and improved HMs based on past performance information and heuristics generated, (4) *evaluation* of each HM's performance, and (5) *generalization* based on available performance information to find HMs that perform well across the entire problem domain.

Based on the survey of existing methods in machine learning, we have developed the TEACHER (an acronym for TEchniques for the Automated Creation of HEuRistics) system using a genetics-based machine learning approach. TEACHER divides the heuristics-design process into four distinct phases. (1) In the classification phase, TEACHER divides the problem domain into subspaces (based on user requirements) and problem subdomains (based on the performance behavior of HMs). (2) In the learning phase, TEACHER generates HMs and evaluates them under resource constraints so that new and better HMs can be discovered. (3) In the verification phase, TEACHER evaluates further good HMs from the learning phase to acquire more accurate and more complete performance information. (4) Finally, in the generalization phase, TEACHER selects the HM most likely to provide the best performance over the entire problem domain based on available performance information.

3.1 Overview of the Design Process

In this section, we present a brief review of the objective and various issues involved in the heuristics-design process described in Chapter 1.

A *heuristics-design process* is a process for developing or *learning* new and "better" HMs for a problem solver. Our motivation is to design better HMs to be used in a problem solver so that better and/or less costly solutions for an application can be found. In this thesis, our objective is to *find new HMs that perform "better" than existing HMs with respect to some average objective measures over the entire problem domain.*

As stated in Chapter 1, we have focused our research on dealing with problems that fit the following specifications:

- an application with a large problem domain,
- a knowledge-lean problem solver that does not have a good world model to relate specifications of an HM to its performance,
- performance-related heuristics (*i.e.*, the choice of HMs affect only the quantitative performance of the problem solver and/or solution),
- expensive evaluation of an HM on a test case, and
- a large (possibly infinite) number of possible HMs.

There are five major issues that need to be addressed in developing new HMs for a target application.

(1) **Decomposition and Integration of Problem-Solver Components**. For a complicated problem solver with many heuristic components, there are many (possibly infinite) possible combinations of heuristics that can be used in each component. In many cases, it is not efficient to try to develop new heuristics for all components simultaneously.

A possible way to improve a complicated problem solver is to use a piecewise fashion. This piecewise improvement would involve (a) decomposition of the problem solver into a number of groups with a smaller number of components in each group, (b) design of "good" heuristics for each group, and (c) integration of "good" heuristics from each group so that the performance of the problem solver can be optimized.

There are several difficulties with this *decomposition/integration process.* First, it is difficult to decide how to group heuristic components so that the final result is optimal. Second, there may be a high degree of interactions needed in designing "good" heuristics for each group in order for the final result to be acceptable. It is difficult to determine the appropriate amount of interactions between designing different parts of the problem solver. For this decomposition/integration process to be efficient, some domain or expert knowledge is required.

One simple method that minimizes the interactions among the design of heuristics for different groups uses a *sequential approach.* First, new heuristic (HM_A) is developed for group A. Then, a new heuristic is developed for Group B with HM_A used in the problem solver. This process can be repeated until heuristics from all groups have been designed.

(2) **Classification of the Problem Domain**. When the performance of an HM is nondeterministic and the cost for evaluating each test case is expensive, only a small number of test cases are likely to be used in evaluating each heuristic method in the design process. We have shown in Chapter 2 that the true performance of a heuristic method can be estimated based on a subset of test cases only when the performance values are independent and identically distributed (IID).

We have also shown in Section 2.5 that different sets of test cases can have different performance behavior. When this happens, it is necessary to partition the problem domain into smaller subsets (known as subdomains in this thesis) to be evaluated independently. In addition, different regions of the problem domain may require different types of heuristics and different partitions of the problem domain in order to obtain appropriate heuristics for each region of the problem domain. In this thesis, we refer to the division and partitioning of the problem domain as the *classification problem.*

(3) **Generation of Heuristics**. To design new and "better" heuristics, it is obviously necessary for the new heuristics to be generated. The *generation of new and "better" heuristics* based on performance information obtained so far has been the focus of most studies in machine learning. The next section contains an overview of previous work related to this issue.

This generation process is much more difficult for knowledge-lean problem solvers since there are no models for relating the specifications of the heuristic to its performance. In this case, the problem solver must be treated as a black box, and heuristics generation must use weak generation methods that do not depend on domain knowledge.

(4) **Evaluation of Heuristics**. To obtain "better" heuristics, we must be able to compare the performance of different heuristics. Performance for these heuristics must then be obtained based on the *evaluation of each heuristic method on test cases.* When the performance of a heuristic is nondeterministic, this evaluation process is more difficult.

The key issue is to be able to compare performance between different heuristics using minimal tests. This ability is especially important in our research since we may be dealing with expensive knowledge-lean problem solvers with nondeterministic and nonstationary performance values over a large problem domain.

(5) **Generalization of Heuristics Learned**. When the performance of each heuristic method is nondeterministic and varies across different test cases, only a subset of test cases is usually used in evaluating the performance of each heuristic during the design process. However, the heuristic selected must perform well not only for test cases used during the design process but also for test cases not seen before. This ability of the selected heuristic is necessary to ensure that the performance of the selected heuristic on a random test case within the problem domain will be good (as compared to the best existing heuristic). This problem is known as the *generalization problem.*

This generalization issue is more important but more difficult when there is a large problem domain and different regions (different subdomains) in the problem domain have different (and possibly inconsistent) performance behavior.

A general model of the heuristics-design process that addresses all of these important issues is presented in Figure 3.1.

Most previous work in an automated heuristics-design process has focused on the issues of generation and evaluation of new heuristics. These issues are the traditional problems in learning. We present an overview of this work in the next section. There has also been some work on the evaluation and generalization issues. This work is discussed in Chapters 2, 4, and 5 when we present our approaches for dealing with these issues. The other two issues (decomposition/integration and classification) have been mostly ignored in the literature since they exist only in complicated design problems. Unfortunately, many real-world applications may fit into this category, and some solutions to these issues are required.

For the type of heuristics-design problems on which we have focused, *all issues* are potentially important. We have already developed a systematic framework that can address all of these issues except the issue on decomposition and integration. We currently require the decomposition-and-integration process to be performed manually by users due to the limitation of our resources. The users can then develop new heuristics for each group in a *sequential approach.* This manual process is necessary because the decomposition-and-integration issue can be studied only after all other issues have been addressed and a good system for designing new heuristics for each component has been developed. Future research will incorporate this issue into our learning framework.

Our TEACHER system that implements our heuristics-design process is presented in Section 3.4. Our system has four different phases of operation. Each phase is isolated to deal with different design issues. We present an overview of our approach in each phase in this chapter. The first phase deals with the classification issue and is discussed in Section 3.5. The second phase deals with both the generation and the evaluation issues and is intended for generating good heuristics for each subdomain (see Section 3.6). The operation in this phase is the one referred to by most researches as a learning process. Our approach in this phase is based on the genetics-based machine learning approach presented in Section 3.3. The third phase also deals with the evaluation issue and is intended for getting more accurate

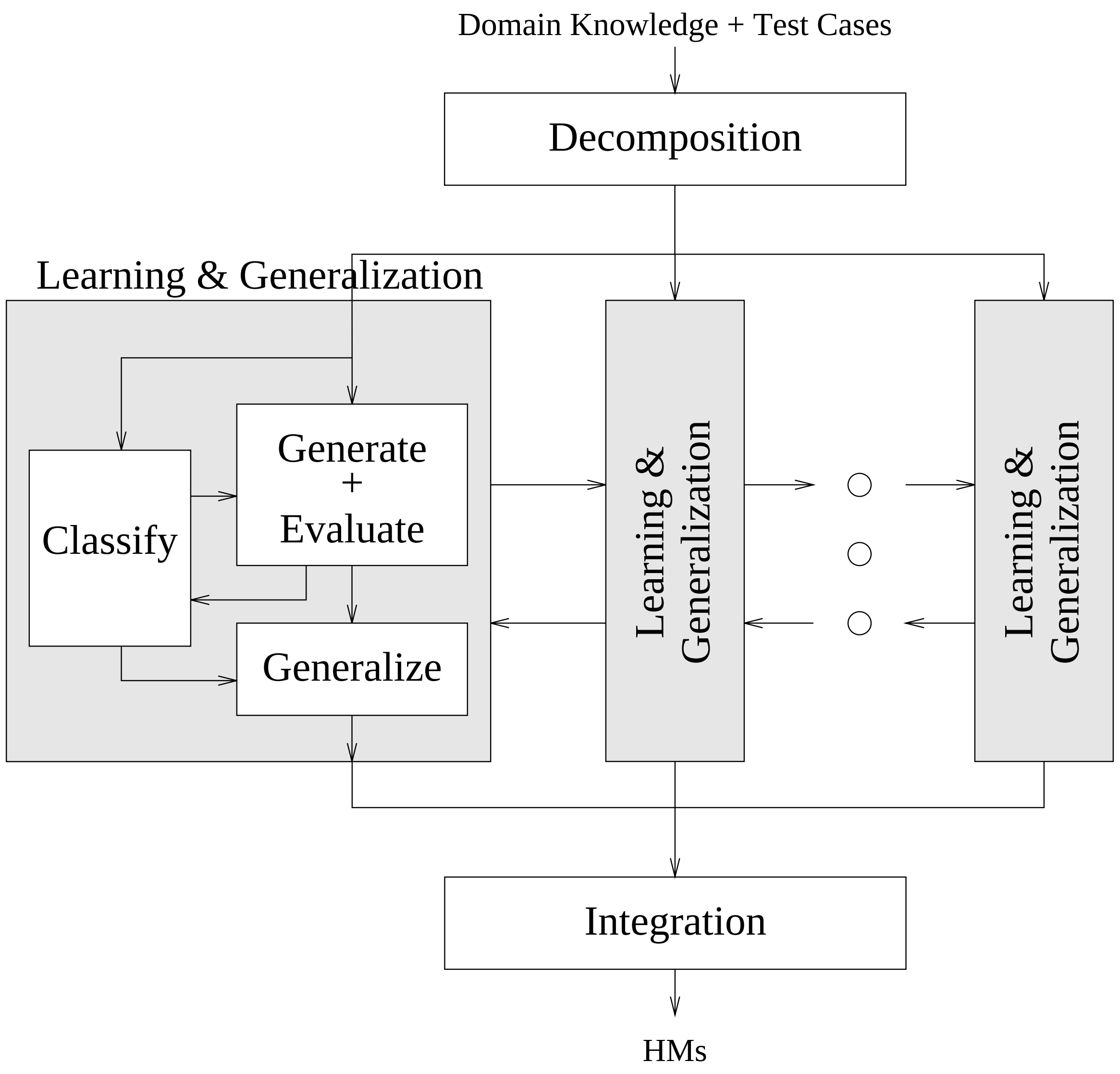

Domain Knowledge + Test Cases
Decomposition
Learning & Generalization
Classify
Generate
+
Evaluate
Generalize
Learning & Generalization
Learning & Generalization
Integration
HMs

Figure 3.1: A general model of the heuristics-design process.

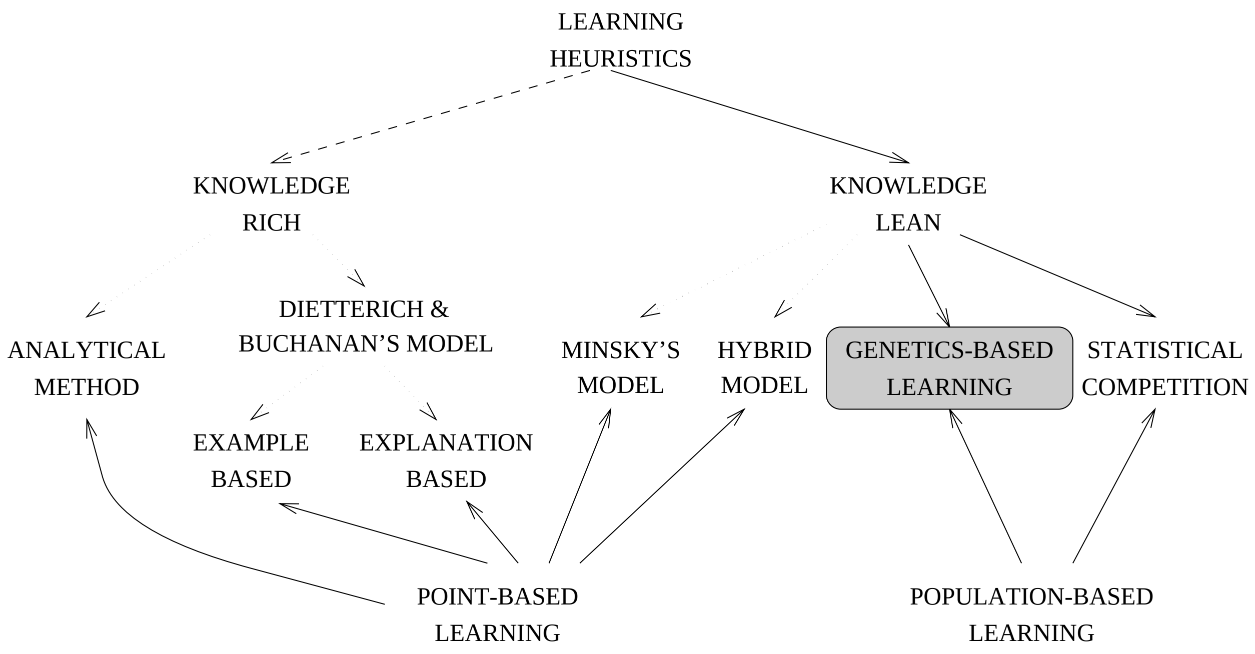


Figure 3.2: Classification of methods for learning heuristics.

performance evaluation for the heuristics selected (see Section 3.7). The final phase deals with the issue of generalization and is discussed in Section 3.8.

3.2 Background on Heuristics Learning

In this section, we present a brief survey of previous work in the area of automated design of new heuristics. The work in learning new heuristics has focused mainly on the issue of generating and evaluating new and "better" heuristics. We first present a classification of approaches in heuristics learning. We then follow this classification to present examples and strengths and weaknesses of different learning methods. We then present a summary of how this work is related to our current research and the type of design problems we have studied.

3.2.1 Classification of heuristics-learning methods

The method for learning heuristics and strategies can be classified as shown in Figure 3.2. There are two main methods we have selected for classification of existing learning methods.

The first classification is based on the amount of domain knowledge about the target problem solver and heuristics that can be used in the design process. In this classification, a learning method can deal with either *knowledge-rich* or *knowledge-lean* heuristics [51]. The second classification is based on the number of heuristics maintained during the design process by the learning system. In this classification, a learning method can be either *point based* or *population based* [51, 66].

Knowledge-Rich Heuristic: As mentioned in Chapter 1, a heuristic method is classified as *knowledge rich* if there exists a good world model for characterizing the relationship between this heuristic method and its performance.

Knowledge-Lean Heuristic: When a model for characterizing the relationship between a heuristic method and its performance is unknown, the heuristic method is classified as *knowledge lean*. In this research, the heuristics to be learned are knowledge lean.

Point-Based Learning Paradigm: In the *point-based learning paradigm*, the learning system maintains one incumbent heuristic method that is modified in place by the learning system. Since each modification of the HM destroys the original HM, there must be high confidence that the new HM would be better than the old one. This learning paradigm works well for learning knowledge-rich heuristics because the world model can be used to guide the generation of new HMs [7, 67].

Three models for learning HMs in traditional machine-learning studies fit within this paradigm. Fundamental work in this area was addressed by Mitchell [68,69], Minsky [70], and Dietterich and Buchanan [71]. The basic principle is based on a generate-and-test paradigm that generates plausible HMs, performs limited tests, and modifies the HMs according to feedback signals obtained. Each of these models are described briefly later on in this section. Many existing machine-learning systems fit one of these models.

The general point-based learning paradigm used by all three point-based models is shown in Figure 3.3 [51]. The general model includes the Learning Performance Database and Preprocessor, Credit Assignment unit, and Learning Element. The problem solver and its initial conditions are shown as components outside the learning system.

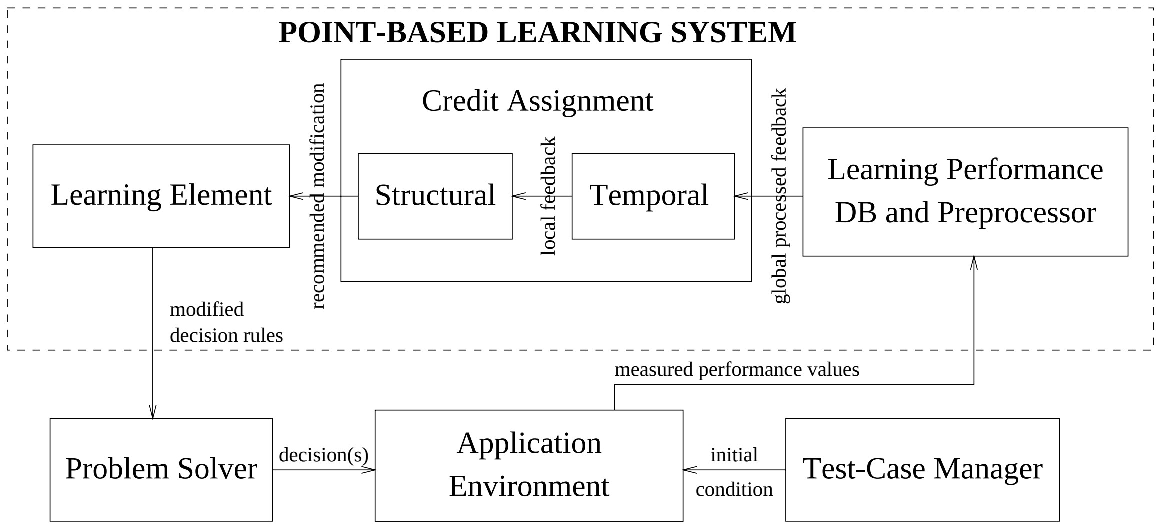


Figure 3.3: A general point-based learning model.

The *Learning Performance Database and Preprocessor* captures the effects of decisions made by the problem solver on the application environment. The preprocessed data are then used for *credit assignment* which include both *temporal* and *structural* credit assignments. The *Learning Element* then modifies the heuristic method based on recommended modifications from the Credit Assignment unit.

Population-Based Learning Paradigm: In contrast to the point-based approach, the *population-based learning paradigm* maintains multiple competing HMs and tries to find the best HM within the pool. During the learning process, new HMs are added to the pool and poor ones removed. This learning paradigm is useful for learning knowledge-lean heuristics because we can apply weak methods for generating new heuristics that do not depend on a good world model [3, 10]. It is not necessary for every new HM to perform well since there are other alternatives.

One important difference between the population-based and the point-based approaches is the potential need for scheduling available resources among the competing HMs in the former (see Chapter 4). More details about this paradigm are presented in Sections 3.3 and 3.6.

3.2.2 Learning knowledge-rich heuristics

Several general methods have been proposed for learning knowledge-rich heuristics. These methods include analytic methods, learning by examples, and explanation-based learning. Learning by examples and explanation-based learning are examples of Dietterich and Buchanan's model of learning.

Analytic Method. An *analytic method* is based on comprehensive analysis on the problem solver with respect to its particular representation [72, 73]. This approach requires a lot of domain knowledge and is very specific to a particular application.

Dietterich and Buchanan's Model. This point-based learning model [51, 71, 74] and similar models proposed by Smith et al. [75] and Langley [76] belong to a class of models for learning HMs of target problems with well-defined objectives. They learn by supervised learning with *perscriptive feedback* that carries explicit information about the desired results to guide the modification of the HM tested. This learning model fits within the framework of the general point-based model shown in Figure 3.3.

Learning by example narrows the scope of possible heuristics by specialization and generalization [7]. *Explanation-based learning* exercises domain knowledge to explain the relationship between a heuristic method and its performance [6, 77].

The major problem with learning knowledge-rich heuristics is that extensive domain knowledge must be available; however, the focus of this research is on learning heuristics without such knowledge.

3.2.3 Learning knowledge-lean heuristics

Several general methods have been proposed for learning knowledge-lean heuristics. These include Minsky's learning model, hybrid point-based learning model, genetics-based learning, and statistical methods.

Minsky's Model. This model [51, 70] is an older and perhaps less restricted model of point-based learning systems. It applies well for learning HMs for target problems with undefined objectives in knowledge-lean environments, which tend to produce evaluative and possibly delayed feedback signals.

An *evaluative feedback* carries only implicit information about the desired behavior but explicit evaluation of the observed behavior. Such behavior are intrinsically *a posteriori*, being measured or generated after the behavior has occurred. It requires a critic [78] that has some prior knowledge of the objective function and can assess the goodness of external states or sequences thereof. Scalar evaluative feedback signals are called *reinforcements* [70] and learning from such signals, *reinforcement learning*.

This model requires the Markovian model to be satisfied in order to simplify temporal credit assignment. This condition is not true in most complex applications. Examples of systems in this class include Klopf's drive reinforcement model [79] and Sutton and Barto's reinforcement model [80].

Hybrid Learning Model. This point-based learning model [51] combines aspects of Dietterich and Buchanan's model and Minsky's model. This learning model uses an approximate temporal model instead of the Markovian model. It is intended for dealing with a knowledge-lean learning environment with an ill-defined objective and evaluative feedback and a non-Markovian temporal scope.

Examples of this type of learning system include EURISKO [81], Samuel's Checker Player [9, 82], Williams' REINFORCEMENT model [83], classifier system (the Michigan approach) [84], and the truck-backer-upper problem of Widrow et al. [85]. This type of learning does not address the lack of domain knowledge for structural credit assignment.

Genetics-Based Machine Learning. This population-based approach is based on generate-and-test methods that generate new heuristics to be tested by applying operators to existing heuristics that perform well [3, 10]. The reproduction operators applied include crossover and mutation. The new heuristics are potentially good as they are generated based on good ones. It is based on the application of genetic algorithms [27,28] to machine-learning problems. More details about this approach are presented in Section 3.3.

Examples of genetics-based machine learning include genetic programming [29] and the Pittsburgh approach to classifier system [3].

Statistical Competition. One form of statistical methods uses statistics to translate data into concepts so that concept learning can be applied [12]. Another form of statistical methods uses statistics to decide which heuristics to test more, given a pool of candidate heuristics [42, 86]. This method is especially useful for learning heuristics whose exact performance cannot be determined by a limited number of tests.

The main shortcoming of the statistical competition approach is the limitation of a fixed pool of heuristics that excludes introduction of new and improved heuristics based on past performance information.

3.2.4 Summary

For our target problems of learning knowledge-lean heuristics, genetics-based machine learning is the most suitable approach. We obviously cannot use methods for learning knowledge-rich heuristics. The two point-based learning models for learning knowledge-lean heuristics have too many requirements that cannot be satisfied by the type of problems we want to address. Minsky's model requires a Markovian temporal model, and the hybrid model requires some domain knowledge for structural credit assignment. The statistical competition approach is limited to deal with a fixed pool of heuristics and does not deal with incremental improvements through mutations and crossovers.

A genetics-based machine learning approach can generate new heuristics with potential improvements over existing heuristics based on past performance information and without requiring a lot of domain knowledge. In our heuristics-design system, we also incorporate some aspects of the statistical competition approach to improve resource scheduling within the genetics-based learning framework. In the next section, we examine the genetics-based approach to machine learning in more detail and identify some key issues that we plan to improve over existing work.

3.3 Background on Evolutionary Computing

Genetics-based machine learning, the approach we have selected for our heuristics-design process, is a part of a bigger field called evolutionary computing [87,88].

Evolutionary computing (EC) include *genetic algorithms (GAs), evolutionary programming (EP), evolution strategies (ES), classifier systems (CFSs), genetic programming (GP)*, and several other problem-solving strategies. They are based on the following biological observations: the means of *natural selection* and the *survival of the fittest* and the theories of *evolution* [87]. They all share a common starting idea of providing the evolution of individual structures through the processes of selection, mutation, and reproduction. These processes depend on the perceived performance of the individual structures as defined by an environment [88].

In this section, we first present a brief overview of genetic algorithm (GA) before proceeding to genetics-based machine learning, an extension of genetic algorithms to machine-learning problems. Finally, we summarize the key differences between our approach to heuristics-design process (TEACHER) and traditional genetics-based machine learning.

3.3.1 Genetic algorithms

Genetic algorithms (GAs) are adaptive methods that may be used to solve search and optimization problems. The foundations of GAs have been developed by Holland [84]. Since then, GAs have been extensively studied [27,31,89]. They are based on the genetic processes of biological organisms described first by Charles Darwin in *The Origin of Species.* Populations of competing individuals evolve over many generations according to the principle of natural selection and "survival of the fittest."

Genetic algorithms work with a *population* of "individuals" and a set of biologically based operators defined over the population (such as *mutation* and *recombination* through *crossover*). Each individual represents a possible solution to a given problem. Based on the theory of evolution and the survival of the fittest, only the most suited elements in a population are likely to survive and generate offsprings [27].

In GAs, each individual is traditionally represented as a string of finite-length binary values (0 or 1). This uniform representation makes GAs more independent of the target applications and allows various studies to focus more on the properties and problems of the approach.

A *fitness level* is assigned to each individual based on how good of a solution to the problem that individual is. The highly fit individuals (having high fitness levels) are given opportunities to *reproduce* by *recombining* with other individuals in the population. This reproduction process results in new individuals known as *offsprings*. An offspring shares some features taken from each *parent*. The least fit members of the population are less likely to get selected for reproduction and so *die out*.

A whole new population of individuals (possible solutions) are thus produced by selecting the best individuals from the current *generation* and mating them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics possessed by good members of the previous generation. Over many generations, good characteristics are spread throughout the population, being mixed and exchanged with other good characteristics as they go. This process allows the most promising areas of the search space to be explored [90].

The overall process within GAs can be viewed as iterating over two different steps: (1) evaluation of individuals of a population in the current generation and assigning fitness levels to each individual and (2) generation of a new set of individuals (or new population) for a new generation by (a) selecting existing individuals based on their fitness values and (b) using selected individuals to reproduce (either by combining and/or modifying these individuals). This iterative process is shown in Figure 3.4.

In most traditional GAs, the fitness of each individual is exact and can be found with negligible costs. However, there are cases where there are "noises" in the evaluation process that result in variations of performance over multiple evaluations and in uncertainties over the fitness of each individual [91–93]. This condition can significantly increase the amount of fitness evaluations performed during each generation.

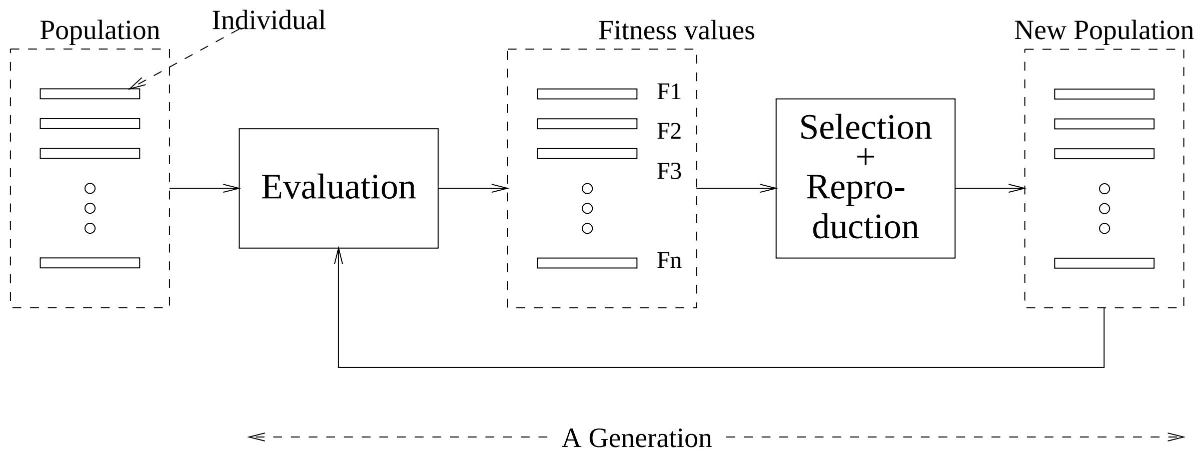


Figure 3.4: General model of the evolution process in a genetics-based approach.

GAs are used for solving many different problems in areas such as scheduling, combinatorial optimization, numerical function optimization, image processing and recognition, and engineering design [27, 87, 90].

There are many issues in genetic algorithms that have been and continue to be studied, including issues of representations, population size, evaluation of fitness, selection of individuals for reproduction, and reproduction methods [88, 90, 94, 95].

3.3.2 Genetics-based machine learning

Genetics-based machine learning is an extension of genetic algorithms (GAs) to machine-learning problems [27]. This term is used in this thesis to cover the applications of the idea behind genetic algorithms to develop a "heuristic" or "strategy" that represents a problem-solving process, instead of developing solutions for each problem directly.

Genetics-based machine learning is still based on the same idea of evolution and natural selection as in genetic algorithms. A population of individuals is maintained and evaluated for fitness values. A new population of individuals is then generated by selecting existing individuals for reproduction based on their fitness values (see Figure 3.4).

The higher complexities of target problems usually mean that the representation of each competing individual is more complicated than just a string of 0's and 1's [27, 29]. Some example representations of an individual in genetics-based machine learning include an if-then rule [27], a set of rules [96], a Lisp expression [29], or a vector of numbers [36].

Because the structure of each individual can be complex, the reproduction operators such as mutation and crossover can also be more complex. In addition, more domain knowledge can be applied to create knowledge-intensive reproduction [27, 94] such as those used in GIL [96].

Since the goal is to develop a "heuristic" or "strategy" for problem solving, this area of evolutionary computing also has to deal with "noisy" conditions more often. This condition means that the fitness of each individual may not be exact, and that multiple applications of each individual may be necessary.

All of these characteristics fit in with the type of heuristics-design problems we want to address: more complex applications with more complex representations for each HM and nondeterministic (noisy) performance.

The cost of evaluating each individual is usually higher in genetics-based machine learning than in traditional GAs since the target problems are more complex and noisy evaluation conditions are more common.

Existing work in genetics-based machine learning can be divided into two different approaches: (a) treating the entire population as the "heuristic" or "strategy" to be developed and (b) treating each individual as a complete strategy.

(A) Population as Heuristic. This approach treats each individual within the current population as a contributing component to the overall solution, which is the entire population. There must be cooperation among individuals in order to achieve better performance for the entire population. This approach is known in the genetic algorithm community as the Michigan approach [89, 97]. This approach requires credit assignments to divide the credits or debits to various contributing individuals in an episode. The most common credit assignment strategy is the *bucket brigade* algorithm [27].

This case with the entire population as a single solution is more like a point-based learning approach (using a hybrid model) rather than a population-based approach. It also requires that all components of a solution be homogeneous. However, this approach can be applied to improve the problem solver during the problem-solving process (*i.e.*, perform learning on-line).

Examples of this type of approach include most classifier systems (CFSs) such as CS-1 [27] and CFS-C [98].

(B) Individual as Heuristic. This approach treats each individual as a solution (or heuristic method) that competes with other individuals within the population. In this case, each individual may have to be more complex than with the first approach and can contain components that are entirely different from one another, *i.e.*, more heterogeneous components. This approach is known as the Pittsburgh approach in the genetic algorithm community [89, 97].

This approach does not require credit assignments to split up credits or debits since each performance feedback is directed to only one individual. As feedbacks come less often to each individual, this approach usually requires more evaluations to reach a final result.

This population-based approach is more suitable for our problems since our HM is usually nonuniform, with no good model of interactions among various components of the HM to allow credit assignments on performance feedback.

Examples of this type of approach include the Pittsburgh approach [89, 97] to classifier system (such as LS-1 [99], GABIL [100], and GIL [96]) and genetic programming (GP) [29].

There are some systems that use a hybrid of both approaches by treating each individual as a potential solution with components that can contribute to the problem-solving process. In this case, credit assignments *within each individual* to assign credits/debits to each component is useful. SAMUEL [3] is an example of this hybrid approach.

3.3.3 Key characteristics of TEACHER

Our present learning system is aimed towards methods for coping with anomalies in performance evaluation, general resource scheduling strategies in multi-objective learning, and generalization of HMs learned. By combining the following three features, our system is unique as compared to other genetics-based learning studies.

- Our learning environment is noisy so that the performance of each HM cannot be evaluated using a single test.
- We consider applications in which HMs behave differently under different situations (subdomains - see Section 2.5). Existing methods generally ignore this problem and focus on only one set of statistically related test cases.
- We assume that the cost of evaluating an HM on a test case is expensive. This cost forbids performing extensive tests on each HM. In several applications presented in Chapter 6, a fast workstation takes a few days to perform one to two thousand tests. This limited testing is in contrast to many other studies that assume that tests are inexpensive and that many tests can be performed in the time allowed [29]. For simplicity, we consider logical time in this thesis in which one unit of time is needed for each test of an HM.

These conditions are more realistic for complicated real-world applications for which we want to design new heuristics.

The goal of our study is to learn, under limited computational resources, good HMs for solving application problems and to generalize the HMs learned to unlearned subdomains. We choose to use the average metric for comparing HMs and examine the spread of performance values when HMs have similar average performance. When there are multiple objectives in comparing HMs, we constrain all but one objective during learning and optimize the unconstrained objective. In this case, our learning system proposes more than one HM, showing trade-offs among these objectives.

3.4 Overview of TEACHER

In this section, we discuss our system for designing heuristics. TEACHER is a genetics-based learning system that we have developed in the last six years [51]. Preliminary designs of the system have been studied with respect to learning process-placement strategies on a network of workstations [20], learning process-placement strategies on distributed-memory multicomputers [42], tuning parameters in a stereo-vision algorithm [21], learning smaller feed-forward neural networks [101], and learning new heuristics for a branch-and-bound search [102, 103]. We have also studied resource scheduling strategies in genetics-based learning algorithms [36,42,91,104,105].

The objective in developing TEACHER is to design, under resource constraints, HMs that are performance related and knowledge lean. The operations of the system are divided into several distinct phases. The operations in each phase are designed to independently deal with different issues in the heuristics-design process (see Section 3.1).

There are four phases of operation in designing new HMs using TEACHER: classification, learning, verification, and generalization. Currently, the classification phase must be performed manually while the other three phases are automated [41]. There are plans for incorporating automated classification and methods for dealing with the decomposition/integration issue into the TEACHER framework in the future. The overall design process used by TEACHER is presented in Figure 3.5. We describe the objectives and key issues of each phase in this section and our solutions for each of these phases in the following sections.

Classification phase

This first phase of the design process partitions test cases in an application into distinct subsets. There are two steps in this phase.

a) Subspace classification. We will show in Section 3.5 that a problem domain can have different regions to be solved efficiently by different HMs. In this case, each region should be identified whenever possible so that different HMs can be developed for different regions. The first step is then to partition the problem domain into a small number of distinct subspaces so that new HMs are learned/designed for each. Such partitioning is guided by commonsense knowledge expressed in the form of decision rules. By applying these rules, we can determine for a new test case the subspace to which it belongs.

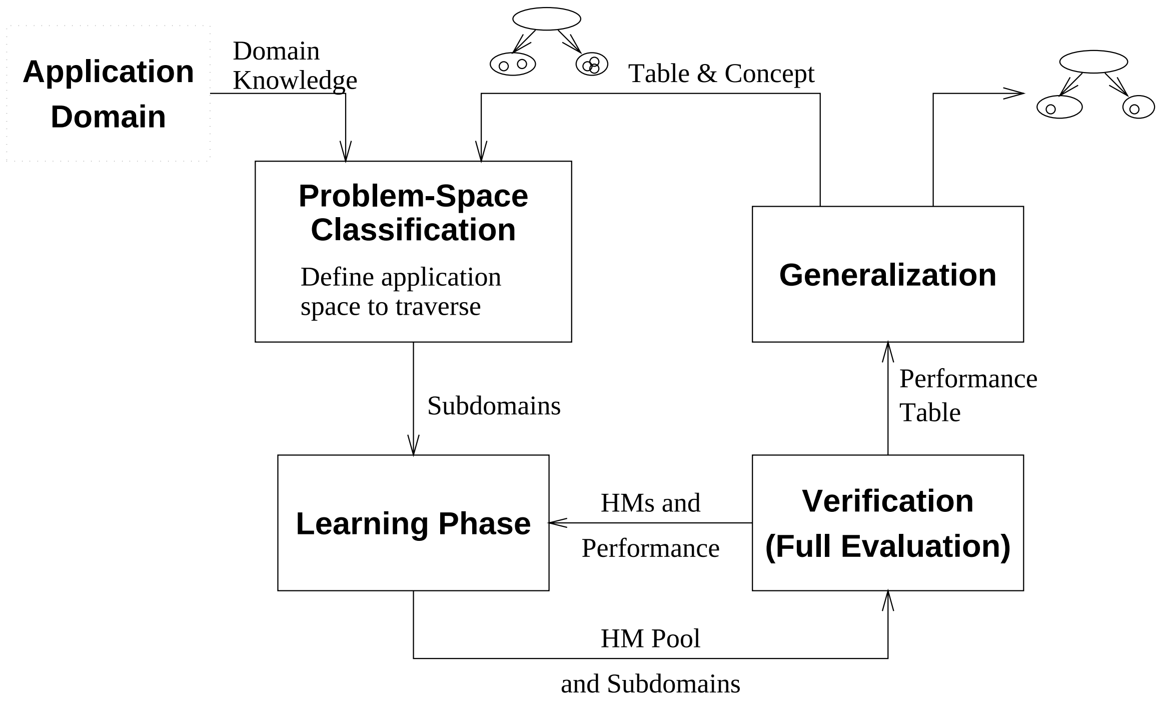


Figure 3.5: Organization of the heuristics-design process in TEACHER.

b) Subdomain classification. For a problem subspace, we need to partition it into subdomains so that the performance of HMs in each subdomain can be statistically estimated based on a subset of test cases within each subdomain. As we have seen in Section 2.5, the performance of HMs cannot be compared or combined directly across subdomains in a learning experiment.

Learning phase

In the *learning phase*, the goal is to find effective HMs for each of a limited set of subdomains. For each subdomain, the learning phase must operate under resource constraints. The tasks in the learning, verification, and generalization phases are shown in Figures 3.6 and 3.7, respectively.

To perform learning, the system first selects a subdomain, generates good HMs (or uses existing HMs from users or previous learning experiments) for this subdomain, and schedules

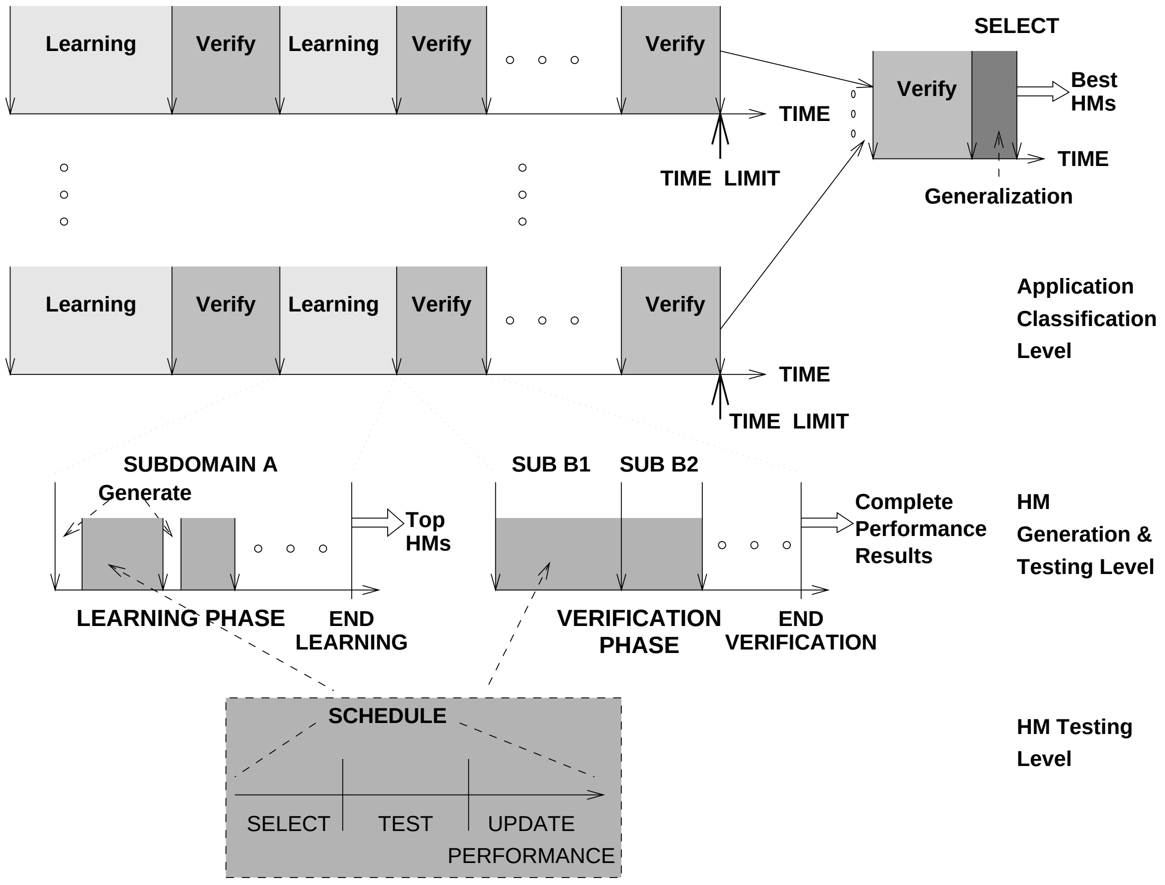


Figure 3.6: Overall learning, verification, and generalization process.

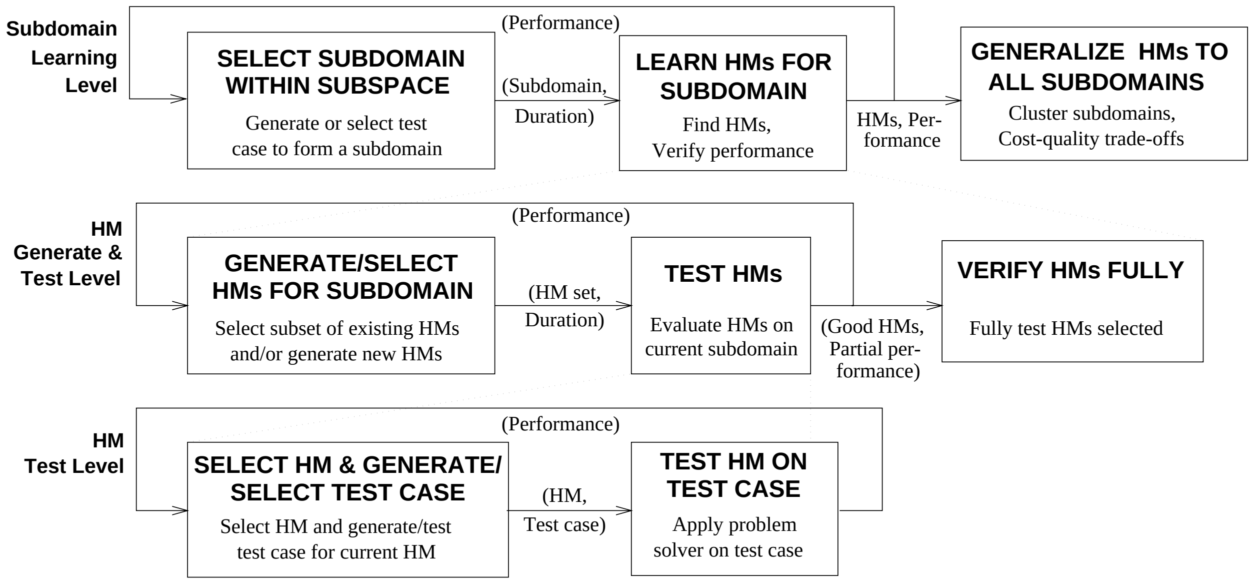

Subdomain Learning Level
(Performance)
SELECT SUBDOMAIN WITHIN SUBSPACE
Generate or select test case to form a subdomain
(Subdomain, Duration)
LEARN HMs FOR SUBDOMAIN
Find HMs, Verify performance
HMs, Performance
GENERALIZE HMs TO ALL SUBDOMAINS
Cluster subdomains, Cost-quality trade-offs
HM Generate & Test Level
(Performance)
GENERATE/SELECT HMs FOR SUBDOMAIN
Select subset of existing HMs and/or generate new HMs
(HM set, Duration)
TEST HMs
Evaluate HMs on current subdomain
(Good HMs, Partial performance)
VERIFY HMs FULLY
Fully test HMs selected
HM Test Level
(Performance)
SELECT HM & GENERATE/ SELECT TEST CASE
Select HM and generate/test test case for current HM
(HM, Test case)
TEST HM ON TEST CASE
Apply problem solver on test case

Figure 3.7: The learning, verification, and generalization actions in TEACHER.

tests of the HMs based on the available computational resources. Learning phase is the only phase in the design process where new HMs are introduced. This phase is also the most complicated phase within the design process. When learning is completed, the resulting HMs need to be fully verified, as HMs obtained during learning may not be tested adequately. Note that learning is performed on one subdomain at a time.

There are three key issues in this phase.

a) Heuristics generation. This issue entails the generation of good HMs given the performance information of "empirically good" HMs. As discussed in Section 3.2, we use weak generation operators from a genetics-based machine-learning approach here [10, 29].

b) Performance evaluation. This problem is related to the evaluation of the performance of HMs during learning, given that there may be multiple performance measures, that there is no defined relationship among them, and that HMs may have different performance across different subdomains (Chapter 2).

c) Resource scheduling. The issues here are on the selection of HMs for further testing, the termination of the current generation, and the initiation of the next generation, given performance information of HMs under consideration. These problems are important when limited computational resources are available and tests of HMs are expensive and noisy. We schedule computational resources rationally by choosing (i) the number of tests on each HM, (ii) the number of competing HMs to be maintained at any time, and (iii) the number of problem subdomains to be used for learning and for generalization. We study in Chapter 4 two related problems in resource scheduling: *sample allocation* and *duration scheduling.*

Verification phase

The goal of the *verification phase* is to obtain more complete performance information about the set of HMs with the highest performance at the end of the previous learning phases. As mentioned previously, HMs obtained during learning may not be tested adequately and the performance of each HM during the learning phase is usually based on incomplete data over a subset of test cases for the target subdomain. This incomplete information is not enough for selecting the best and generalized HMs during the generalization phase. The incomplete performance information is remedied through full evaluation of each HM selected at the end of each learning phase during each verification phase.

In addition, the final generalization phase for evaluating the selected HMs over the entire problem domain requires performance information for every HM on every subdomain. A verification phase can evaluate each selected HM fully on all subdomains from all learning phases and any additional subdomains provided by the users. The main potential issue in the verification phase is in the scheduling of this full evaluation process.

Generalization phase

The last phase is the *generalization phase* whose goal is to generalize the performance of the HMs learned in one or more learning phases to cover the entire problem domain, including subdomains that may not have been used during the design process. There are two key issues to be studied.

a) Performance of HMs across different subdomains. As discussed in Section 2.5, HMs may have different performance behavior in different subdomains; hence, these values cannot be combined directly. A definite result is possible only for the simplest and most ideal case in which one HM has the best performance in all subdomains. For other cases, only heuristics can be used to compare different HMs. We present in Chapter 5 a heuristics method to evaluate the performance of HMs for a group of subdomains.

b) Cost-quality trade-offs. This issue involves determining efficient HMs that perform well in the application. Should there be multiple HMs to be applied (at a higher total cost and better quality of results), or should there be one HM that is costly to run but generates high-quality results? These issues are studied in Chapter 6 when we present experimental results on learning new HMs for several applications.

3.5 The Classification Phase

The purpose of the classification phase is to partition the target problem domain of a target application into smaller subsets in order to (a) identify different regions within the problem domain that require different types of HMs to solve efficiently and (b) make sure that performance values from test cases used during each learning phase are representative and can be used for statistical estimation of the true performance of unseen test cases.

Within an application domain, different regions of the problem domain may have different characteristics, each of which can best be solved by a unique HM [26]. Since learning is difficult when test cases are of different behavior and it is necessary to compare HMs quantitatively, we need to decompose the problem domain into smaller partitions before learning begins. In the following we define a problem subspace and a problem subdomain.

Problem subspace

A *problem subspace* is a user-defined partition of a problem domain so that HMs for one subspace can be learned independently of HMs in other subspaces. Subspace partitioning is important when test cases in an application have vastly different behavior. However, in some cases, it may not be possible to define the attributes needed for partitioning or the number of attributes may be too large. When this happens, nonparametric clustering methods, such as those based on neural networks, may have to be used. Another possibility is to always apply multiple HMs for each test case, resulting in a higher computational cost for a better solution.

We use two examples from Chapter 2 to demonstrate the idea of problem subspaces.

Example 3.1 For instance, consider solving a vertex-cover problem (see Example 2.2). In designing a decomposition HM to decide which vertex to be included in the covered set, previous experience on other optimization problems indicates that HMs for densely connected graphs are generally different from HMs for sparsely connected ones. Consequently, the problem domain of all graphs may be partitioned (in an ad hoc fashion) into a small number of subspaces based on graph connectivities and learned independently. ■

Example 3.2 As another example, in generating test patterns for VLSI circuits, previous experience shows that sequential circuits require tests that are different from those of combinatorial circuits. As a result, we can partition the problem domain into two subspaces. However, we are not able to partition the subspace of sequential circuits into smaller subspaces as it is not clear which attributes should be used in this partitioning. ■

Problem subdomain

We have already defined a different type of partitioning of problem domains in Section 2.5. A *problem subdomain* in this thesis is a subset of the problem domain (or problem subspace) whose performance statistic satisfies the *independent and identically distributed* (IID) property over the entire subset. The reason for this partitioning is to allow statistical estimation of the performance of HMs in a subdomain, which is not possible across subdomains.

In the same way that test cases are partitioned into subspaces, minimal domain knowledge should be used in knowledge-lean applications to partition test cases into subdomains. In addition, we have identified in Section 2.4.3 several statistical tests for testing the IID property. These tests can be used in conjunction with domain knowledge to form problem subdomains.

Continuing with the example on the vertex-cover problem, we discover in Example 2.12 that a problem subdomain can be defined as random graphs with a certain degree of connectivity. As another example, in generating test patterns for testing VLSI circuits, we may have to treat each circuit as an individual subdomain, as we do not know the best set of attributes to classify circuits (see Example 2.13).

Current strategy

Currently, subspace classifications must be guided by expert or domain knowledge based on past performance information. Such partitioning is generally guided by commonsense knowledge or by user experience in solving similar application problems. It requires knowing one or more attributes to classify test cases and is driven by a set of decision rules that identify the subspace to which a test case belongs. In most cases, we simply assume that the entire problem domain is one subspace.

Subdomain classifications are currently performed by users. Statistical tests from Section 2.4.3 can be used to determine (a) whether a given set of performance data is IID and (b) whether different sets of IID performance values (based on test cases with different attributes) belong to the same problem subdomain. These results can then be used to form subdomains. Minimal domain knowledge should be used in this process.

3.6 Architecture for Learning Heuristics in One Subdomain

In this section, we present our approach to learn new HMs under resource constraints for a single subdomain. Figure 3.8 shows the architecture of our resource-constrained learning system for one subdomain [51]. This population-based learning system is based on the genetics-based machine-learning paradigm. There are five main components in the system:

(a) *Resource Scheduler*, which decides on the best way to use the available resources,

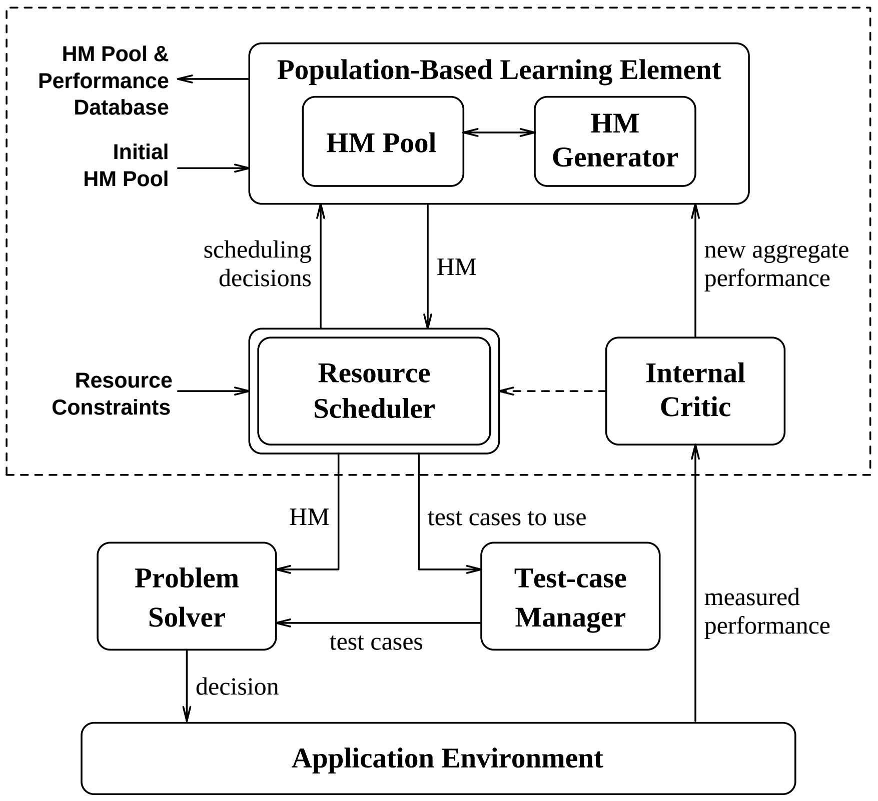


Figure 3.8: Architecture of our population-based learning for one subdomain.

(b) *Internal Critic*, which provides feedback based on the performance measured to indicate how well a particular HM has performed,

(c) *Population-Based Learning Element*, which generates new HMs and maintains a pool of existing ones and their past performance,

(d) *Test-Case Manager*, which generates and maintains a database of test cases used in HM evaluation, and

(e) *Problem Solver*, which evaluates an HM using a test case.

In this research, we assume that the application-specific Problem Solver and Test-Case Manager are user-supplied. The remaining three components are designed to deal with the three key issues presented in Section 3.4: heuristics generation, performance evaluation, and resource scheduling.

3.6.1 Problem Solver

The *Problem Solver* component is simply the target problem solver whose heuristics we want to improve. The purpose of the problem solver is to solve a particular test case using its heuristics. The performance of applying the problem solver on a test case is in terms of the quality of the solution found and the cost of the problem-solving process.

In our learning strategy, the problem solver must be able to accept (a) the specification of the HM to be used during the problem-solving process and (b) the test case to be solved. There must also be a mechanism for the measured performance of the problem-solving process using the specified HM on a specific test case to be fed back into the learning system.

3.6.2 Test-Case Manager

The purpose of the *Test-Case Manager* is to provide test cases to be used in learning. These test cases are either generated randomly or retrieved from a database of stored test cases.

In our current implementation, the Test-Case Manager simply selects from a predefined sequence of a user-supplied pool of test cases. When a test case is requested for a particular

HM, the Test-Case Manager returns the first test case within the sequence that has not yet been evaluated by the chosen HM, which means that different HMs are evaluated over the same set of test cases. This fact simplifies performance comparison between different HMs.

3.6.3 Population-Based Learning Element

The *Population-Based Learning Element* maintains a pool of active HMs. At the end of each generation, a new set of HMs is generated to replace existing HMs. Several top active HMs are usually retained along with the new HMs while other HMs are removed from the active pool.

In this research, the Population-Based Learning Element generates new HMs using weak domain-independent operators, such as crossover, mutation, and hill-climbing. These are traditional operators in genetic algorithms for generating new HMs [10, 31]. The process for selecting existing HMs for reproduction is also the same as in traditional genetics-based machine learning.

More advanced methods that usually require additional domain knowledge are left for future study. They are currently not necessary because the application domains in which we are interested are knowledge lean, a fact which makes it very difficult to find a powerful method for generating new HMs. In addition, a more powerful HM generator is likely to be dependent on the targeted application domain.

3.6.4 Internal Critic

The *Internal Critic* normalizes the performance value of each test case tested by a candidate HM against the performance value of the same test case evaluated by the baseline HM. It then updates the performance metrics of the candidate HM. Note that this is similar to updating the fitness values of HMs in classifier-system learning.

We have extended the performance-evaluation process for two HMs proposed in Chapter 2 to deal with the general case with more than two HMs. We have chosen to use a fixed baseline HM during each learning phase and compare different HMs based on their estimated average normalized performance. The baseline HM should be the best existing HM that is the target of improvement whenever possible. There are some potential anomalies due to the usage of a fixed baseline that will be explained in Chapter 5. However, the ordering of HMs does not have to be perfect during a learning phase, and a certain degree of uncertainty is acceptable. In addition, a fixed baseline HM allows constraints in multi-objective optimization to be applied in a uniform fashion.

In general, the Internal Critic performs credit assignment [106] that apportions credit and blame on HDEs using results obtained in testing (see Figures 1.1 and 3.3). Credit assignments can be classified into temporal credit assignment (TCA) and structural credit assignment (SCA). TCA is the first stage in the assimilation of feedback and precedes SCA during learning. TCA divides up feedback between current and past decisions. Methods for TCA depend on whether the state space is Markovian: non-Markovian representations often require more complex TCA procedures. On the other hand, SCA translates the (temporally local but structurally global) feedback associated with a decision point into modifications associated with various parameters of the decision process.

In knowledge-lean applications that we consider in this thesis, a world model that relates states, decisions, and feedback signals generated by the learning system or measured in the environment is missing. As a result, credit assignment has a much weaker influence on performance improvement. An example of such a TCA algorithm is the bucket-brigade algorithm in classifier-system learning [10]. Note that the lack of a world model for credit assignment is the main reason for maintaining competing HMs in our learning system.

3.6.5 Resource Scheduler

The *Resource Scheduler* schedules tests of HMs based on the available computational resources. Note that scheduling is critical when tests are computationally expensive. There are two main problems in scheduling during each learning phase.

The *sample-allocation problem* involves the scheduling of tests of HMs in a generation, given a fixed numbers of tests in the generation and HMs to be tested. This problem is known in statistics as the (sequential) *allocation problem* [107, 108] and the scheduler is known as the *local scheduler.*

The *duration-scheduling problem* involves deciding when to terminate an existing generation and start a new one. The part of the resource scheduler that deals with this problem is known as the *global scheduler.*

These two related problems, sample allocation and duration scheduling, as well as the scheduling of tests under multiple performance objectives, are studied in the next chapter.

3.7 The Verification Phase

Within the verification phase, we want to find more complete performance information about the HMs we have generated during the learning phase(s). The generalization phase requires performance information on every subdomain used in the heuristics-design process from each HM selected. However, the performance information obtained during each learning phase pertains to only one subdomain and is usually incomplete even for that subdomain due to resource constraints. In the verification phase, we have to evaluate the HMs we consider good in more detail, which is accomplished through verification of each HM on all problem subdomains under consideration.

The operations within the verification phase (shown in Figure 3.9) are very similar to the operations in the learning phase except for a few differences. First, there is no generation of new HMs in the verification phase, and a fixed pool of HMs is maintained. Second, more than one subdomain of test cases can be used by the Test-Case Manager. Third, performance from different subdomains is dealt with separately and independently by the Internal Critic. Fourth, the resource scheduling problem is different from the one in the learning phase.

In the verification phase, a fixed set of HMs is evaluated on test cases from several different subdomains. The general issue in resource scheduling is in selecting a proper subset of HMs and in allocating the proper amount of time to each subdomain. The goal is to minimize uncertainties in the performance of all HMs across all subdomains. At present, we evaluate each HM in the selected subset fully on each problem subdomain under consideration. This

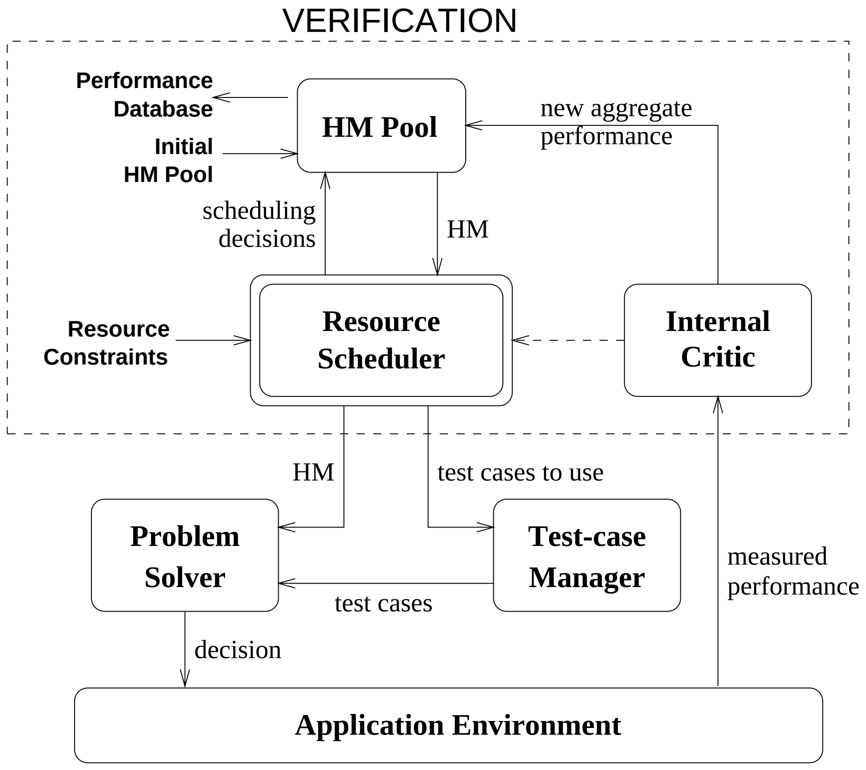


Figure 3.9: General model for verification of HMs learned.

process is known as *full evaluation*, which requires no scheduling. Future work can extend the strategy by considering resource constraints in the full evaluation process.

Full evaluation ensures that all HMs have the same amount of performance information such that performance comparisons among HMs can be "fair." Assuming that each subdomain π_i has a finite set I_i of test instances, full evaluation ensures that when subdomain π_i is used in a verification phase, all HMs are tested over the entire set I_i of test cases.

3.8 The Generalization Phase

Based on information in Section 2.5, a problem domain (or problem subspace) can contain many problem subdomains. Each of these subdomains have different performance behavior and cannot be combined or compared directly. Although good HMs for a particular subdomain can be developed during each learning phase, the objective of the design process is to find a good HM for the entire problem domain.

Generalization is the process of finding a good HM for solving a randomly chosen test case in a problem domain so that this HM has a high probability of performing better than other competing HMs for solving this test case. Since there are in general a large (and possibly infinite) number of subdomains in a problem domain, this generalization process is critical in the overall heuristics-design process.

Example 3.3 To illustrate the concepts presented in this section, we show in this example the average symmetric speedups of four decomposition HMs used in a branch-and-bound search to solve vertex-cover problems. (The use of symmetric speedup is defined in Eq. (2.8).) We treat all test cases as belonging to one subspace, and graphs with the same degree of connectivity are grouped into a subdomain. We apply genetics-based learning to find the five best HMs for each of three subdomains with connectivities 0.1, 0.35, and 0.6.

Figure 3.10 shows the performance of the best HMs learned in each subdomain across all subdomains. We have also identified a single generalized HM among the fifteen HMs learned and show its performance in Figure 3.10. We find that the generalized HM is not the top HM learned in each subdomain, indicating that the best HM in each subdomain may be too specialized to the subdomain. We have also found that generalization is possible in terms of average performance. We must point out that the average performance should not be used as the sole indicator, as performance variances may differ from one subdomain to another. ■

The generalization process is difficult because (a) performance from different subdomains must be treated separately and independently and (b) there are usually many more subdomains within the target problem domain (or problem subspace) than the ones used in the heuristics-design process. Consequently, each subdomain performance must be treated independently. A definite result can then be reached only for the ideal case of having one HM that performs better than all other selected HMs in all given subdomains. For all other

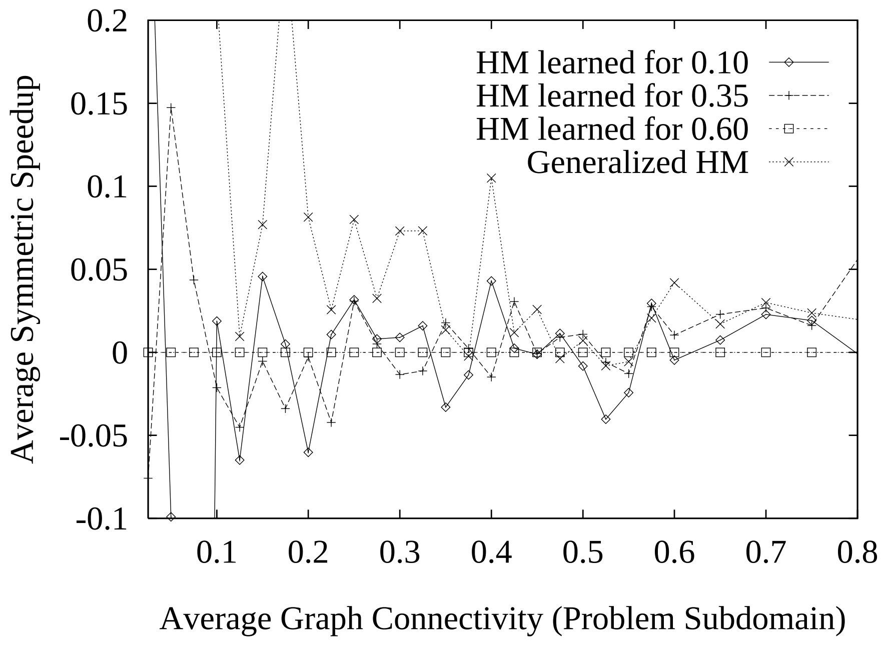


Figure 3.10: Average symmetric speedups (over 15 test cases) of three decomposition HMs for the vertex-cover problem, where subdomains are manually selected based on graph connectivity. The HM learned for 0.6 connectivity is the same as the baseline HM.

cases, the final result must be based on some heuristics and should be left for the user to decide.

It is still possible to provide an educated guess in distinguishing some of the better HMs within the selected set. We have developed some generalization heuristics for ordering HMs based on the available performance information so that the user's task can be simplified. We discuss generalization in detail and present some of these heuristics in Chapter 5.

In some situations, multiple HMs may have to be identified and applied together at a higher cost to find a solution of higher quality. We discuss this issue in Chapter 6.

3.9 Summary

In this chapter, we have examined issues involved in the heuristics-design process and have developed a systematic framework for addressing these issues in our heuristics-design problems.

There are five major issues in designing new heuristics.

1. *Decomposition and Integration of Problem Solvers.* This issue involves breaking down complex problem solvers into smaller and more manageable components so that a good HM can be designed for each component. It also involves integration of HMs from various components together so that the final problem solver is as efficient as possible.

2. *Classification of Problem Domains.* This issue involves partitioning a target problem domain into smaller subdomains so that performance data within each subdomain are IID. This IID property allows performance of HMs to be statistically estimated based on a subset of performance values.

3. *Generation of New Heuristics.* This issue involves the generation of new HMs based on past performance information with the objective of getting "better" HMs.

4. *Evaluation of Heuristics.* This issue involves finding and comparing the performance of various HMs.

5. *Generalization of Heuristics to Unseen Test Cases.* This issue involves identifying the HM that is most likely to give the best performance over the entire problem domain, based on available performance information.

Our target application problems are characterized as having no world model to relate heuristics specifications to its performance (knowledge lean) and having nondeterministic performance with expensive tests.

Based on our survey of previous work in machine learning, we have selected to use *genetics-based machine learning*, a *population-based* approach. This method is based on applications of an evolution and natural selection process to machine-learning problems and has its root in the field of *genetic algorithms*. It is also a part of a larger field called evolutionary computing.

TEACHER, our system for designing new HMs, divides the operations in the overall design process into four distinct phases: classification, learning, verification, and generalization. Each phase addresses different design issues.

1. The *classification phase* involves partitioning the problem domain into subdomains so that the performance behavior within each subdomain is IID. Domain knowledge and statistical tests for the IID property (see Section 2.4.3) can be used in this process. This phase must be done before the start of the remaining phases.

2. The *learning phase* involves developing better HMs for one particular subdomain under resource constraints. The implementation of this phase is mostly based on traditional genetics-based machine learning.

There are three main operations within this phase:

(a) generation of new HMs based on traditional genetics-based reproduction methods, such as crossover and mutation,

(b) performance evaluation (similar to fitness identification in genetic algorithms) based on the strategy presented in Chapter 2, and

(c) scheduling of HMs for evaluation to optimize system performance based on limited resources.

The issues involved in resource scheduling and our strategies are presented in Chapter 4.

3. The *verification phase* involves obtaining more accurate and more complete performance information for the best HMs from each learning phase. This verification is necessary since performance information about each HM during the learning phase is usually based on a partial subset of test cases for a single subdomain. In contrast, the generalization phase requires accurate information about the performance of each HM over multiple subdomains.

4. The *generalization phase* involves selecting from among the learned HMs the best HM over the entire problem domain. This process is difficult because (a) performance information about each HM is incomplete even after the verification phase and (b) performance from different subdomains must be treated separately and independently. Our strategy for generalization is presented in Chapter 5.

4. RESOURCE SCHEDULING IN GENETICS-BASED LEARNING

In this chapter, we study the problem of resource scheduling in genetics-based learning. There are two issues to be addressed in resource scheduling in population-based learning: (a) allocation of resources among active HMs, known as the *sample-allocation* problem and (b) deciding the proper time to start a new generation by generating a new set of HMs, known as the *duration-scheduling* problem.

The most common sample-allocation strategy is the static *round-robin* strategy that allocates an equal amount of resources to each HM. We have developed a *nonparametric minimum-risk* strategy that takes advantage of available performance information without making any assumptions on performance distribution. In addition, when certain assumptions on performance distributions are met, we have also developed a *minimum-risk* strategy that can take advantage of this information.

We have also developed a dynamic duration-scheduling strategy operating under multiple performance objectives. This strategy, known as *DMDS*, is designed to avoid the condition where all HMs are eliminated from consideration when each HM violates some or all of the constraints imposed on various performance objectives. DMDS uses an iterative refinement approach to develops HMs for harder and harder constraints until all target constraints are met. Ideally, the amount of time spent to achieve each successive set of constraints should increase in a geometric fashion.

Table 4.1: Symmetric speedups of the best HMs based on three different schedules and 150 tests. HMs in each run are selected randomly from a pool of 100 HMs. The average speedup is evaluated using 15 randomly generated test cases.

Schedule	Sub-domain	Sym-Speedup of the Best HM		
		Run 1	Run 2	Run 3
10 HMs of 15 tests each	D_A	-0.56	-2.08	0.01
	D_B	-0.22	-0.16	-0.01
75 HMs of 2 tests each	D_A	0.01	0.01	0.01
	D_B	-0.03	-0.03	-0.03
20 HMs of 5 tests each; the 5 best HMs 10 times each	D_A	0	-0.56	0.01
	D_B	0	-0.16	-0.01

4.1 Introduction

Resource scheduling strategies are used for scheduling tests of HMs during each learning phase, based on the available computational resources. In population-based learning, multiple active HMs are maintained during learning. The resource scheduler chooses either to continue learning by allocating available resources among active HMs or to generate a new set of HMs to be evaluated. Resource scheduling of tests in learning is crucial when tests are expensive and computational resources are limited.

Example 4.1 To illustrate the importance of scheduling, consider the testing of HMs in the vertex-cover problem discussed in Chapter 2. Suppose we have identified two subdomains: D_A (with graph connectivity of 0.1) and D_B (with graph connectivity of 0.6). To illustrate the effect of scheduling, we randomly generated 100 decomposition HMs and evaluated each on D_A and D_B.

Table 4.1 shows the average symmetric speedups of HMs selected under three resource schedules with respect to those of the conventional HM. The results show that (a) there are trade-offs between the number of HMs tested and the performance of the best HM found and (b) more detailed evaluation of several top HMs at the end of learning is beneficial. ■

In this chapter, we discuss two problems in resource scheduling and their solutions in our population-based learning system: sample allocation and duration scheduling. The *sample-allocation problem* involves the scheduling of tests of HMs in a generation, given a fixed number of tests in each generation and the HMs to be tested. The *duration-scheduling problem* involves deciding when to terminate an existing generation and to start a new one.

In the rest of this chapter, we discuss our model and assumptions on the sample-allocation and duration-scheduling problems, issues on designing resource scheduling strategies, and our proposed scheduling algorithms. We also present some methods for estimating parameters of performance distributions required by various dynamic strategies in order to increase the efficiency of these strategies. Finally, we evaluate our resource scheduling strategies by applying them on some real-world applications.

4.2 Model and Assumptions

We describe in this section a statistical model for scheduling tests in our learning system. A general comprehensive model is too complex to be analyzed since many parameters are unknown *a priori*. Here, we find good scheduling strategies based on a simplified model and apply these strategies as heuristics in practice. We then evaluate empirically these strategies on the original general model.

We assume that the performance values of HM_a over a problem subdomain constitute a population with a distribution $f_a(x)$. Each evaluation of HM_a is equivalent to drawing a performance value from the distribution. We make the following assumptions in our study.

- In multi-objective applications, we assume that there are $k+1$ performance measures $J_1, ..., J_k, J_0$. The original objective is assumed to be maximizing the normalized performance for each of these measures. We then transform measure J_i, $i = 1, \ldots, k$, into a *constrained measure* by imposing a minimum level constraint, θ_i, for each constrained measure J_i (see Section 2.3). The objective is then for each HM to satisfy the constraints for J_i, $i = 1, \ldots, k$, and to maximize J_0, the *unconstrained measure* to be optimized.

- We define $\mu_{a,i}$ and $\hat{\mu}_{a,i}$, respectively, as the expected normalized performance and the estimated sample-mean performance for HM_a on performance measure J_i. Without loss of generality, we may use μ_a and $\hat{\mu}_a$ to refer to $\mu_{a,0}$ and $\hat{\mu}_{a,0}$, respectively. Here, $f_a(x)$ is the distribution of performance measure J_0 for HM_a.

- Distribution function $f_a(x)$ is generally unknown and nonidentical for different HM_a's, and tests drawn from $f_a(x)$ may be dependent. In our simplified analysis, we assume that samples drawn from $f_a(x)$ are IID for each HM_a.

- The means of populations belong to a distribution that is hard to estimate. Further, crossovers and mutations applied at the end of a generation may change this distribution in an unknown fashion. For simplicity, we ignore this effect in our simplified model.

- We assume that the time to evaluate one test case using one HM is one unit. That is, we consider only logical time in our scheduling study.

4.3 Sample-Allocation Problem

Sample allocation entails the scheduling of tests of HMs in a generation, given a fixed number of tests in the generation and the set of HMs to be tested. This problem is known in statistics as the (sequential) *allocation problem* [107, 108] and the scheduler is known as the *local scheduler.*

In this section, we first briefly summarize previous work in this area, which includes the *round-robin* strategy, the most commonly used sample-allocation strategy. We then describe two dynamic sample-allocation strategies we have developed based on different assumptions: the *minimum-risk* and the *nonparametric minimum-risk* strategies.

4.3.1 Previous work

The original sequential allocation problem suggested by Bechhofer [107] is to decide the optimal allocation of picks, given a fixed total number of picks, assuming that the population mean and variance are known. The objective of these strategies is to maximize $P(CS)$, the probability of correctly selecting the population with the highest population mean when time is expended. Optimal solutions to problems in this class are unknown, and many extensions have been proposed to accommodate various trade-offs and relaxed assumptions.

Existing sample-allocation strategies can be classified into static and dynamic.

Static sample-allocation strategies have a selection sequence fixed ahead of time, independent of the values of the picks obtained during selection. They are easier to analyze due to their simplicity. The most commonly used static strategy is the *round-robin* strategy.

The *round-robin* strategy takes samples from each population in turn. It allocates T/n tests to each population, given T tests and n population, while maximizing the worst-case $P(CS)$ when all populations have the same variance [109]. Its drawback is that it tests the worst population to the same extent as the best, an obviously inefficient way of using resources. This drawback is important when the amount of resources available is limited.

This strategy is also the most commonly used strategy in genetics-based learning systems [3, 10, 29]. It is the benchmark to be tested against when any new strategies are developed.

Dynamic (or adaptive) sample-allocation strategies select the population for testing based on previous sample values and other run-time information. Although more flexible, they are usually more complicated. One such strategy was developed by Tong and Wetzell to optimize $P(CS)$ when the selection process ends. It focuses on populations with high sample means but also tests others with smaller means if they were not tested enough [108].

Sample-allocation strategies developed in statistics are not directly applicable in our learning system because they were developed with different objectives. In statistical sample allocation, the objective is to maximize $P(CS)$, given a finite number of populations. In contrast, our objective is to maximize the expected population-mean value of the population selected, given infinitely many populations initially. Since the maximum number of tests in learning is limited, we are interested in how close the actual performance of the selected HM is to the maximum performance within a pool of HMs.

We have previously developed a *minimum-risk* scheduling strategy [42, 91], which is a dynamic sample-allocation strategy with the above objective in mind. This strategy is presented in Section 4.3.2. In our derivation, we assume that the distribution of each population is normal with a common variance, an obviously restricted assumption for many applications.

4.3.2 Minimum-risk sample-allocation strategy

The *minimum-risk* strategy is a dynamic allocation strategy we have previously developed for minimizing the difference in performance of the selected HM from the maximum [42]. This strategy is designed for allocating resources during the learning phase and is based on decision theoretic methods [110, 111]. Its goal is to minimize the expected value of a predefined loss function (or *risk*). We have chosen to define the loss function in terms of the estimated error between the true and the estimated performance level of the best population.

In deriving the strategy, we assume that the distribution of each population (HM_i), $f_i(x)$, is normal with population mean μ_i and variance σ_i^2 ($N(\mu_i, \sigma_i^2)$).

Let K be the number of populations (HMs). Population i (HM_i) is characterized by μ_i, its population mean; σ_i, its population standard deviation; $\hat{\mu}_i$, its sample mean; $\hat{\sigma}_i$, its unbiased sample standard deviation; and n_i, the number of samples tested.

We define the loss function for population i, L_i, using the squared-error function:

$$L_i \;=\; L(\mu_i, \hat{\mu}_i) \;\equiv\; (\mu_i - \hat{\mu}_i)^2 \;=\; Var[\mu_i - \hat{\mu}_i] \;=\; \frac{\sigma_i^2}{n_i} \,. \tag{4.1}$$

The variance of $\hat{\mu}_i$, σ_i^2/n_i, can be estimated by $\tilde{\sigma}_i^2 \;\equiv\; \hat{\sigma}_i^2/n_i$.

The expected estimation loss (or risk R) due to selection of the population with the highest sample mean is then expressed as

$$R \;=\; E[L(\mu_{(best)}, \hat{\mu}_{(best)})] \;=\; \sum_{i=1}^{K} P_i^* E[L(\mu_i, \hat{\mu}_i)] \;, \tag{4.2}$$

where P_i^* represents the probability that population i has the best population mean based on the current information.

For our case in which each population is normally distributed, we can calculate P_i^* as

$$P_i^* \;=\; \int_{-\infty}^{\infty} \left[\prod_{j \neq i} \Phi \left(\frac{\mu - \hat{\mu}_j}{\tilde{\sigma}_j} \right) \right] \phi \left(\frac{\mu - \hat{\mu}_i}{\tilde{\sigma}_i} \right) d\mu \;, \tag{4.3}$$

where Φ and ϕ represent the *cdf* (cumulative distribution function) and *pdf* (probability density function) of the standard normal distribution, respectively.

The scheduling objective can then be formulated as

$$\text{minimize} \quad R \qquad \text{subject to } \sum_{i=1}^{K} n_i \;=\; N \;, \tag{4.4}$$

where N is the total number of tests of all populations at the end of the current generation.

By applying Lagrange multiplier λ, we obtain the following equations:

$$\frac{\partial}{\partial n_i} \left[\sum_{i=1}^{K} P_i^* \tilde{\sigma}_i^2 \;-\; \lambda \sum_{i=1}^{K} n_i \right] \;=\; 0 \;,\; i = 1, \ldots, K \;. \tag{4.5}$$

Since $\partial P_i^* / \partial n_i = 0$, it follows that

$$P_i^* \frac{\partial \tilde{\sigma}_i^2}{\partial n_i} \;-\; \lambda \;=\; 0 \;,\; i = 1, \ldots, K \;. \tag{4.6}$$

The optimality criteria can then be stated as follows:

$$-\lambda \;=\; P_i^* \tilde{\sigma}_i^2 \;=\; P_j^* \tilde{\sigma}_j^2 \;,\; \text{for } i \neq j \;. \tag{4.7}$$

At any time t, the strategy is to minimize Eq. (4.2) for time $(t+1)$, *i.e.*, only one of n_i can be increased by one.

$$\textbf{ONE-STAGE POLICY:} \qquad n_j{}' \;=\; n_j + 1 \quad \text{when } \max_i P_i^* \tilde{\sigma}_i^2 = P_j^* \tilde{\sigma}_j^2 \;. \tag{4.8}$$

In the usual case, we assume that the value P_i^* is changing slowly and $P_i^{*\prime} \approx P_i^*$.

Since the calculation of P_i^* requires complex numerical integration, further simplification is sometimes desirable to reduce its complexity. We approximate P_i^* by w_i^*, the probability that population i is better than the current best population (or second best, in case that HM_i is the best). Let i be the permutation index where $\hat{\mu}_{(i)}$ is the i^{th} estimated mean and $\hat{\mu}_{(K)}$ is the largest $\hat{\mu}$. Then w_i^* is defined as

$$\begin{aligned} w_i^* &= \int_{-\infty}^{\infty} \Phi\left(\frac{\mu - \hat{\mu}_j}{\tilde{\sigma}_j}\right) \phi\left(\frac{\mu - \hat{\mu}_i}{\tilde{\sigma}_i}\right) d\mu \\ &= \Phi\left(\frac{\hat{\mu}_i - \hat{\mu}_j}{\sqrt{\tilde{\sigma}_i^2 + \tilde{\sigma}_j^2}}\right) \quad , \qquad \text{where } j = \begin{cases} (K) & \text{for } i \neq (K) \\ (K-1) & \text{for } i = (K)\ . \end{cases} \end{aligned} \tag{4.9}$$

4.3.3 Bayesian estimation

In this subsection, we present a method to improve the performance of the minimum-risk scheduling strategy by getting more accurate estimations of σ_i and μ_i when more information about each population is available.

When we have prior knowledge about the distribution of the population mean, $f_\mu(x)$, we can improve the computation of its risk value. Assume that we know the distribution of the population means of population i as $h_i(\mu_i)$, where $h_i(\mu_i)$ is known as the *prior distribution* of population i and represents *a priori* knowledge about the performance of the population *before* it is tested.

We make the following assumptions.

1. The distribution of each population (HM_i), $f_i(x)$, is normal with population mean μ_i and variance σ_i^2 ($N(\mu_i, \sigma_i^2)$).

2. The distribution of the population mean, f_μ or h_i, is normal with population mean μ_α and population variance σ_α^2 ($N(\mu_\alpha, \sigma_\alpha^2)$).

After population i has been evaluated n_i times to get a sequence of performance values $< S_i >$, with sample mean $\hat{\mu}_i$ and sample variance $\hat{\sigma}_i^2$, we can use its prior distribution to get a more accurate estimate of its performance distribution using Bayes' theorem [112]:

$$h_i{}^*(\mu_i| < S_i >) = \frac{f_{S_i|\mu_i}(< S_i > |\mu_i)h_i(\mu_i)}{\int f_{S_i|\mu_i}(< S_i > |\mu_i)h_i(\mu_i)d\mu_i} , \tag{4.10}$$

where $f_{S_i|\mu_i}$ is a pdf of S_i with mean value μ_i. The *posterior distribution*, $h_i^*(\mu_i| < S_i >)$, represents the belief or knowledge about the value of μ_i *after* population i has been tested, given the prior distribution, $h_i(\mu_i)$.

In our case, the posterior distribution of population i will be

$$h_i^*(\mu| < \hat{\mu}_i, n_i >) \sim N\left(\frac{n_i\hat{\mu}_i + \alpha_i\mu_\alpha}{n_i + \alpha_i}, \frac{\sigma_i^2}{n_i + \alpha_i}\right) , \alpha_i = \frac{\sigma_i^2}{\sigma_\alpha^2} \tag{4.11}$$

$$\mu_i^* = \frac{n_i\hat{\mu}_i + \alpha_i\mu_\alpha}{n_i + \alpha_i} , \sigma_i^{*2} = \frac{\sigma_i^2}{n_i + \alpha_i} . \tag{4.12}$$

The Bayesian means and variances are more accurate than the sample means and variances because prior knowledge is used. However, they require that the given prior knowledge is correct, *i.e.*, h_i is correct.

We can substitute μ_i^* and σ_i^{*2} in place of $\hat{\mu}_i$ and $\tilde{\sigma}_i^2$ in calculating the risk values when the assumptions are met. This substitution would provide a more accurate risk value when h_i is correct and also allows the risk to be computed even after only one test has been performed.

Note that the computation of Bayesian means and variances requires the values of μ_α, σ_α^2, and σ_i^2 to be known. If we make an additional assumption that population variance, σ_i^2, is identical for all populations ($\sigma_i^2 = \sigma^2$ for all i), then we can estimate the values of μ_α, σ_α^2, and σ^2 at the start of each generation. We show two methods for estimating these parameters in Section 4.5.

4.3.4 Nonparametric minimum-risk sample-allocation strategy

In general, we may not have information on the distributions of performance measures since they change dynamically and are difficult to estimate. In this section, we propose a nonparametric sample-allocation strategy for determining HMs to be evaluated based on run-time information of population performance. Our *nonparametric minimum-risk* strategy is extended from the parametric minimum-risk method.

The objective of resource scheduling is to find the best HMs when all resources are exhausted. However, this objective cannot be achieved in general since we cannot model changes in distributions between generations. To cope with this problem, we restrict our objective on sample allocation within a generation to the following objective: *minimize the risk that the populations selected for generating new HMs when the generation ends are wrong.* Note that this objective is for scheduling *within* a generation, but *not across* generations.

Consider a generation of K populations (HMs). Population i (HM_i) is characterized by information such as n_i (number of tests performed), μ_i (unknown population mean), σ_i^2 (unknown population variance), $\hat{\mu}_i$ (sample mean), $\hat{\sigma}_i^2$ (sample variance), F_i (true fitness value), and f_i (sample fitness value), where $F_i \equiv \mu_i - c$, $f_i \equiv \hat{\mu}_i - c$, and c is a constant that is usually set to be the minimum fitness value of all of the populations. Note that f_i is an unbiased estimator of F_i since $\hat{\mu}_i$ and S_i^2 are unbiased estimators of μ_i and σ_i^2, respectively.

We define the loss due to believing f_i as $L_i \equiv E[(f_i - F_i)^2]$. Given $\hat{\mu}_i$ and σ_i (or $\hat{\sigma}_i$), we can calculate the value of L_i, noting that $E[(f_i - F_i)^2] = Var[f_i - F_i]$, as

$$L_i = \frac{\sigma_i^2}{n_i} . \tag{4.13}$$

The probability that population i will be selected for generating new populations is defined as $P_i \equiv f_i / \sum_{j=1}^{K} f_j$ [27]. The scheduling problem can be formulated (heuristically) as follows:

$$\begin{aligned} \text{minimize } \phi \equiv & \sum_{i=1}^{K} P_i L_i \\ \text{subject to } & \sum_{i=1}^{K} n_i = N , \end{aligned} \tag{4.14}$$

where N is the number of tests performed in the current generation. By applying Lagrange multipliers, we have

$$\text{minimize } \hat{\phi} \equiv \phi + \lambda \left(\sum_{i=1}^{K} n_i - N \right) . \tag{4.15}$$

By equating $\partial\hat{\phi}/\partial n_i$ to zero, we have the optimality criteria as follows:

$$-\lambda = \frac{P_i \sigma_i^2}{n_i^2} = \frac{P_j \sigma_j^2}{n_j^2} , \text{ for } i \neq j . \tag{4.16}$$

At any time t, the strategy is to minimize Eq. (4.14) for time $(t+1)$; *i.e.*, only one of the n_i's can be increased by one.

$$\textbf{ONE-STAGE POLICY:} \quad n_j \leftarrow n_j + 1 \quad \text{where} \quad \max_i \frac{P_i \sigma_i^2}{n_i^2} = \frac{P_j \sigma_j^2}{n_j^2} . \tag{4.17}$$

Eq. (4.17) says that the population to be tested is one that has a large fitness value (*i.e.*, a large probability of being chosen for reproduction in the next generation) and has large variance (*i.e.*, large uncertainty in its mean). Note that Eq. (4.17) only tries to find the next population to be tested. In this case, P_i generally changes slowly ($P_i' \approx P_i$) and can be approximated using information in the current generation.

In our experiments, we use $\hat{\sigma}_i$ as an approximation to σ_i. To estimate $\hat{\sigma}_i$, at least two tests must be performed on each population (preferably four tests or more). Note that the Lagrange multiplier procedure is valid only when performance values are continuous values.

Example 4.2 As an example, consider population i with four samples: 0.971, 1.006, 0.988, and 1.055. In this case, $\hat{\mu}_i$, S_i, and n_i are 1.005, 0.036, and 4, respectively. Assuming that there are a total of 30 populations and that c is 0.910, then the fitness value of population i is $\hat{\mu}_i - 0.910 = 0.095$. Further, assume the current total fitness of the remaining 29 populations to be 0.781. Hence, P_i is $0.095/(0.781 + 0.095) = 0.108$, and the risk value of this population is $P_i\hat{\sigma}_i^2/n_i^2$, $(= 8.78 \times 10^{-6})$.

Assume that population i has the largest risk value and that a new sample with value 0.920 is drawn from it. With this new sample, $\hat{\mu}_i$, $\hat{\sigma}_i$, n_i, f_i, and P_i become 0.988, 0.049, 5, 0.078, and 0.091, respectively, and the new risk value is reduced to 7.59×10^{-6}.

This example shows that populations with high mean (hence, high P_i) and high $\hat{\sigma}_i^2/n_i$ are more likely to have high risk values and be tested. Generally, risk values are reduced as more samples are drawn. ■

This nonparametric minimum-risk strategy can still be applied when additional information about various distributions is available. In fact, Bayesian estimations (μ_i^* and σ_i^{*2}) can be used in place of $\hat{\mu}_i$ and $\hat{\sigma}_i^2/n_i$ if the assumptions on distribution information are met.

4.4 Duration-Scheduling Problem

Duration scheduling entails deciding when to terminate an existing generation and start a new one. The resource scheduler dealing with this problem is the *global scheduler*. A common strategy is to allocate a fixed duration to each generation, although better decisions can be made if past information is used.

In this section, we first present a brief survey of previous work on duration scheduling. We then present duration-scheduling methods for multi-objective applications. As discussed in Section 2.3, we must constrain all but one objective and optimize the unconstrained objective.

4.4.1 Previous work

Duration-scheduling strategies can be classified as *static* and *dynamic*. A *static (or fixed) duration-scheduling strategy* simply sets the duration of each generation to a predetermined value. This is the most common duration-scheduling strategy in the field of evolutionary computing [3, 10, 29]. Previous work [92, 104] has shown that the most appropriate duration is dependent on the total time allocated to learning and the target application. To find a proper duration size for a given time limit, experiments with different durations must be run. The overhead for this approach is deemed too high to be useful.

A *dynamic (or adaptive) strategy*, on the other hand, uses run-time information to determine when each generation should end. A new set of HMs should be generated when the expected improvement from the new HMs is larger than the expected improvement from further testing the current set of HMs. There is very little research on this problem in statistics. One strategy we have studied extends our minimum-risk sample-allocation strategy [42] by estimating the distribution of new populations to be generated in the next generation using statistics collected in previous generations [91,105]. (In the first generation, samples have to be drawn to estimate the initial distribution.) The strategy is restrictive because it assumes that all populations have normal distributions with the same variance.

Another dynamic strategy we have studied is based on Bayesian analysis [91,105], which results in a strategy that increases the duration size as the variance ratio (ratio of sample variance to the variance of the μ_i's) decreases. When the variance of the μ_i's is large, it is easy to identify good populations; hence, the duration should be small. In general, the variance ratio is large when learning begins and decreases as learning proceeds. Consequently, the duration size should be small initially and should increase gradually. The difficulty with the Bayesian strategy is that it is difficult to find the correct duration size without making simplifying assumptions on the distributions.

Instead of varying the duration size, a dual strategy is to fix the duration of a generation but vary the number of populations in it [42,92,93]. This strategy is less flexible because it is difficult to adjust the population size dynamically.

The main shortcoming of existing work is that it assumes that HMs generated always have acceptable performance, even though most HMs may be pruned after a few tests. This pruning is especially true in multi-objective applications in which we set constraints on performance measures, and there may not be any acceptable HMs at the end of a generation. We address this problem in Section 4.4.3.

4.4.2 Constraint handling for multi-objective optimization

Based on the strategy we have developed in Section 2.5.2 for the case with two HMs, we outline in the following subsection a method for determining the likelihood that an HM satisfies the given constraints using notations defined in Section 4.2. It is not possible to prune every HM_a violating one or more constraints ($\mu_{a,i} > \theta_i$) on one or more test cases because (a) $\mu_{a,i}$ is unknown and the estimated $\hat{\mu}_{a,i}$ is used instead, (b) there is uncertainty in determining $\hat{\mu}_{a,i}$, and (c) it is not possible to set worst-case performance bounds of an HM on a test case because by the nature of heuristics, their worst-case behavior may not be bounded.

We want to penalize HMs based on P_{ok}, the *probability of satisfying the given set of constraints*. Since the problems we study have high evaluation cost, we need to prune HMs that are unlikely to satisfy the constraints ($P_{ok} \ll 0.5$). Further, we would like to give a higher chance for further reproduction and testing to HMs with P_{ok} close to one.

Given the performance values of an HM, HM_a, over n_a test cases with sample mean $\hat{\mu}_{a,i}$ and sample variance $\hat{\sigma}^2_{a,i}$ for each constrained measure J_i, random variable $(\hat{\mu}_{a,i} - \mu_{a,i})\sqrt{n}/\hat{\sigma}_{a,i}$ has Student's t-distribution with $n-1$ degrees of freedom [44]. Accordingly, we can compute the probability that this HM satisfies the threshold value θ_i on J_i as

$$P(\mu_{a,i} \geq \theta_i) \;=\; F_t\left(n_a - 1, \frac{\hat{\mu}_{a,i} - \theta_i}{\hat{\sigma}_{a,i}/\sqrt{n_a}}\right) \;, \tag{4.18}$$

where $F_t(\nu, x)$ is the cdf of Student's t-distribution with ν degrees of freedom.

When there are multiple constrained measures, the probability that all constraints (θ_i for $i = 1, \ldots, k$) are satisfied is equal to $P(\mu_{a,1} \geq \theta_1 \cap \ldots \cap \mu_{a,k} \geq \theta_k)$. Based on probability theory, we know that

$$\prod_i P(\mu_{a,i} \geq \theta_i) \;\leq\; P_{ok} \;=\; P(\mu_{a,1} \geq \theta_1 \;\cap\; \ldots \;\cap\; \mu_{a,k} \geq \theta_k) \;\leq\; \min_i \; P(\mu_{a,i} \geq \theta_i)\,. \tag{4.19}$$

Hence, we use $\min_i \; P(\mu_{a,i} \geq \theta_i)$ as an approximation to P_{ok}.

4.4.3 Dynamic multi-objective duration-scheduling (DMDS) strategy

Due to constraints under multi-objective optimization, all HMs may be pruned during learning when constraints are too tight. Applying random generation at that point is not helpful because random generation is the weakest of all generation methods, and it is unlikely that newly generated HMs will satisfy the constraints.

To avoid this undesirable scenario, we must relax our original goal and find HMs that are as close as possible to the desired level of constraints, given the available resources. To this end, we must first start with loose constraints and gradually tighten them as learning proceeds.

Using the relaxed goal, the learning system iteratively finds HMs using increasingly harder constraints instead of trying to find HMs that immediately satisfy the final target constraints. The initial set of constraints is selected in such a way that almost all randomly generated HMs will be accepted. These easy constraints ensure that some HMs will be available for generating new HMs in the next generation. To set constraints in successive iterations, we apply an iterative refinement method that we have developed in a real-time search algorithm for solving time-constrained combinatorial optimization problems [113]. We set new thresholds so that the times used in learning with successive thresholds grow in a geometric fashion. In this way, a small portion of the total time is used in all intermediate iterations, and most of the effort is spent in the last iteration.

Using this iterative method, we must set intermediate thresholds on constrained measures J_i , $i = 1, \ldots, k$ that are easier to achieve than the final target thresholds. We define the j^{th} set of intermediate thresholds on k performance measures as $\hat{\theta}_{1,j}, \hat{\theta}_{2,j}, \ldots, \hat{\theta}_{k,j}$. Since we want increasingly tougher constraints, we have the following property:

$$-\infty \;=\; \hat{\theta}_{i,0} \;<\; \hat{\theta}_{i,1} \;<\; \ldots \;<\; \hat{\theta}_{i,\infty} \;=\; \theta_i \; i = 1, \ldots, k \;, \tag{4.20}$$

where θ_i is the final target threshold for J_i, and $\hat{\theta}_{i,0}$ is the initial threshold at the start of learning.

To control both the duration of each generation and the values of intermediate thresholds, we have developed the DMDS strategy. This strategy has two stages: (a) when not all target thresholds are satisfied ($\exists J_i$ with $\hat{\theta}_{i,j} < \theta_i$) and (b) when all target thresholds have been reached ($\hat{\theta}_{i,j} = \theta_i$ for all i).

Stage 1: $\exists J_i$ with $\hat{\theta}_{i,j} < \theta_i$ (not all target thresholds are satisfied). In this stage, DMDS decides the time for updating constraints and the values of the thresholds. It starts a new generation and a new set of thresholds when HMs satisfying the current constraints have been found and most HMs violating the current constraints have been eliminated. DMDS determines new thresholds based on profile data collected on thresholds during learning and the amount of time spent in finding acceptable HMs using these thresholds. The thresholds are set so that the time spent in each iteration to find feasible HMs that satisfy the constraints grows geometrically.

Stage 2: $\hat{\theta}_{i,j} = \theta_i$ for all i (all target thresholds have been reached). When all of the performance constraints have been satisfied in a generation, the learning system finds the best HM that satisfies all of the constraints. To do so, good HMs found in this generation are tested more thoroughly to ascertain if they satisfy the constraints to within a high degree of certainty before the next generation begins.

We defer until Section 4.6 to show the performance of the various scheduling algorithms discussed in this section.

4.5 Distribution-Estimation Methods

For all dynamic resource scheduling strategies we have presented, some information on population variance is required. The population (or sample) variance is used in the minimum-risk strategy (both with and without Bayesian estimation) and in the nonparametric minimum-risk strategy to compute the risk function. DMDS requires the variance of the constrained measures in computing P_{ok}.

The sample variance to estimate a population variance is usually inaccurate until *at least four* performance values are available, which means that every HM, even the worst one, must be tested at least four times. This minimum testing is extremely inefficient since the worst HM can usually be identified after just one or two tests.

For Bayesian estimation to be applied, additional information about the distribution of the population means (μ_α and σ_α^2) is required assuming that the distribution is $N(\mu_\alpha, \sigma_\alpha^2)$.

In this section, we show a method for estimating these parameters (σ_i^2, μ_α, and σ_α^2) at run time with little overhead so that each population does not have to be evaluated at least four times. Our methods require the assumption that all population variances must be identical, *i.e.*, $\sigma_i^2 = \sigma_i$, the common variance, for all i. We can still use this method as a heuristic by using the estimated common variance for all populations that have not been tested. For each well-tested population with six or more tests, its sample variance can be calculated and used directly.

First, we present statistical equations for computing these values from some subset of performance values. Then, we present two possible ways to acquire these performance values at run time and the cost associated with each.

4.5.1 Statistical-distribution estimation

We have made the following assumptions:

1. The distribution of each population (HM_i), $f_i(x)$, is normal with population mean μ_i and common variance σ^2 ($N(\mu_i, \sigma^2)$).
2. The distribution of population mean, f_μ or h_i, is normal with population mean μ_α and population variance σ_α^2 ($N(\mu_\alpha, \sigma_\alpha^2)$). This assumption is necessary only for estimating μ_α and σ_α^2.

Let K be the number of populations that have been tested. For HM_i, let $S_{i,j}$ be the performance value obtained for test case t_j, n_i be the number of tests performed, and $\hat{\sigma}_i^2$ be the sample variance. Let N be the total number of tests performed on the entire set of populations, $\sum_{i=1}^{K} n_i$.

First, we estimate the common variance, σ^2, using the following equation [114]:

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{K}(n_i - 1)\hat{\sigma}_i^2}{\left(\sum_{i=1}^{K} n_i\right) - K}, \text{ where } (n_i - 1)\hat{\sigma}_i^2 = \sum_{j=1}^{n_i}(S_{i,j} - \hat{\mu}_i)^2 . \quad (4.21)$$

Variable $(\sum_{i=1}^{K} n_i - K)\hat{\sigma}^2/\sigma^2$ has χ^2 distribution with degrees of freedom $(\sum_{i=1}^{K} n_i - K)$.

The estimation of μ_α and σ_α^2 is the same as in a two-factor analysis [114]. When $n_i = n$ for all populations, the distribution of the $\hat{\mu}_i$'s is $N(\mu_\alpha, \sigma_\alpha^2 + \sigma^2/n)$. Therefore, μ_α and σ_α^2 can be estimated as

$$\begin{aligned} \hat{\mu}_\alpha &= \frac{1}{K}\sum_{i=1}^{K} \hat{\mu}_i \ , \\ \hat{\sigma}_\alpha^2 &= \frac{1}{K(K-1)}\left(K\sum_{i=1}^{K}(\hat{\mu}_i)^2 - \left(\sum_{i=1}^{K}\hat{\mu}_i\right)^2\right) - \frac{\hat{\sigma}^2}{n} \ . \end{aligned} \tag{4.22}$$

4.5.2 Pre-sampling

The simplest method for estimating the distribution of population means and population variances is to allocate some time (usually a certain percentage, such as 10 %, of the total time) at the start of each generation to test a subset of populations (four or more populations) in the pool several times (three or more tests). We can then get the estimate for σ^2, μ_α, and σ_α^2 using the equations described in the previous section. This process has to be repeated in every generation since the distribution of the population means is changing in an unknown fashion in each generation.

With this approach, we can get a good estimate of the population variances and the distribution of the population means in every generation. However, the method is very costly. First, it is likely that poor performing populations are tested too many times, in addition to testing populations that performed well. Since some percentage of time within each generation is spent doing pre-sampling, less time is available for evaluating other populations in the pool. This behavior is undesirable since the time required during pre-sampling can be quite large in order to get the necessary information. Hence, this method should not be applied in every generation.

4.5.3 Dynamic update

Within a generation, it is necessary that each population be evaluated at least once before applying the dynamic allocation strategy. Otherwise, there is not enough information for the resource scheduler to use in dynamic scheduling. Hence, we can always evaluate each untested population once at the start of a generation and use this information to estimate μ_α and σ_α^2.

However, since each population only provides one sample, σ^2 cannot be estimated. In addition, the computation of σ_α^2 requires the value of σ^2. Because of this, it is necessary for the value of σ^2 to be estimated differently before this method can be used. The simplest method is to apply pre-sampling during the first generation to estimate σ^2 and then use dynamic updates in the remaining generations. With this approach, there is only minor overhead for pre-sampling in the first generation. The only potential problem is that $\hat{\sigma}^2$ from the pre-sampling may not be accurate, and there is no way to recompute the value during learning.

4.6 Experimental Results

In this section, we show some experimental results for comparing various resource scheduling strategies. These results are collected in applying TEACHER to two real-world applications. The first target problem solver is the post-game analysis (PGA) system for mapping processes on a distributed-memory multicomputer system (see Section 6.1). The second target problem solver is the branch-and-bound search for solving a vertex-cover problem (see Section 6.2).

Process mapping on distributed-memory multicomputers

There are two performance measures for post-game HMs: quality of a mapping (process completion time of the mapping) and cost of finding the mapping (PGA execution time).

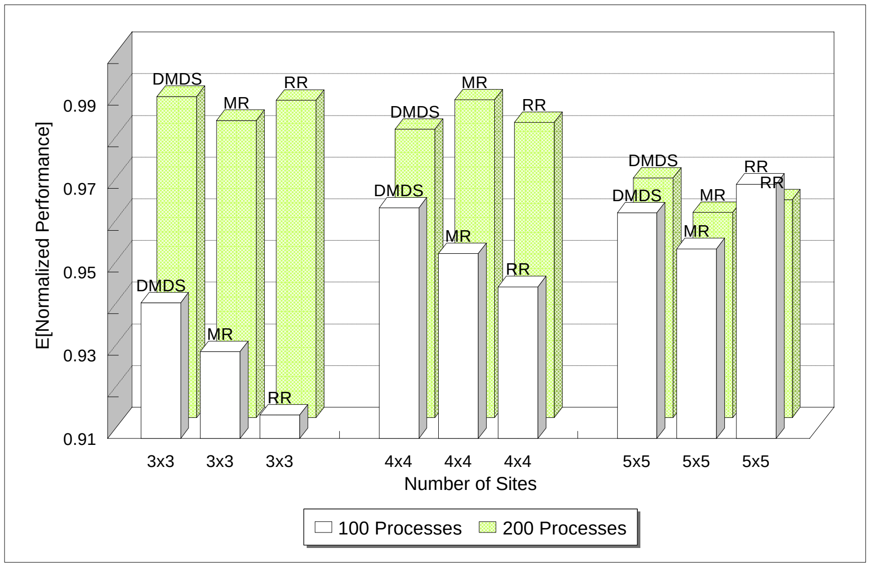


Figure 4.1: Average quality of HMs selected under a cost constraint of 30% of the original PGA HM for three resource scheduling strategies: DMDS, MR, and RR. (The number of HMs retained for full verification is five.)

Learning aims to find PGA HMs with the minimum completion time and cost within a user-specified limit. This objective is necessary since PGA has to be run concurrently with the application program, and a new mapping should be proposed within a time constraint.

We have evaluated three resource scheduling strategies: DMDS with minimum-risk sample allocation, fixed duration with minimum-risk sample allocation (MR), and fixed duration with round-robin sample allocation (RR). For each strategy, we performed five learning experiments of 800 tests over each of the six subdomains, using a cost constraint of 30% of the cost of the original PGA HM by Yan and Lundstrom [18, 40]. We studied six cases of retaining 1, 3, 5, 10, 15, and 20 HMs at the end of learning for full verification and compared the best and average qualities of the HMs achieved over five runs of each scheduling strategy.

Figure 4.1 shows the average speedups (quality) of HMs achieved by the three scheduling strategies over the six subdomains. Since the cost constraint is tight and all performance values represent slowdowns, we do not use symmetric speedups here. All of these HMs have significantly lower costs and slightly worse qualities than those of the original PGA HM.

Further, DMDS performs the best in four out of six subdomains and the second best in the two other subdomains. The other results are similar, with smaller differences in quality as the number of HMs retained for verification increases.

DMDS also consistently finds acceptable HMs more often than the two fixed-duration scheduling strategies when the cost constraint is tight. Of the 30 experiments performed under a 30% cost constraint, DMDS failed once when the five best HMs were retained for verification (RR failed five times and MR failed four times). When the cost constraint is loose, DMDS does not perform better than the other scheduling strategies.

Branch-and-bound search

There is one objective measure for a branch-and-bound search. The normalized cost of an HM is defined in terms of its *average symmetric speedup* (see Eq. (2.8)), which is related to the number of nodes expanded by a B&B search using the baseline HM and by a B&B search using the new HM.

Our experiments here study the effects of fixed-duration strategies (RR and MR) on learning. DMDS was not used because there is only one objective measure. Figure 4.2 shows the quality of HMs found as a function of learning time for the two scheduling strategies. Each point on the graph was obtained by averaging the symmetric speedups of the HMs selected as if learning had been stopped at that point.

Figure 4.2 shows that MR outperforms RR, that MR is more likely to identify the top HM, and that MR requires fewer HMs to be retained for full verification at the end of learning. For this reason, we have used the fixed-duration MR scheduling strategy in the remaining results in this thesis. Figure 4.2 also shows that verifying more HMs at the end leads to better HMs (albeit a higher verification cost).

4.7 Summary

In this chapter, we have examined the two main actions in resource scheduling in genetics-based machine learning: sample allocation and duration scheduling.

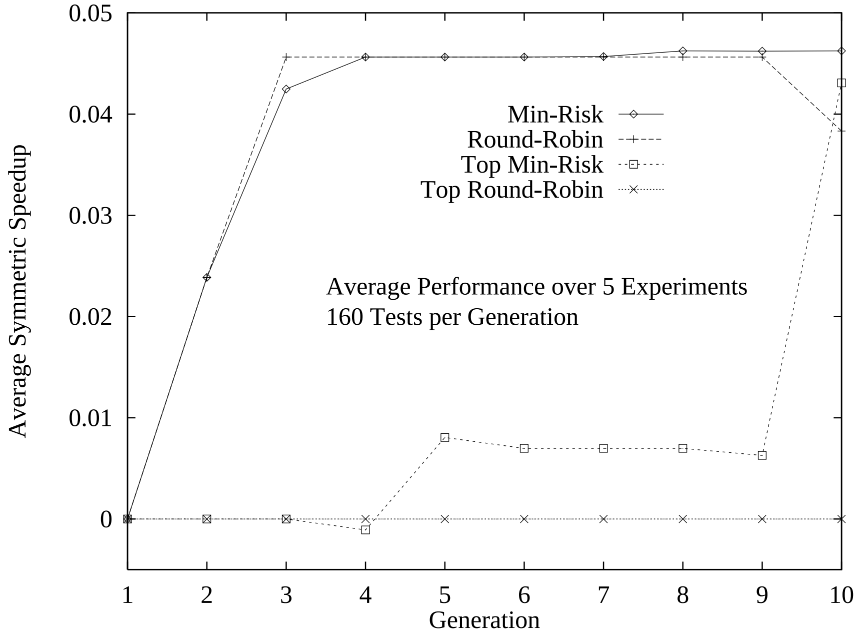


Figure 4.2: Average performance over five runs of the HMs selected for VC problems with a degree of connectivity of 0.15 using the MR and RR scheduling strategies. (Top-RR and top-MR mean that only one HM was retained for verification; in RR and MR, five HMs were retained.)

The *sample-allocation problem* involves the scheduling of tests of HMs in a generation, given a fixed number of tests in each generation and a given pool of HMs to be tested. The most common sample-allocation strategy is the static *round-robin* strategy that allocates an equal amount of resources to each HM. We have developed two dynamic sample-allocation strategies that can take advantage of available performance information. The *nonparametric minimum-risk* strategy does not require any assumption on performance distribution. The *minimum-risk* strategy requires that the performance distribution of each HM be Gaussian (normal).

A dynamic strategy (preferably nonparametric minimum-risk) should be used unless the performance variance of each HM is very high or the objective is not based on the average normalized performance. In these cases, the round-robin strategy should be used.

The *duration-scheduling problem* involves deciding when to terminate an existing generation and to start a new one. The most common sample-allocation strategy is to use a fixed duration for the entire learning process. We have developed a dynamic strategy, *DMDS*, for dealing with the case when there are constraints imposed by multiple performance objectives. This strategy uses iterative refinement to find acceptable HMs. Constraints are tightened until the target constraints have been reached. This iterative approach is intended to avoid having all HMs pruned due to constraint violations.

When dealing with multi-objective optimization and constraints imposed on these objectives, DMDS should be used unless the constraints are relatively easy to satisfy. In all other cases, fixed-duration scheduling should be used.

5. STATISTICAL GENERALIZATION OF HEURISTIC METHODS LEARNED

In this chapter, we study the *statistical generalization process* for determining the performance of a given set of HMs found in the heuristics-design process over the entire problem domain. Note that the performance of HMs found in learning is incomplete as HMs are tested on a small subset of test cases. The objective of generalization is to *first find a subset of HMs that performs better than the current incumbent HM over the entire problem domain, followed by selecting the best HM among this subset.*

When a problem domain contains different regions with different statistical performance behavior, we propose to partition this domain into *subdomains* so that performance within each subdomain is IID (independent and identically distributed). The generalization process can then be divided into two parts: generalization of performance within each subdomain and generalization of performance across all subdomains.

Within each subdomain, all performance values should be normalized with respect to that of the incumbent HM as the baseline. The sample-mean value of each HM in a subdomain can be used as a statistical estimate of its true average performance over the entire subdomain due to the IID property. An HM can be considered better than the incumbent HM in a subdomain when all constraints are satisfied ($P_{ok} > 0.5$), and the average performance for the unconstrained optimization objective is better than the baseline performance with a certain degree of certainty ($\hat{\mu} > BL$ and $P_{win} > 0.5 + \Delta$).

Across the entire problem domain, the performance from different subdomains must be treated independently and equally. The performance of each HM over the entire problem domain can be represented by its worst-case performance over the selected set of subdomains. The HM that provides the best worst-case P_{win} and that satisfies all constraints is the most likely HM to be better than the incumbent HM. When there is no acceptable incumbent HM, the objective can be transformed into finding the HM with the best worst-case average normalized performance for the unconstrained optimization objective.

5.1 Introduction

The term "generalization" usually refers to the process of making or choosing an element that is more general than the elements before the start of the process. In the context of the heuristics design, generalization is the process of *finding or estimating the performance of a given set of HMs over the entire problem domain based on incomplete performance information found during the design process.* This generalization is necessary and important since only an incomplete subset of test cases is used for performance evaluation during the heuristics-design process. This generalization concept is shown in Figure 5.1.

The primary objective of the generalization process is to find the HM that performs the "best" over the entire problem domain among the given set of HMs. In general, there is also a secondary objective. When it is not possible to distinguish the "best" HM with respect to an existing HM already in use, it is desirable to find HM(s) that consistently "outperform" this existing HM over the entire problem domain.

The generalization process is simplified when the performance of each HM shares common statistical characteristics, such as the *independent and identically distributed (IID)* property, for the entire problem domain. In this case, performance over a subset of test cases can be statistically generalized to represent the performance of the entire problem domain. For example, the sample-mean performance from a subset of test cases can represent and predict the population mean over the entire problem domain. However, this IID condition is usually not true in complex real-world applications. There are usually many different regions within the problem domain with different performance behavior. In this situation, we propose to

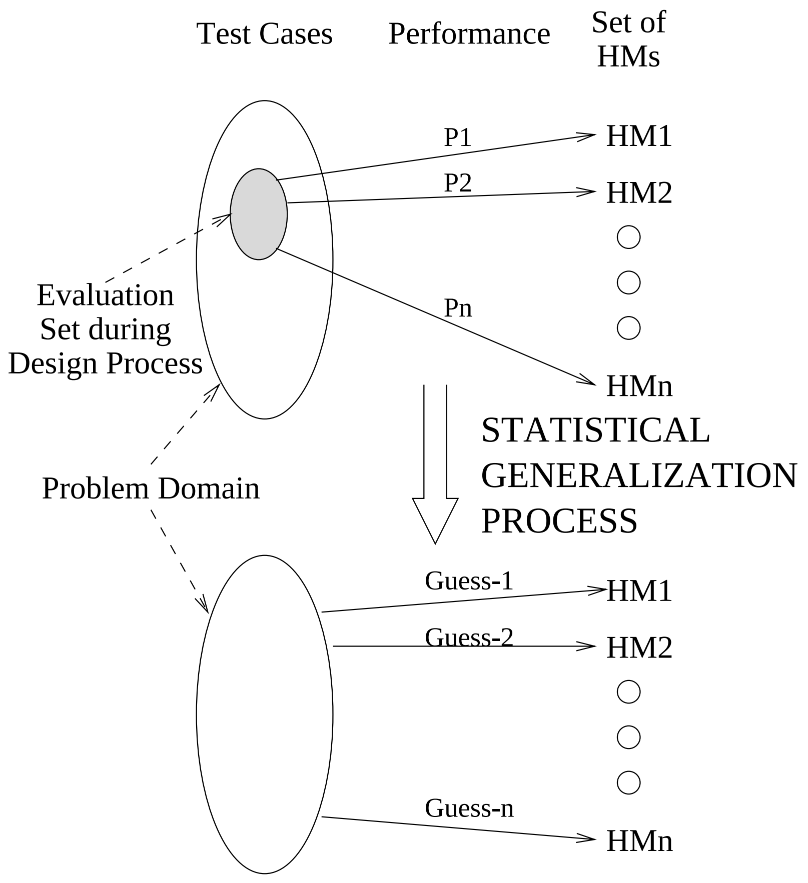


Figure 5.1: The concept of statistical generalization.

divide the problem domain into smaller subsets called *subdomains* so that the performance of each HM within each subdomain is IID.

The generalization process with multiple subdomains in the target problem domain is more difficult because performance from different subdomains has different statistical characteristics and cannot be combined or compared. Performance evaluation for each subdomain must be dealt with separately and independently from other subdomains.

For example, in testing two heuristic methods HM_1 and HM_2 on two sets of test cases TC_1 and TC_2, we find the average performance for HM_1 to be $\{10, 100\}$ and for HM_2 to be $\{150, 5\}$. It will be difficult to say whether HM_1 is better than HM_2 and which HM should be used as a general HM for other test cases.

When there are many more subdomains than what can be evaluated during the heuristics-design process, generalization is even more difficult. Unfortunately, this is the case for many applications.

The problem of generalization with multiple subdomains can be divided into two parts. First, the performance of each HM within each subdomain must be determined and compared. Second, the performance over the entire problem domain must be determined or estimated based on the performance indicators by which each HM is evaluated in a problem subdomain.

In Chapter 2, we have developed a strategy for *statistical performance evaluation* of two HMs. This strategy can be applied to address both parts of the generalization process when only two HMs are considered. In this chapter, we extend the *statistical generalization process* to the more general case when there are more than two HMs. When there is no clear winner, we propose heuristics that help determine the best HM in our generalization process.

We first present a brief overview of the generalization process commonly used in existing genetics-based machine-learning systems. We then present our extended strategy to handle generalization within one subdomain and over multiple subdomains.

5.2 Previous Work

In heuristics design under "noisy" evaluation data, *i.e.*, when performance of each HM cannot be obtained completely or accurately from a single evaluation, generalization is an important phase of the design process. This condition is true because time for learning is often limited, and only a small set of test cases can be evaluated during learning.

Generalization has not been emphasized in most existing genetics-based machine-learning systems [28, 29, 115]. In general, these systems implicitly assume that all performance values from each HM share common statistical characteristics, *i.e.*, performance values during the design process are representative of *all* possible performance values. In this case, the estimated performance level (*i.e.*, the fitness value) of each HM (usually in the form of the sample mean) can be used as a statistical estimator of its true performance level (population mean) over the entire problem domain. These systems can then simply validate this assumption by evaluating their learned HMs on a new set of test cases.

The inherent uncertainty of the estimated performance is usually ignored by existing genetics-based machine-learning systems. This uncertainty is more acute when performance evaluation is costly and HMs are evaluated less during learning. In Section 5.3, we present a method to deal with this uncertainty.

This implicit statistical generalization is not adequate when there are multiple subdomains and when no single subset of test cases can be representative of the entire problem domain. Implicit statistical generalization is still necessary within each subdomain as shown in Section 5.3. However, it is necessary to perform generalization across multiple subdomains when performance across subdomains is not totally correlated. We explore this issue in Section 5.4.

Note that in this thesis we are dealing with statistical generalization of each HM's performance to get some representative performance over the entire problem domain. We do not modify an HM in order to make it applicable to wider (and more general) conditions. This is the type of generalization studied in artificial intelligence [96], which requires more domain knowledge specific to a particular application and problem solver.

5.3 Performance Evaluation Within One Subdomain

In this section, we first review the performance-evaluation strategy we have developed in Chapter 2 for dealing with two HMs and one subdomain. We then explore the potential problems in extending this strategy to more general cases with more than two HMs. Finally, we present the strategy we have developed for general performance evaluation within a single subdomain.

5.3.1 Strategy for evaluating two HMs

In this subsection, we review the strategy for evaluating the performance of two HMs in one subdomain. This strategy has been developed under the following assumptions:

1. The normalized performance values of each HM within a subdomain are IID.
2. The average performance of each HM is the objective metric to optimize.

The overall strategy can be divided into three steps.

(1) *Normalize* the performance values in order to (a) obtain the relative performance between the two given HMs on each test case and (b) reduce the difference in magnitude of the performance values among different test cases. One of the two given HMs is used as the baseline HM in the normalization process.

To avoid anomalies (inconsistent ordering) due to the choice of the baseline HM, normalization methods used should satisfy the *symmetric-normalization condition* discussed in Chapter 2. The symmetric-improvement method in Section 2.2.3 is an example of such a normalization method.

(2) When there are multiple performance measures corresponding to multiple objectives, *constrain the average performance of all but one objective measure.* This modified goal helps to find the best HM among possible HMs that satisfies all of the constraints based on a single unconstrained objective. With this approach, each performance measure is dealt with independently from other measures.

(3) *Evaluate* the performance of each HM within each subdomain. Due to the IID property in each subdomain, the performance of each HM within a given subdomain can be estimated based on a subset of test cases. The sample-mean values are statistical estimations of the population mean values and can be used in deciding (a) whether an HM satisfies or violates a given set of constraints and (b) whether an HM has higher performance for the single unconstrained optimization objective.

However, there is uncertainty involved in using estimated sample means to make decisions. Probabilistic analysis can be used to determine the degree of certainty involved in using the sample-mean values in a decision-making process. This analysis results in P_{ok} for checking constraint satisfaction and P_{win} for ordering heuristics according to their average performance of the single unconstrained optimization objective. These probabilistic measures can be applied to increase the degree of confidence in the performance estimation in each subdomain.

Assume that in subdomain π, HM_a has the following performance characteristics for performance measure J_i: sample mean $\hat{\mu}_{a,i}$, sample variance $\hat{\sigma}^2_{a,i}$, number of test cases n_a, and baseline performance (*i.e.*, performance of the baseline HM, HM_B), BL_i. From Section 2.5.2, we have derived the following equation for P_{win}:

$$P_{win}(X, J_i) \;=\; F_t\left(n_X - 1, \frac{\hat{\mu}_{X,i} - BL_i}{\sqrt{\hat{\sigma}^2_{X,i}/n_X}}\right) \;, \tag{5.1}$$

where $F_t(\nu, x)$ is the cdf of Student's t-distribution with ν degrees of freedom, and $P_{win}(a, J_i)$ is the probability that the true performance (population mean) of HM_a for performance measure J_i is better than that of the baseline HM, HM_B. When $n \to \infty$, we have

$$P_{win}(X, J_i) \approx \Phi\left(\frac{\hat{\mu}_{X,i} - BL_i}{\sqrt{\hat{\sigma}^2_{X,i}/n_X}}\right) \;, \tag{5.2}$$

where Φ is the standard cumulative normal distribution function [44]. Usually, J_0 is used to denote the single unconstrained optimization measure.

We define $P_{ok}(a)$, the probability that HM_a satisfies a given set of constraints (minimum level constraints θ_j for each constrained measure J_j, $j = 1, \ldots, k$), as follows:

$$P_{ok}(X) \approx \min_j P\left(\mu_{X,j} \geq \theta_j\right) \ , \tag{5.3}$$

$$P(\mu_{X,j} \geq \theta_j) \ = \ F_t\left(n_X - 1, \frac{\hat{\mu}_{X,j} - \theta_j}{\sqrt{\hat{\sigma}^2_{X,j}/n_X}}\right) \ , \tag{5.4}$$

where $F_t(\nu, x)$ is the cdf of Student's t-distribution with ν degrees of freedom.

To take the uncertainty into consideration, an HM is considered in violation of a given set of constraints when P_{ok} is less than $0.5 - \Delta$. During learning, HMs should not be eliminated too easily; hence, Δ can be as high as 0.25. During generalization, more accurate performance information is available; consequently, HMs should be eliminated with $\Delta = 0$.

For ordering HMs based on the unconstrained optimization objective, P_{win} is a measure of certainty in ordering an HM as being better than the baseline HM. In this case, HM_a is considered to have better normalized performance than the baseline HM when $P_{win}(a) > 0.5 + \Delta$ for some $\Delta \geq 0$. A high Δ leads to a higher degree of certainty that HM can satisfy this condition. However, when Δ is too high, an HM that is actually better than the baseline HM may also be eliminated.

5.3.2 Potential performance anomalies with more than two HMs

There are two possible approaches for extending the above performance-evaluation strategy for two HMs to the general case with more than two HMs. The first approach is to perform pair-wise comparison for every possible pair of HMs. The second approach is to compare all HMs based on their normalized performance with respect to a single baseline HM. In this subsection, we present potential problems that can arise with these approaches.

Pair-wise evaluation

With this approach, each pair of HMs can be compared using the strategy described in Section 5.3.1. This approach allows the ordering between each pair of HMs to be determined.

Table 5.1: Summary of raw performance values for four HMs.

Test Case	HM ID			
	HM_{75}	HM_{76}	HM_{86}	HM_{99}
t_1	30.19	30.31	26.21	40.61
t_2	43.12	43.34	34.09	24.65
t_3	71.93	72.49	104.51	98.41

Unfortunately, the orderings are not transitive, which means that when HM_1 is better than HM_2 and HM_2 is better than HM_3, there are no guarantee that HM_1 is better than HM_3. Consequently, there can be cycles in ordering with $HM_1 \rightarrow HM_2 \rightarrow HM_3 \rightarrow HM_1$, where $HM_a \rightarrow HM_b$ denotes HM_a being better than HM_b in a pair-wise comparison. This circular ordering is demonstrated in the following example.

Example 5.1 Table 5.1 shows performance values of four HMs on three different test cases. In this example, an HM represents a computer system, and a test case represents a benchmark program. Each performance value is the time for executing a benchmark program on a computer system. The objective is to minimize this execution time.

Using the symmetric-improvement method (S^{sym+} — see Eq. (2.8)), we can compute the average normalized performance for each HM with different HMs as the baseline. These average normalized performance values are shown in Table 5.2. The baseline performance for this normalization method (BL) is zero.

From this table, we observe that $HM_{75} \rightarrow HM_{86}$ and $HM_{86} \rightarrow HM_{99}$ when the orders are based on the average normalized performance. However, we also observe that $HM_{99} \rightarrow HM_{75}$, creating a cycle, $HM_{75} \rightarrow HM_{86} \rightarrow HM_{99} \rightarrow HM_{75}$, as shown in Figure 5.2. Note that P_{win} for each ordering is close to 0.5, meaning a high degree of uncertainty for each. ■

Due to possible cycles in pair-wise orderings of HMs, it is not always possible to determine the best HM over a subdomain from a given set of HMs. It is also difficult to determine the total ordering of all HMs with this approach.

Table 5.2: Summary of the average normalized symmetric improvement for four HMs with different HMs as the baseline ($BL = 0$).

HM ID	Baseline HM ID							
	HM_{75}		HM_{76}		HM_{86}		HM_{99}	
	$\hat{\mu}$	P_{win}	$\hat{\mu}$	P_{win}	$\hat{\mu}$	P_{win}	$\hat{\mu}$	P_{win}
HM_{75}	0.000	–	0.006	0.981	0.012	0.519	−0.012	0.489
HM_{76}	−0.006	0.019	0.000	–	0.005	0.507	−0.020	0.481
HM_{86}	−0.012	0.481	−0.005	0.493	0.000	–	0.035	0.545
HM_{99}	0.012	0.511	0.020	0.519	−0.035	0.455	0.000	–

$$\begin{tikzcd}
\mathrm{HM}_{75} \arrow[r, "0.012 > 0", "\mathrm{Pwin} = 0.519"'] & \mathrm{HM}_{86} \arrow[r, "0.035 > 0", "\mathrm{Pwin}= 0.545"'] & \mathrm{HM}_{99} \arrow[ll, bend left, "0.012 > 0"', "\mathrm{Pwin} = 0.511"]
\end{tikzcd}$$

Figure 5.2: Example of a cycle in pair-wise ordering of HMs (based on data in Table 5.2).

$$\begin{tikzcd}
HM_{99} \arrow[r] & \boxed{HM_{75}} \arrow[r] & HM_{76} \arrow[r] & HM_{86} \\
HM_{99} \arrow[r] & HM_{75} \arrow[r] & \boxed{HM_{76}} \arrow[r] & HM_{86} \\
HM_{75} \arrow[r] & HM_{76} \arrow[r] & \boxed{HM_{86}} \arrow[r] & HM_{99} \\
HM_{86} \arrow[r] & \boxed{HM_{99}} \arrow[r] & HM_{75} \arrow[r] & HM_{76}
\end{tikzcd}$$

$\boxed{\phantom{HM_{00}}}$ = Baseline HM

Figure 5.3: Orderings of HMs based on average normalized performance with different HMs as the baseline HM (based on the data in Table 5.2).

Using a single fixed baseline HM

A different approach for evaluating multiple HMs is to normalize each HM with respect to a single fixed baseline HM. All HMs can then be ordered based on their average normalized performance. Unfortunately, the ordering of these HMs can be dependent on the choice of the baseline HM, *i.e.*, different baseline HMs can lead to different orderings. These anomalies are *due to the choice of the baseline HM in ordering more than two HMs.* They can be illustrated in the following example.

Example 5.2 Based on the data in Example 5.1, we plot in Figure 5.3 the orderings of the four HMs when a different HM is used as the baseline. From this figure, we observe that the ordering with HM_{75} as the baseline HM is different from the ordering with HM_{86} as the baseline HM. In fact, there are three different orderings, depending on the choice of the baseline HM. It is obvious that by using this approach, it is difficult to determine which HM is the best among the four given HMs. ■

Due to the anomalies that may happen when different HMs are used as the baseline HM, it is difficult to determine the best HM over a subdomain. We may discover different "best" HMs using different HMs as the baseline.

5.3.3 Proposed strategy for general performance evaluation

In the previous subsection, we have presented potential inconsistencies in obtaining a total ordering of a given set of HMs within a given subdomain. There are no existing methods that can guarantee a correct ordering of these HMs based on incomplete performance information. This inconsistency means that the primary generalization objective, *i.e.*, selecting the "best" HM from a given set of HMs, is not always possible. In this case, the secondary objective of obtaining an HM that is "better" than the best existing HM becomes the main goal of the generalization process.

To accomplish this secondary objective, it is reasonable to use the best existing HM as the baseline. This approach should be used consistently throughout all learning, verification, and generalization phases of the design process. There are three advantages to using the approach based on a single baseline HM.

First, this approach is equivalent to obtaining pair-wise comparisons between the best existing HM (the baseline HM) and all other HMs in the given set. This equivalence means that the ordering between any single HM and the best existing HM can be obtained by comparing the HM's sample-mean value to the baseline level, BL.

Similarly, the uncertainty in the ordering between any one of these HMs and the incumbent HM can be determined based on P_{win} defined in Eqs. (5.1) or (5.2). The P_{win} value of an HM determines the degree of certainty that the given HM is better than the baseline HM. The closer P_{win} is to 1, the higher is the degree of certainty. In this case, P_{win} determines the likelihood that an HM is better than the current incumbent HM. Only HMs that satisfy all constraints ($P_{ok} > 0.5$) for all subdomains should be considered. The degree of certainty in determining whether an HM is better than the current incumbent HM can be controlled by requiring P_{win} to be greater than $0.5 + \Delta$ for some $\Delta > 0$.

Example 5.3 We continue to use the data in Example 5.1 as a running example. In this example, we assume that HM_{76} is the best existing HM to be improved upon, *i.e.*, the incumbent. Hence, we consider the performance of all HMs with HM_{76} as the baseline HM.

From Table 5.2 and Figure 5.3, we observe that both HM_{75} and HM_{99} have better average normalized performance values than HM_{76} (and so have $P_{win} > 0.5$).

If $\Delta = 0$, then both HMs would be acceptable based on their probability-of-win values. If our comparison is based only on each HM's sample-mean value, then HM_{99} will be considered a better HM. In contrast, if the selection between the two HMs is based on P_{win} (*i.e.*, the degree of certainty of being better than the baseline HM), then HM_{75} with the higher P_{win} value is a better choice.

If $0.4 > \Delta > 0.05$, then only HM_{75} ($P_{win} = 0.981$) can be considered better than HM_{76} because the P_{win} of HM_{99} ($P_{win} \equiv 0.519$) is too low. In short, for the objective of finding a better HM than HM_{76}, HM_{75} is the most robust choice among the given set of HMs. ■

The second advantage of using a fixed baseline is that all normalized performance values of all HMs are computed with respect to the same baseline HM in all subdomains. This approach allows performance constraints to be applied consistently over all subdomains.

The third advantage in this approach is that the average normalized performance value of an HM is an indication of the likelihood that the corresponding HM can satisfy the objective of being better than the baseline HM. The total ordering of a given set of HMs based on their average normalized performance can then be used during learning when performance information of each HM is usually incomplete and when a certain degree of inaccuracy is acceptable. Although this ordering is probably not totally "correct," it can be used during learning to distinguish between desirable and undesirable HMs. These average normalized performance values can also be used as fitness values in the selection/reproduction process during each learning experiment.

Finally, we consider some special cases of performance evaluation and generalization due to the unavailability of an incumbent HM. First, when *the best existing HM does not satisfy a given set of constraints*, the incumbent HM should still be used as the baseline HM in computing normalized performance. The constraints applied in each subdomain during the design process would then be consistent, relative to the performance of the incumbent HM in each subdomain. This case is discussed further in Section 5.4.1.

Second, when *there is no incumbent HM upon which to be improved*, the generalization process is less clearly defined since we do not have a secondary objective. One possible approach here is to choose one HM (either randomly or based on the user's choice) to fill the role of the incumbent HM upon which to be improved.

Another possible approach is to use the *median* performance value for each test case as the baseline performance for that test case. This approach has the advantage of not relying on treating one HM in a special fashion. We discuss this approach in more detail in Section 5.4.2.

5.4 Generalization Across Multiple Subdomains

In this section, we study the problem of statistical generalization over a problem domain that has multiple subdomains. Because different subdomains have different statistical behavior, performance from different subdomains cannot be combined or compared. Performance from each subdomain must then be treated independently and separately from that of any other subdomains.

The ideal case for generalization is when a single HM is considered the best in every subdomain under consideration (see the previous section). In this case, this HM can be selected as the new HM to be applied in the problem solver. Unfortunately, this case is unlikely in most cases.

In this section, we assume that the set of subdomains used in the design process is *representative of other subdomains.* In other words, other subdomains are assumed to behave in a similar fashion to one or more of the subdomains used in learning and in generalization.

Since there is no way to determine the relative importance of performance behavior in each selected subdomain, all selected subdomains are treated equally. One general approach is to ensure a minimum level of performance of the selected HM by *optimizing the worst-case performance* among the selected set of subdomains. Performance of the selected HM in new subdomains should likely be no worse than the worst performance among the selected set of subdomains.

We have divided our generalization strategy into two parts to deal with different conditions. First, we present our generalization strategy when there is an incumbent HM upon which to be improved. Second, we present a generalization strategy for the case when there is *no incumbent HM.*

5.4.1 Generalization with the best existing HM as the baseline

In this subsection, we consider the generalization process when there is an existing HM upon which to be improved. In this case, the objective of our generalization process is to find an HM that is better than the incumbent HM over the entire problem domain. When there are multiple such HMs, any of these HMs can be selected. However, the selection procedure should attempt to maximize the likelihood for selecting the best HM among the given set.

Based on the strategy in Section 5.3.3, the performance of an HM in a subdomain must be normalized with respect to the incumbent HM as the baseline. An HM is considered better than the incumbent HM in one subdomain when its P_{win} is greater than $0.5 + \Delta$ for that subdomain, while satisfying all other performance constraints ($P_{ok} > 0.5$). The value Δ controls the degree of certainty of this ordering. An HM can then be considered better than the incumbent HM for the entire problem domain when it is better than the incumbent HM in all selected subdomains.

There are several different scenarios that must be dealt with depending on the number of HMs that are better than the incumbent HM in all selected subdomains (none, one, or more than one). In addition, there is a special case when the incumbent HM is considered an unacceptable choice due to the set of constraints imposed. This situation happens when there are multiple objective measures, and it is desirable to have a performance trade-off different from the one provided by the incumbent HM.

We enumerate our generalization strategy for each possible scenario.

CASE 1: *One HM is better than the incumbent HM in all subdomains.* This case is the most ideal case. The one HM that is better than the incumbent HM in all selected subdomains can replace the incumbent HM in the target problem solver.

CASE 2: *Multiple HMs are better than the incumbent HM in all subdomains.* In this case, there are multiple acceptable HMs. The selection among these acceptable HMs should be based on maximizing the likelihood of the HM actually being better than the incumbent for the entire problem domain. This procedure corresponds to increasing the degree of confidence on the final HM selected, which can be achieved by enforcing a higher degree of certainty, *i.e.*, higher P_{win} within each subdomain, by increasing Δ. The Δ can be gradually increased until only one HM is considered superior to the incumbent in all subdomains. This generalization strategy is equivalent to optimizing the worst-case P_{win} over the selected set of subdomains.

An alternative approach is to perform selection based on maximizing the likelihood of the HM providing the best performance over the problem domain. This selection procedure can be based on optimizing the worst-case normalized average performance ($\hat{\mu}$) over the selected set of subdomains. In this alternative approach, the set of acceptable HMs under consideration should have P_{win} that satisfies a fairly high level of Δ in order to provide robustness with respect to achieving the generalization objective.

CASE 3: *No HM is better than the incumbent HM in all subdomains.* In this case, there is no HM that can be considered clearly superior to the incumbent HM. The target problem solver should continue to use the incumbent HM in its problem-solving process.

However, due to the uncertainty in performance evaluation, some of the HMs in the given set may actually perform better than the incumbent HM. It is then desirable to find this HM and that the P_{win} of this HM should be as high as possible in all selected subdomains.

When $\Delta > 0$ and no better HM is found, either Δ should be decreased until it reaches zero, or we should find an HM that is better than the incumbent HM in all selected subdomains. In this latter case, the selected HM is better than the incumbent HM at a lower degree of confidence than what is desired.

When $\Delta = 0$ and there is still no better HM, then Δ can be reduced below zero until such an HM can be found, which is also equivalent to finding an HM with the highest worst-case P_{win} among the given set of HMs. When $\Delta < 0$, the selected HM is likely to be inferior to the incumbent HM.

CASE 4: *The incumbent HM violates some constraints.* In this case, the incumbent HM cannot satisfy the desired performance trade-offs, which implies that any HM that satisfies the given constraints would perform better than the incumbent HM on some but not necessarily all performance measure(s).

In general, most HMs that satisfy all performance constraints would likely perform worse than the incumbent on the unconstrained optimization objective. Among HMs that satisfy the constraints, we must select the HM that provides the highest level of performance for the unconstrained optimization objective.

When different HMs in the set of acceptable HMs (satisfying all constraints in all subdomains) have the highest average performance for the unconstrained measure in different subdomains, we cannot easily select the best HM for the entire problem domain. We can use the lowest average performance (for the unconstrained measure) of each HM over the selected set of subdomains as the criteria for ordering the acceptable HMs. We can treat the average performance from different subdomains in an identical fashion since all performance values in all subdomains are normalized with respect to the same incumbent HM. The HM with the highest worst-case performance is then selected as the best among the available HMs.

We do not use P_{win} as the criterion in this case since it is not necessary for the selected HM to be better than the incumbent HM.

5.4.2 Generalization without an incumbent HM

In this subsection, we consider the generalization procedure when there is no incumbent HM. As previously stated in Section 5.3.3, we can pick an HM to act as the incumbent and apply the generalization process discussed in the previous subsection. However, the results of generalization in this case may not be meaningful because the selection of the incumbent is somewhat heuristic.

This situation is somewhat similar to Case 4 in the previous subsection in the sense that there is no acceptable incumbent HM. Unfortunately, in the current case with no existing incumbent HM, it is difficult to choose a proper HM to act as the baseline for normalization in all subdomains. We have already shown in Section 5.3.2 that vastly different results can be reached when different HMs are used as the baseline HM.

To avoid potential anomalies, we propose a different method for computing normalization performance values for a fixed set of HMs. Instead of using a single fixed baseline HM to compute normalized performance, we can use the median performance value for each test case as the baseline performance for that test case. The usage of the median performance value avoids the pitfall of choosing one HM as the baseline and still provides normalized performance values that can be treated similarly in all subdomains.

After obtaining the normalized performance based on this new approach, we can select the best HM in a fashion similar to that in Case 4 in the previous subsection. We can use the lowest average normalized performance metric in the unconstrained objective of each HM over the selected set of subdomains as the criteria for ordering HMs. The HM with the highest worst-case performance is then selected as the best HM for the entire problem domain.

A potential problem for this approach is the unavailability of the true median performance value for each test case. We have to estimate this true median of each test case with the sample median. During learning, the estimated median values are inaccurate due to insufficient performance data and are sensitive to the set of HMs under consideration. Since constraints are applied relative to the median values with this approach, inaccurate median values can cause inconsistencies for these constraints in different subdomains.

5.5 Summary

In this chapter, we study the statistical generalization process for *estimating the performance of an HM over the entire problem domain based on its performance on a subset of test cases evaluated in the design process.* This generalization process is essential because it is necessary to assert the performance of the selected HM on the entire problem domain, but only an incomplete subset of test cases can be tested during the design process.

The objective of statistical generalization is to determine the "best" HM from among a given set of HMs based on each HM's performance over the entire problem domain. When there is an incumbent HM already in use in the target problem solver, a secondary objective is to find an HM(s) that outperforms the incumbent HM over the entire problem domain. Due to the difficulty in correctly determining the true "best" HM, the generalization objective in this thesis is *based on finding a subset of HMs that is better than the incumbent HM and on selecting the best HM among this subset.*

When there are multiple subdomains in the target problem domain, *i.e.*, there are regions with different performance behavior, the generalization process can be divided into two parts. First, the true performance in each subdomain is determined based on statistical estimation. Second, the true performance across all subdomains is determined based on the performance of each HM in the selected set of subdomains. We assume that the selected set of subdomains is *representative* of all possible subdomains in the target problem domain.

(1) *Generalization Within a Subdomain.* When there is an existing incumbent HM, all performance values of all HMs are first normalized with respect to that of the incumbent HM as the baseline HM. The sample-mean value of each HM and each objective measure can then be used as statistical estimates of the true average performance over a subdomain. This statistical estimation is only possible when all normalized performance values within a subdomain are IID.

Because the incumbent HM is used as the baseline, an HM can be considered better than the incumbent HM in a subdomain when all constraints are satisfied ($P_{ok} > 0.5$) and the average normalized performance is judged to be above the baseline level ($\hat{\mu} > BL$) with a certain degree of confidence ($P_{win} > 0.5 + \Delta$).

(2) *Generalization Across Subdomains.* Since performance from different subdomains has different behavior and the relative importance of each selected subdomain is unknown, each subdomain must be treated equally. The performance of each HM over a selected set of subdomains can then be represented by its *worst-case performance* among these subdomains. By optimizing the worst-case performance, the true performance over the entire problem domain can be bounded above this minimum level.

There are three main cases in dealing with generalization across subdomains.

(a) There is an acceptable incumbent HM. In this case, the goal is to find the HM most likely to provide better performance than the incumbent HM. The objective is then to maximize the worst-case P_{win} over the selected subdomains.

When minimum $P_{win} < 0.5 + \Delta_{min}$, the selected HM may or may not actually be superior to the incumbent HM. More detailed performance evaluation is necessary before this new HM can replace the incumbent HM in the problem solver.

When minimum $P_{win} \geq 0.5 + \Delta_{min}$, then there is a high degree of confidence that the selected HM is actually better than the incumbent HM over the problem domain. In this case, the new HM can replace the incumbent HM in the problem solver. However, some performance validation should still be performed (see Section 2.6).

(b) The incumbent HM exists but violates some constraints. In this case, it is not necessary for the selected HM to be better than the incumbent HM for the unconstrained optimization objective. It is still desirable for the selected HM to perform better relative to the incumbent HM. The objective is then to maximize the worst-case average normalized performance ($\hat{\mu}$) on the unconstrained optimization objective (J_0) over the selected subdomains. Note that only HMs that satisfy all constraints ($P_{ok} > 0.5$) in all subdomains should be considered.

(c) There is no incumbent HM. In this case, it is difficult to provide consistent normalization across all subdomains so that the worst-case performance over the selected subdomains can be compared meaningfully.

There are two possible alternatives here. First, an HM can be designated as the fixed baseline HM to be used throughout all subdomains. The difficulty with this alternative is in selecting the appropriate HM so that the generalization result is meaningful. Second, we can use the *median performance value* of each test case as the baseline for normalization of that test case. There is a potential problem related to the application of constraints with this normalization method during learning. In addition, this normalization method is more difficult to apply during learning when we do not have sufficient tests on each HM to estimate the medians accurately.

The objective in this case is the same as the case in case (b), *i.e.*, optimizing the worst-case average normalized performance for J_0 when all constraints are satisfied.

6. RESULTS ON REAL-WORLD APPLICATIONS

In this chapter, we present experimental results based on applications of our TEACHER heuristics-design system to develop new HMs for several real-world problem solvers. The target problem solvers include (a) process mapping using PGA, (b) branch-and-bound search on several combinatorial optimization problems, (c) CRIS and GATEST for generating test-patterns for VLSI circuits, (d) TimberWolf for the placement and routing of VLSI circuits, and (e) blind equalization using a gradient-descent approach.

The parameters used during learning for each application are summarized in Table 6.1.

6.1 Process Mapping on Distributed-Memory Multicomputers

The first target application is the process mapping problem. Process mapping involves placing a set of communicating processes on a multicomputer system so that the completion time of the processes is minimized. The problem is characterized by nondeterministic (data-dependent) execution times between inter-process communications and by the amount of data communicated between processes. It can be solved as a deterministic optimization problem using average execution times and data volumes; however, the solution is inaccurate when execution and communication activities change with time. Further, a deterministic model does not account for delays incurred due to blocked messages. Such unpredictable delays can only be found when the processes are actually executed or simulated.

Table 6.1: Genetic-algorithm parameters used in our learning system. (*Num. Active HMs* is the number of active HMs in each generation, *New HMs* is the number of new HMs generated at the beginning of each generation, and *HMs Ver.* is the number of HMs selected for verification at the end of the last generation.)

Application	Num. of Gen.	Dur. of a Gen.	Num. Active HMs	New HMs	Cross. Rate	Mut. Rate	Rand. Gen. Rate	HMs Ver.
PGA	10	80	30	20	0.40	0.40	0.20	20
B&B	10	160	40	30	0.50	0.17	0.33	20
CRIS	10	100	30	20	0.45	0.35	0.20	20
GATEST	10	160	40	30	0.50	0.17	0.33	20
TimberWolf	10	100	30	20	0.45	0.35	0.20	20
Blind Equalizer	10	160	30	20	0.35	0.20	0.25	20

In this section, we use post-game analysis (PGA) [18,39,40] as the target problem solver. This problem solver is described in Section 6.1.1. We then present results obtained by applying TEACHER to improve the HMs in PGA.

6.1.1 Post-game analysis heuristics

Yan and Lundstrom proposed post-game analysis (PGA) [18,39,40], a simulation-based method for finding good mappings. Their system collects an execution trace consisting of actual execution times in between communications and amounts of data sent between processes and uses them in a simulation system to find the actual completion time of a specific mapping. It then applies heuristics to propose a new mapping, evaluating the effectiveness of the new mapping through the simulation system. This iterative refinement is repeated until no further improvement is possible. PGA can be applied in practice by repeatedly collecting a trace for a short period of time, optimizing the mapping by PGA on a different computer while the original application program is running, and proposing a new mapping for the application program to use.

There are four components of the heuristics used in PGA:

(a) *proposal-generation heuristics* for generating proposals to perturb a mapping based on independent optimization subgoals,

(b) *priority-assessment heuristics* for prioritizing each site and process,

(c) *transformation-generation heuristics* for generating possible transformations from the ordered list of sites and processes, and

(d) *feasibility heuristics* for checking the feasibility of a move.

These heuristics are represented as expressions (or heuristic decision elements – HDEs) that combine values collected during program execution and are applied to make decisions.

These four heuristics interact extensively in proposing alternative mappings. Consequently, we cannot isolate each set of heuristics and learn them independently. Instead, we consider the four components that make up a *PGA HM*, and learning aims to find the best collection of HDEs and the proper value for each threshold. PGA HMs used in learning are generated randomly, as well as by crossovers and mutations.

PGA HMs are evaluated by two performance measures: the quality of a mapping (process completion time of the mapping) and the cost of finding the mapping (PGA execution time). Learning aims to find PGA HMs with the minimum completion time and cost within a user-specified limit. This objective is necessary since PGA has to be run concurrently with the application program, and a new mapping should be proposed within a time constraint.

6.1.2 Experimental results

In designing PGA HMs, we chose an application based on a divide-and-conquer algorithm: each process computes for a random amount of time, sends a message to each of its child processes to start their computation, and waits for the results from its descendents before reporting to its parent. We used three machine architectures (3-by-3, 4-by-4, and 5-by-5 meshes) and two process sizes (200 and 300 processes), resulting in six subdomains. All subdomains were assumed to belong to one problem domain. Each *PGA test case* specifies the number of processes in the divide-and-conquer tree, the execution time of each node of the tree, one of the machine architectures, and an initial mapping of the processes on the architecture. All experiments were performed on a Sun SparcStation 10/30. For the applications we have used, testing a PGA HM on one test case is time consuming: a small learning experiment with 6,400 evaluations (6 subdomains, each with 800 tests for learning and 400 tests for final verification) took between 7 to 11 days of CPU time. Our experiments address the generalization of the HMs learned. (See Section 4.6 for scheduling results.) We used three subdomains (a 3-by-3 mesh with 100 and 200 processes and a 5-by-5 mesh with 200 processes) for learning and generalized the HMs learned to the remaining three subdomains. In learning, 800 tests were performed for each subdomain, and the best five HMs that satisfied the cost constraint were selected for full verification. We considered two cost constraints: 30% and 100% of the cost of the original PGA HM by Yan and Lundstrom [18, 40].

Based on our experimental results, we have found that under a 0.3 cost constraint, there are two subspaces (subspace 1 contains three subdomains with 100 processes, and subspace 2 contains the remaining three subdomains with 200 processes). In this case, the PGA HMs learned do not generalize well and are biased towards the number of processes in the application program. Table 6.2 shows the costs and qualities of the generalized HMs for the case with one subspace and for the case with two subspaces. These HMs are obtained using the strategy described in Case 4 in Section 5.4.1. Note that the generalized HM for the one-subspace case is the same as the generalized HM for the 100-process subspace in the two-subspace case. In this case, there are other HMs that perform better for the 200-process subspace, but violate constraints for the 100-process subspace.

Table 6.3 shows the costs and qualities of the generalized HMs as compared to those of the learned HMs. We see that both have similar costs and qualities. In contrast, the cost of learning is around 500 times higher than that of the generalized HMs.

Next, we have found that under the 1.0 cost constraint, all subdomains should belong to one subspace and can be evaluated by one HM. In the case with two subspaces, the same HM is the best HM for both subspaces. Table 6.4 shows the costs and qualities of the generalized HMs. This HM is obtained by optimizing the worst-case P_{win} (see Section 5.4.1) since the incumbent HM is acceptable in this case. Although this HM actually has worse qualities in two subdomains, it also has significantly smaller costs than the incumbent HM. In addition, it outperforms the incumbent HM in four subdomains at significantly smaller costs.

Table 6.2: Quality-cost comparison of generalized HMs using a cost constraint of 30% of the original PGA cost with one and two subspaces. Subdomains with "*" are learned; subdomains with "+" are generalized from subdomains with the same number of processes. (Subdomains 1-3 belong to one subspace, and subdomains 4-6 belong to another in the two-subspace case.)

Subdomain			Generalized HM(s)			
			Two Subspaces		One Subspace	
ID	Architecture	Processes	Quality	Cost	Quality	Cost
1 *	3-by-3 mesh	100	0.934	0.251	0.934	0.251
2 +	4-by-4 mesh	100	0.933	0.231	0.933	0.231
3 +	5-by-5 mesh	100	0.954	0.235	0.954	0.235
4 *	3-by-3 mesh	200	0.993	0.283	0.978	0.195
5 +	4-by-4 mesh	200	0.986	0.244	0.943	0.088
6 *	5-by-5 mesh	200	0.964	0.274	0.913	0.144

Table 6.3: Quality-cost comparison of learned and generalized HMs using a cost constraint of 30% of the original PGA cost. Subdomains with "*" are learned; subdomains with "+" are generalized from subdomains with the same number of processes. (Subdomains 1-3 belong to one subspace, and subdomains 4-6 belong to another.) The cost of learning is the sum of normalized execution times with respect to the baseline HM.

Subdomain			Generalized HM(s)		Learned HM(s)		Normalized
ID	Architecture	Processes	Quality	Cost	Quality	Cost	Learning Cost
1 *	3-by-3 mesh	100	0.934	0.251	0.934	0.251	583.3
2 +	4-by-4 mesh	100	0.933	0.231	0.948	0.277	349.9
3 +	5-by-5 mesh	100	0.954	0.235	0.951	0.230	428.8
4 *	3-by-3 mesh	200	0.993	0.283	0.993	0.283	505.8
5 +	4-by-4 mesh	200	0.986	0.244	0.993	0.274	348.0
6 *	5-by-5 mesh	200	0.964	0.274	0.961	0.269	416.9

Table 6.4: Quality-cost comparison of generalized HMs using a cost constraint of 100% of the original PGA cost. Subdomains with "*" are learned; subdomains with "+" are generalized from subdomains with the same number of processes.

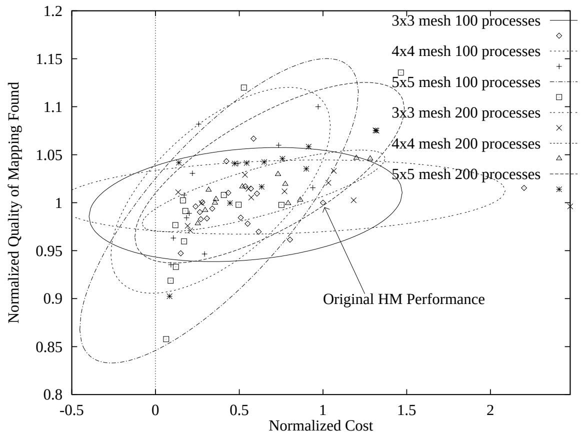
Subdomain			Generalized HM		
ID	Architecture	Processes	Quality	Cost	P_{win}
1 *	3-by-3 mesh	100	0.998	0.538	0.388
2 +	4-by-4 mesh	100	1.013	0.389	0.795
3 +	5-by-5 mesh	100	0.992	0.380	0.356
4 *	3-by-3 mesh	200	1.006	0.750	0.850
5 +	4-by-4 mesh	200	1.012	0.645	0.968
6 *	5-by-5 mesh	200	1.031	0.681	0.981

Next, we show the performance of learned PGA HMs when generalized across the three multicomputer architectures under various combinations of cost constraints and number of processes in the application program. As discussed in Sections 2.3 and 2.6.2, performance values related to each objective in a multi-objective application need to be considered independently in order to avoid inconsistencies in evaluation.

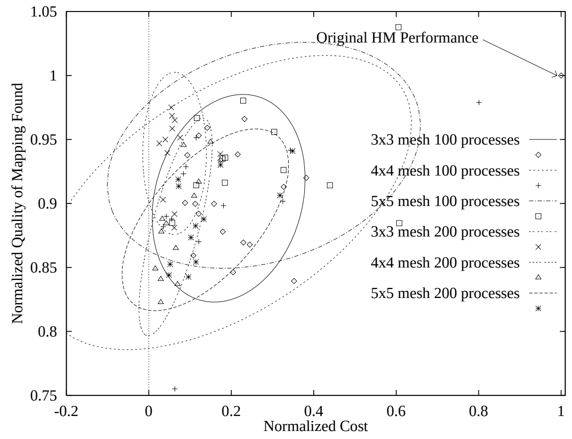
To this end, we plot in a two-dimensional graph the distribution of the normalized quality of an HM on a test case and the corresponding normalized cost of the same HM and test case. Using a method we have presented earlier in Section 2.6.2 to show cost-quality tradeoffs, we identified a 90% constant probability contour for each HM after removing outliers, checking for bivariate normality, and finding the 90% constant probability-density contour of the bivariate distribution.

Figure 6.1 shows the cost-quality of generalized PGA HMs on various architectural configurations and the number of nodes in the divide-and-conquer tree. Each of these four graphs represents the performance of one HM obtained by learning in one subdomain and generalizing to two other subdomains.

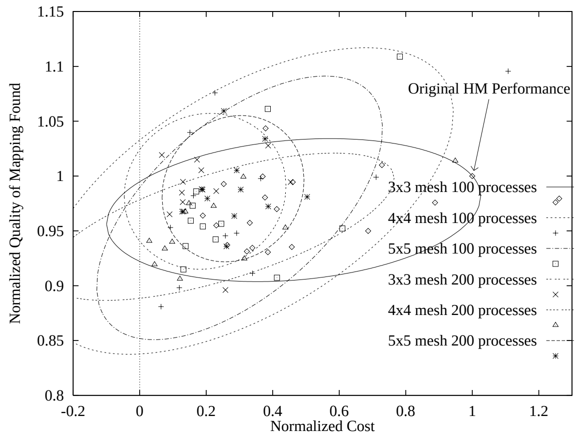
The HM used in Figure 6.1(a) was obtained by generalizing HMs learned under a 1.0 cost constraint on three subdomains (see Table 6.4) to the other three subdomains. Likewise, the HM used in Figure 6.1(b) (resp., 6.1(c)) are HM_1 (resp., HM_4) in Table 6.3. In learning



(a) all subdomains and 1.0 cost constraint



(b) 100-process subdomain and 0.3 cost constraint



(c) 200-process subdomain and 0.3 cost constraint

Figure 6.1: Performance of PGA HMs learned for a 3-by-3 mesh architecture.

the HMs, the fixed-duration minimum-risk strategy (resp., DMDS minimum-risk strategy) was used in Figure 6.1(a) (resp., Figure 6.1(b) and 6.1(c)). We see in Figure 6.1(a) that all of the contour plots are clustered together, implying that the PGA HMs selected under the 1.0 cost constraint generalize well to other subdomains. Further, these HMs have lower costs than the original HM (normalized to one) while having qualities close to or better than the original HM. In Figure 6.1(b) and 6.1(c), we find both generalized HMs to have similar quality levels, but the HM generalized from the 100-process subdomains have higher costs than those from the 200-process subdomains. In this case, HMs that perform well for the 200-process subdomains would violate the cost constraint for the 100-process subdomains. This difference in cost happens because the achievable cost for each subdomain in the 0.3 cost-constraint case is lower; for larger test cases, there is more room for improvement in terms of cost, and a lower relative cost can be achieved.

6.2 Branch-and-Bound Search

In this section, we study the application of TEACHER to improve heuristics in a branch-and-bound-search problem solver. This problem-solving approach has been applied to solve many combinatorial optimization applications, with different sets of heuristics for different applications. In this section, we develop the decomposition heuristics in a branch-and-bound search for three different target applications.

6.2.1 Application descriptions

A branch-and-bound (B&B) search algorithm is a systematic method for traversing a search tree or search graph in order to find a solution that optimizes a given objective while satisfying given constraints [50]. It decomposes a problem into smaller subproblems and repeatedly decomposes them until a solution is found or infeasibility is proved. Each subproblem is represented by a node in the search tree/graph. The algorithm has four sets of HMs:

Table 6.5: Generation of 12 subdomains of test cases for testing decomposition HMs in a B&B search.

Appl.	Subdomain Attributes
VC	• Connectivity of vertices is (0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, or 0.6) • Number of vertices is between 16 and 45
TSP	• Distributions of 8-18 cities (uniformly distributed between 0-100 on both the X and Y axes, uniformly distributed on one axis and normally distributed on another, or normally distributed on both axes) • Graph connectivity of cities is (0.1, 0.2, 0.3, or 1.0)
KS	• Range of both profits and weights is (100-1000), (100-200), and (100-105) • Variance of profit/weight ratio is (1.05, 1.5, 10, and 100) • 13-60 objects in the knapsack

(a) the *selection HM* for selecting a search node for expansion based on a sequence of selection keys for ordering search nodes,

(b) the *decomposition HM* (or branching mechanism) for expanding a search node into descendents using operators to expand (or transform) a search node into child nodes,

(c) the *pruning HM* for pruning inferior nodes in order to trim potentially poor subtrees, and

(d) the *termination HM* for determining when to stop.

In this thesis, we apply learning to find only new *decomposition HMs*; preliminary results on learning of selection and pruning HMs can be found in [25]. We consider optimization search, which involves finding the optimal solution and proving its optimality.

We illustrate our method on three applications: traveling salesman problems on incompletely connected graphs mapped on a two-dimension plane (TSP), vertex-cover problems (VC), and knapsack problems (KS). Table 6.5 shows the parameters used to generate a *test case* in each application. All subdomains are assumed to belong to one problem subspace.

Table 6.6: Original and generalized decomposition HMs used in a B&B search. (The new HM learned for VC can be interpreted as follows: l is the primary key, and $n - \Delta l$ is the secondary key.)

Application and Variables Used in Constructing HMs	Orig. HM	Gen. HM
Vertex-Cover Problem l = live degree of vertex (uncovered edges) d = dead degree of vertex (covered edges) n = average live degree of all neighbors Δl = difference between l from parent node to current node	l	$1000l$ $+n$ $-\Delta l$
Traveling Salesman Problem c = length of current partial tour m = min. length to complete current tour a = avg. length to complete current tour l = number of neighboring cities not yet visited d = number of neighbor already visited	c	$m * c$
Knapsack Problem p, w = profit/weight of object s = weight slack = weight limit − current weight p_{max}, p_{min} = max./min. profit of unselected object w_{max}, w_{min} = max./min. weight of unselected object	p/w	p/w

The problem solver here is a B&B algorithm, and a test case is considered solved when its optimal solution is found. Note that the decomposition HM is a component of the B&B algorithm. We use well-known decomposition HMs developed for these applications as our baseline HMs (see Table 6.6). The normalized cost of a candidate decomposition HM is defined in terms of its *average symmetric speedup* (see Eq. (2.8)), which is related to the number of nodes expanded by a B&B search using the baseline HM and by a B&B search using the new HM. Note that we do not need to measure quality, as both the new and existing HMs when applied in a B&B search look for the optimal solution.

6.2.2 Experimental results

First, we develop a generalized HM for each target application using TEACHER. In each application, we selected six subdomains and performed learning on each using 1,600 tests. We then selected the top five HMs from each learned subdomain, fully verified them on all of the learned subdomains, and selected one final HM to be used across all subdomains. Table 6.7 summarizes the learning, generalization, and validation results. For learning, we show the average symmetric speedup of the top HM learned in each subdomain and the normalized cost of learning, where the latter was computed as the ratio of the total CPU time for learning to the harmonic mean of the CPU time required by the baseline HM on test cases used in learning. The results show that a new HM learned for a subdomain has approximately 1-35% improvement in its average symmetric speedups and 3,000-16,000 times increment in learning costs.

Table 6.7 also shows the average symmetric speedups of the generalized HMs. We picked six subdomains randomly for learning. After learning and full verification of the top five HMs in each subdomain, we identify the generalized HM across all twelve subdomains. Our results show that the generalized HMs have between 0-8% improvement in average symmetric speedups (using 30 test cases in each subdomain). Note that these results are worse than those obtained by learning and that the baseline HM is the best HM in solving the knapsack problem.

The bottom part of Table 6.7 shows the average symmetric speedups when we validate the generalized HMs on larger test cases. These test cases generally require 10-50 times more nodes expanded than those used earlier. Surprisingly, our results show better improvement (9-23% using 15 test cases in each subdomain). Furthermore, six of the twelve subdomains with a high degree of connectivity in the vertex-cover problem have slowdowns. This result is a clear indication that these subdomains should be grouped in a different subspace and learned separately.

Table 6.6 shows the new decomposition HMs learned for the three applications. We list the variables that were supplied to the learning system. In addition to these variables, we have also included constants that can be used by the heuristics generator. An example of

Table 6.7: Results of learning and generalization for VC, TSP, and KS. (In the results on generalization, numbers with "*" are the ones learned; only one common HM is generalized to all 12 subdomains.)

Type	Application	Perf. Measure	Subdomain												Average
			1	2	3	4	5	6	7	8	9	10	11	12	
Learning	VC	Sym-SU	0.000	0.011	0.041	0.000	0.044	0.022	0.008	0.013	0.000	0.000	0.000	0.000	0.012
		Cost	26343.5	23570.9	21934.1	12951.6	11034.3	12414.4	5871.0	8093.3	6878.0	5051.2	3826.2	3107.3	11756.3
	TSP	Sym-SU	0.194	0.073	0.288	0.378	0.106	0.068	0.267	0.382	0.048	0.165	0.208	0.083	0.188
		Cost	2846.6	1543.9	2077.7	2207.7	2314.9	1865.6	1889.9	1847.5	2509.7	1947.0	1445.4	1958.8	2037.9
	KS	Sym-SU	0.000	0.000	0.000	0.000	0.000	0.000	0.893	0.000	0.263	0.107	2.840	0.089	0.349
		Cost	25707.7	32587.9	9671.6	26408.1	24903.6	22309.1	3648.1	7943.1	8114.7	6476.2	772.9	10684.4	14935.6
Generalization	VC	Sym-SU	0.218	0.283*	0.031	0.068*	0.054	0.060*	0.017	0.049*	0.016	−0.000*	−0.011	0.028*	0.068
	TSP	Sym-SU	0.072*	0.004	0.082*	0.225	0.005*	0.061*	0.139	0.155	−0.010	0.054	0.090*	0.083*	0.080
	KS	Sym-SU	0.000*	0.000*	0.000	0.000	0.000	0.000*	0.000*	0.000	0.000*	0.000	0.000	0.000*	0.000
Validation	VC	Sym-SU	0.070	0.638	0.241	0.078	0.073	0.020	−0.013	−0.004	−0.018	−0.000	−0.019	−0.010	0.088
	TSP	Sym-SU	0.417	0.036	0.144	0.155	0.131	0.364	1.161	0.101	0.108	0.008	0.022	0.131	0.231
	KS	Sym-SU	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

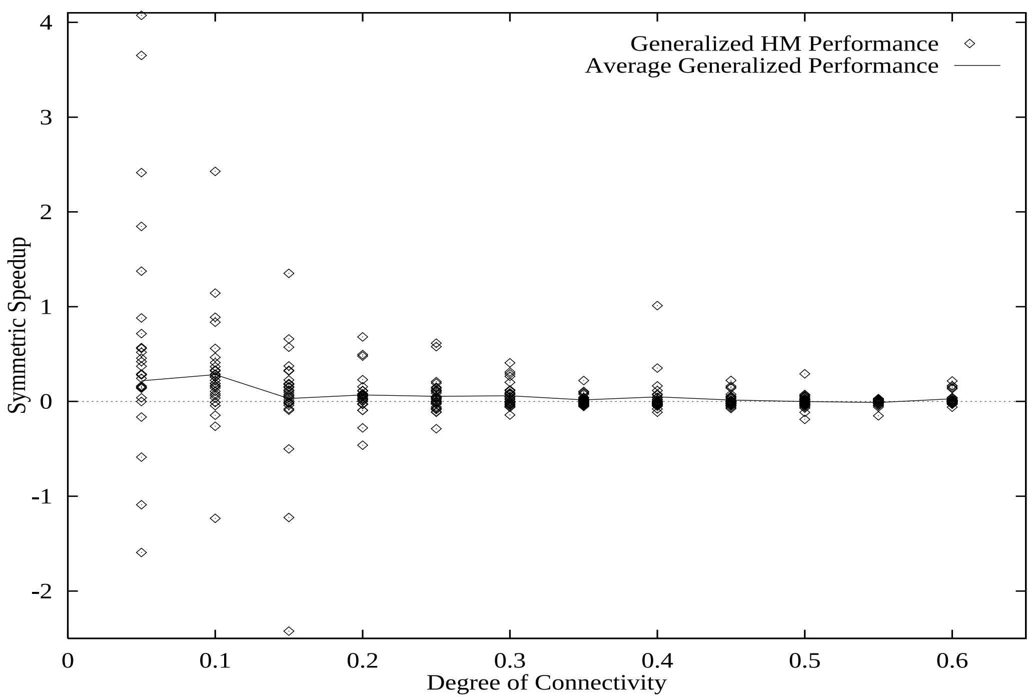


Figure 6.2: Distribution of performance values of the generalized decomposition HM to solve VC normalized on a case-by-case basis with respect to those of the original HM (performance values are for the 30 test cases used during the design process).

such a constant is shown in the HM learned for the vertex-cover problem. This formula can be interpreted as using l as the primary key for deciding which node to be included in the covered set. If the l's of two alternatives are different, then the remaining terms in the formula $(n - \Delta l)$ are insignificant. In contrast, when the l's are the same, then we use $n - \Delta l$ as a tie breaker.

Finally, Figure 6.2 plots the distribution of symmetric speedups of the generalized HM for VC with respect to the original HM using test cases in our generalization database. It shows performance improvements of each test case with respect to that of the original HM. It further shows that performance is fairly evenly distributed above and below the average value without unnatural compression of ranges. This observation confirms that symmetric speedup is a proper normalization measure in this case. This plot also shows the difference in performance distribution across different subdomains, indicating the need to use a range-independent measure such as the probability of win.

In short, our results show that reasonable improvements can be obtained by learning and by generalization. We anticipate further improvements by (a) learning and generalizing new pruning HMs in a depth-first search, (b) partitioning the problem space into a number of subspaces and learning a new HM for each, and (c) identifying attributes that help explain why one HM performs well in some subdomains.

6.3 Test Generation for VLSI Circuits

The next application is based on CRIS [15] and GATEST [35], two genetic-algorithm software packages for generating patterns to test sequential VLSI circuits. In our experiments, we used sequential circuits from the ISCAS89 benchmarks [116] plus several other larger circuits. Since these circuits are from different applications, it is difficult to classify them by some common features. Consequently, we treat each circuit as an individual subdomain. As we like to find one common HM for all circuits, we assume that all circuits are from one subspace.

6.3.1 CRIS

CRIS [15] is based on continuous mutations of a given input test sequence and on analyzing the mutated vectors for selecting a test set. Hierarchical simulation techniques are used in the system to reduce the memory requirements, thereby allowing test generations for large VLSI circuits. The package has been applied successfully to generate test patterns with high fault coverages for large combinatorial and sequential circuits.

CRIS in our experiments is treated as a problem solver in a black box, as we have minimal knowledge in its design. An HM targeted for improvement is a set of eight parameters used in CRIS (see Table 6.8). Note that parameter P_8 is a random seed, implying that CRIS can be run multiple times using different random seeds in order to obtain better fault coverages. (In our experiments, we used a fixed sequence of ten random seeds from Table 6.9.)

Table 6.8: Parameters in CRIS treated as an HM in learning and in generalization. (The type, range, and step of each parameter were recommended to us by the designer of CRIS. The default parameters were not given to us as they are circuit dependent.)

Parameter	Type	Range	Step	Definition	Learned Value
P_1	integer	1-10	1	related to the number of stages in a flip-flop	1
P_2	integer	1-40	1	related to the sensitivity of changes of state of a flip-flop (number of times a flip-flop changes its state in a test sequence)	12
P_3	integer	1-40	1	selection criterion — related to the survival rate of a candidate test sequence in the next generation	38
P_4	float	0.1-10.0	0.1	related to the number of test vectors concatenated to form a new test sequence	7.06
P_5	integer	50-800	10	related to the number of useless trials before quitting	623
P_6	integer	1-20	1	number of generations	1
P_7	float	0.1-1.0	0.1	how genes are spliced in the genetic algorithm	0.1
P_8	integer	any	1	seed for the random number generator	-

Table 6.9: Sequence of random seeds used in learning experiments for CRIS, GATEST, and TimberWolf.

Sequence of Random Seeds									
61801	98052	15213	48823	55414	60203	43212	08540	94702	92715

A major problem in using the original CRIS is that it is hard to find proper values for the seven parameters (excluding the random seed) for a particular circuit. The designer of CRIS manually tuned these parameters for each circuit, resulting in HMs that are hard to generalize. This tuning was done because the designer wanted to obtain the highest possible fault coverage for each circuit, and computation cost was only a secondary consideration. Note that the times for manual tuning were exceedingly high and were not reported in [15].

Our goal is to develop one common HM that can be applied across all the benchmark circuits and that has similar or better fault coverages as compared to those of the original CRIS. The advantage of having one HM is that it can be applied to new circuits without further manual tuning.

6.3.2 GATEST

GATEST [35] is another test-pattern generation package based on genetic algorithms. It augments existing techniques in order to reduce execution times and to improve fault coverages. The genetic-algorithm component evolves candidate test vectors and sequences, using a fault simulator to compute the fitness of each candidate test. To improve performance, the designers manually tuned various genetic-algorithm parameters in the package, including alphabet size, fitness function, generation gap, population size, and mutation rate as well as selection and crossover schemes. High fault coverages were obtained for most of the ISCAS89 sequential benchmark circuits [116], and execution times were significantly lower in most cases than those obtained by HITEC [117], a deterministic test-pattern generator.

The entire genetic-algorithm process was divided into four phases, each with its own fitness function that had been manually tuned by the designers. The designers also told us that Phase 2 has the largest impact on performance and recommended that we improve it first. As a result, we treat GATEST as our problem solver, and the fitness function (a symbolic formula) in Phase 2 as our HM. The original form of this fitness function is

$$fitness_2 = \#\_faults\_detected + \frac{\#\_faults\_propagated\_to\_flip\_flops}{(\#\_faults)(\#\_flip\_flops)} . \tag{6.1}$$

In learning a new fitness function, we have used the following variables as possible arguments of the function: *#_flip_flops, #_faults_detected, #_faults_propagated_to_flip_flops, #_faults, #_circuit_nodes,* and *sequence_length.* The operators allowed to compose new fitness functions include +, −, ∗, and /.

6.3.3 Experimental results

In our experiments, we chose five circuits as our learning subdomains. In each of these subdomains, we used TEACHER to test CRIS 1000 times with different HMs, each represented as the first seven parameters in Table 6.8. At the end of learning, we picked the top 20 HMs and evaluated them fully by initializing CRIS by ten different random seeds (P_8 in Table 6.8 with values from Table 6.9). We then selected the top five HMs from each subdomain, resulting in a total of 25 HMs supplied to the generalization phase. We evaluated the 25 HMs fully (each with ten random seeds) on the five subdomains used in learning and five new subdomains. We then selected one generalized HM to be used across all ten circuits. Since there is no incumbent HM, we use the generalization strategy from Section 5.4.2 in this case. The elements of the generalized HM found are shown in Table 6.8.

For GATEST, we applied learning to find good HMs for six circuits (s298, s386, s526, s820, s1196, and s1488 in the ISCAS89 benchmark). We then generalized the best 30 HMs (five from each subdomain) by first evaluating them fully (each with ten random seeds from Table 6.9) on the six subdomains and by selecting one generalized HM for all circuits. Since there is an incumbent HM, we use the usual generalization strategy from Section 5.4.1 in this case. The final fitness function we got after generalization is

$$fitness_2 \;=\; 2 \times \#\_faults\_propagated\_to\_flip\_flops \;-\; \#\_faults\_detected\ . \tag{6.2}$$

Table 6.10 shows the results after generalization for CRIS and GATEST. For each circuit, we present the average and maximum fault coverages (over ten random seeds) and the corresponding computational costs. These fault coverages are compared against the published fault coverages of CRIS [15] and GATEST [35] as well as those of HITEC [117]. Note that the maximum fault coverages reported in Table 6.10 were based on ten runs of the underlying problem solver, implying that the computational cost is ten times the average cost.

Table 6.10: Performance of HMs in terms of computational cost and fault coverage for CRIS and GATEST. (Learned subdomains for CRIS are marked by "*" and generalized subdomains by "+"). Performance of HITEC is from the literature [15,117]. Costs of our experiments are running times in seconds on a Sun SparcStation 10/51; costs of HITEC are running times in seconds on a Sun SparcStation SLC [35] (around 4-6 times slower than a Sun SparcStation 10/51).

Circuit ID	Total Faults	Fault Coverage				Cost		CRIS Generalized HM			GATEST Generalized HM		
		HITEC	CRIS	Avg. GATEST	Max. GATEST	HITEC	Avg. GATEST	Avg. FC	Max. FC	Avg. Cost	Avg. FC	Max. FC	Avg. Cost
*s298	308	86.0	82.1	85.9	86.0	15984.0	128.6	84.7	86.4	10.9	85.9	86.0	126.4
s344	342	95.9	93.7	96.2	96.2	4788.0	134.8	96.1	96.2	21.8	96.2	96.2	133.3
s349	350	95.7	–	95.7	95.7	3132.0	136.9	95.6	95.7	21.9	95.7	95.7	128.3
+s382	399	90.9	68.6	87.0	87.5	43200.0	203.3	72.4	87.0	7.2	87.0	87.5	208.9
s386	384	81.7	76.0	76.9	77.9	61.8	67.6	77.5	78.9	3.5	78.6	79.3	78.6
*s400	426	89.9	84.7	85.7	86.6	43560.0	229.3	71.2	85.7	8.4	85.7	86.6	215.1
s444	474	87.3	83.7	85.6	86.3	57960.0	259.4	79.8	85.4	9.3	85.6	86.3	233.8
*s526	555	65.7	77.1	75.1	76.4	168480.0	333.4	70.0	77.1	10.0	75.5	77.3	302.7
s641	467	86.5	85.2	86.5	86.5	1080.0	181.2	85.0	86.1	19.5	86.5	86.5	195.0
+s713	581	81.9	81.7	81.9	81.9	91.2	219.9	81.3	81.9	23.0	81.9	81.9	256.5
s820	850	95.6	53.1	60.8	68.0	5796.0	266.4	44.7	46.7	51.3	69.3	80.9	225.4
*s832	870	93.9	42.5	61.9	66.8	6336.0	265.8	44.1	45.6	44.6	66.9	72.8	251.0
s1196	1242	99.7	95.0	99.2	99.5	91.8	292.1	92.0	94.1	20.0	99.2	99.4	421.7
*s1238	1355	94.6	90.7	94.0	94.4	132.0	380.5	88.2	89.2	23.0	94.0	94.2	585.2
s1488	1486	97.0	91.2	93.7	96.0	12960.0	512.3	94.1	95.2	85.6	94.3	96.5	553.4
+s1494	1506	96.4	90.1	94.0	95.8	6876.0	510.4	93.2	94.1	85.5	93.6	95.6	584.3
s1423	1515	40.0	77.0	81.0	86.3	–	3673.9	82.0	88.3	210.4	81.3	87.3	4325.7
+s5378	4603	70.3	65.8	69.5	70.1	–	9973.3	65.3	69.9	501.8	69.6	71.9	8875.7
s35932	39094	89.3	88.2	89.5	89.7	13680.0	184316.0	77.9	78.4	4265.7	89.4	89.7	184417.0
am2910	2573	85.0	83.0	–	–	–	–	83.7	85.2	307.6	–	–	–
+div16	2147	72.0	75.0	–	–	–	–	79.1	81.0	149.9	–	–	–
tc100	1979	80.6	70.8	–	–	–	–	72.6	75.9	163.8	–	–	–

Table 6.11: Summary of results comparing the performance of our generalized HMs with respect to those of HITEC, CRIS, and GATEST. (The first number in each entry shows the number of wins out of all applicable circuits, the second, the number of ties, and the third, the number of losses. A second number in the entry on wins indicates the number of circuits in which the test efficiency is already 100%. For these circuits, no further improvement is possible.)

Our HM wins/ties/loses with respect to the following systems	CRIS Generalized HM			GATEST Generalized HM		
	Total	Max. FC	Avg. FC	Total	Max. FC	Avg. FC
HITEC	22	6, 2, 14	4, 0, 18	19	5+2, 2, 10	4+2, 1, 12
Original CRIS	21	16, 1, 4	11, 0, 10	18	18, 0, 0	17, 0, 1
Original GATEST	19	4, 3, 12	3, 0, 16	19	7+2, 7, 3	7+2, 8, 2
Both HITEC and CRIS	21	5, 2, 14	3, 0, 18	18	5+2, 1, 10	3+2, 1, 13
Both HITEC and GATEST	19	3, 3, 13	1, 0, 18	19	3+2, 4, 10	2+2, 2, 13
All HITEC, CRIS, and GATEST	18	2, 3, 13	1, 0, 17	18	3+2, 3, 10	1+2, 1, 14

Table 6.11 summarizes the improvements of our learned and generalized HMs as compared to the published results of CRIS, GATEST, and HITEC. Each entry of the table shows the number of times our HM wins, ties, and loses in terms of fault coverages with respect to the method(s) in the first column. Our results show that our generalized HM based on CRIS as the problem solver is better than the original CRIS in 16 out of 21 circuits in terms of the maximum fault coverage and better than 11 out of 21 circuits in terms of the average fault coverage. Furthermore, our generalized HM based on GATEST as the problem solver is better than the original GATEST in 7 out of 19 circuits in terms of both the average and maximum fault coverages. Note that the average fault coverages of our generalized HM are better than or equal to the original GATEST in all subdomains used in the heuristics-design process. Our results show that our generalization procedure can discover good HMs that work better than the original HMs.

Table 6.11 also indicates that HITEC is still better than our new generalized HM for CRIS in most of the circuits (in 14 out of 21 in terms of the maximal fault coverage and in 17 out of 21 in terms of the average fault coverage). This poor performance happens because our generalized HM is bounded by the limitations in CRIS and by our HM generator for

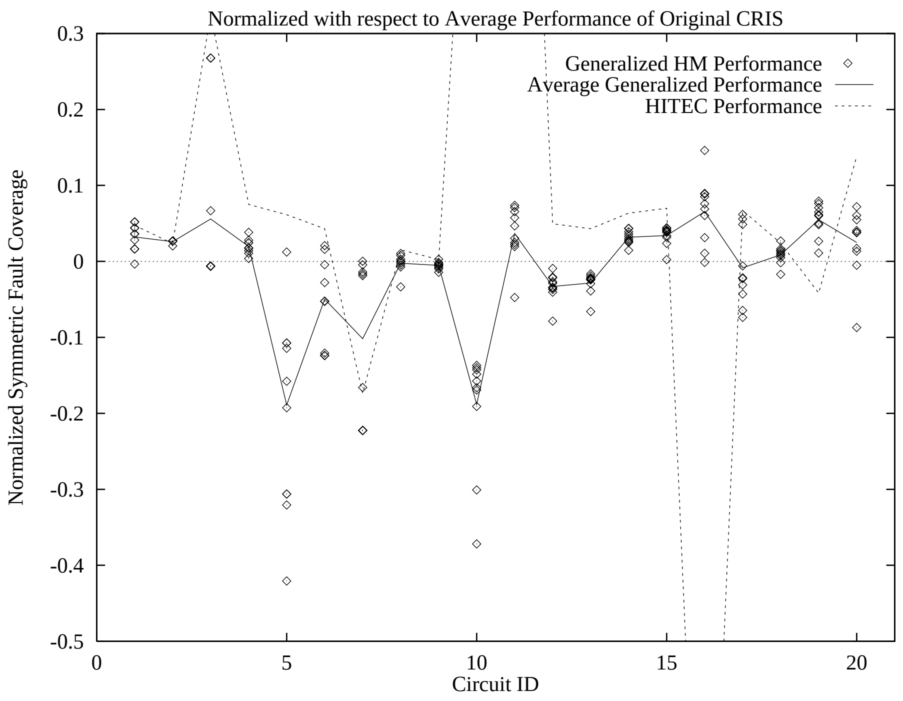


Figure 6.3: Distribution of normalized symmetric fault coverages of our generalized HM with respect to average fault coverages of the original CRIS on 20 benchmark circuits (s298, s344, s382, s386, s400, s444, s526, s641, s713, s820, s832, s1196, s1238, s1488, s1494, s1423, s5378, am2910, div16, and tc100 in that order).

CRIS. Such limitations cannot be overcome without generating more powerful HMs in our HM generator or without using better test-pattern generators like HITEC as our baseline problem solver.

Finally, we plot the distributions of symmetric fault coverages of our generalized HMs normalized with respect to average fault coverages of the original CRIS (Figure 6.3) and GATEST (Figure 6.4). These plots clearly demonstrate improvements over the original systems.

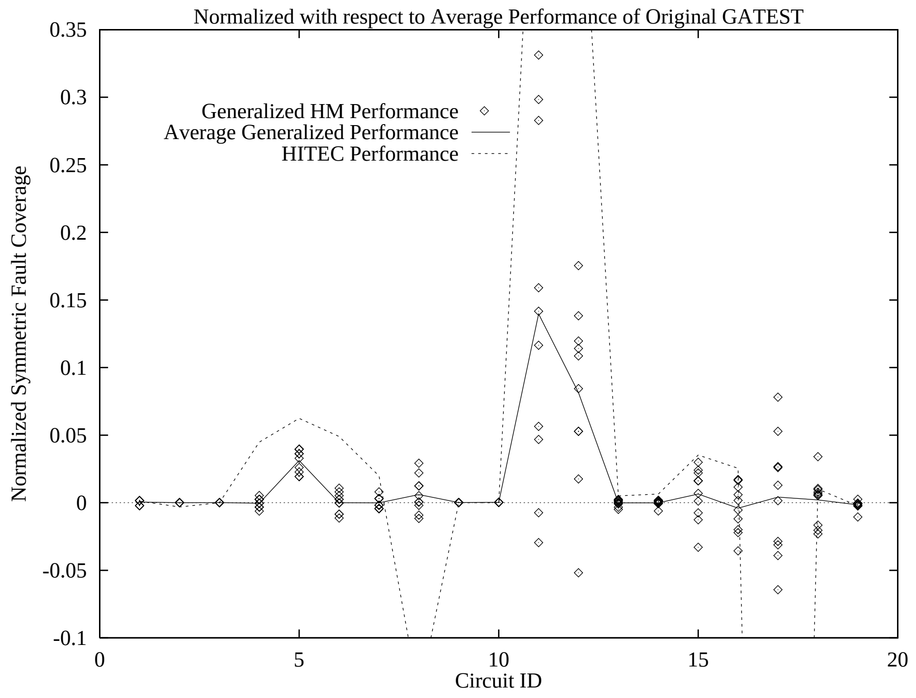


Figure 6.4: Distribution of normalized symmetric fault coverages of our generalized HM with respect to average fault coverages of the original GATEST on 19 benchmark circuits (s298, s344, s349, s382, s386, s400, s444, s526, s641, s713, s820, s832, s1196, s1238, s1488, s1494, s1423, s5378, and s35932 in that order).

6.4 Placement and Routing of VLSI Circuits

In this section, we choose TimberWolf [118] as our target problem solver for the VLSI placement-and-routing application. We first describe the target application and problem solver. We then present our results in designing new HMs for TimberWolf.

6.4.1 TimberWolf

TimberWolf is a software package based on simulated annealing [119] to place and route various components (transistors, resistors, capacitors, wires, etc.) on a piece of silicon. Its goal is to minimize the chip area needed while satisfying constraints such as the number of layers of poly-silicon for routing and the maximum signal delay through the paths. Its operations can be divided into three steps: placement, global routing, and detailed routing.

The placement and routing problem is NP-hard; hence, heuristics are generally used. Simulated annealing (SA) used in TimberWolf is an efficient method to randomly search the space of possible placements. Although in theory SA converges asymptotically to the global optimum with probability one, the results generated in finite time are usually suboptimal. As a result, there is a trade-off between quality of a result and cost (or computational time) of obtaining it. In TimberWolf version 6.0, the version we have studied in this subsection, there are two parameters to control the running time (which indirectly control the quality of the result): *fast-n* and *slow-n*. The larger the *fast-n* is, the shorter amount of time SA will run. In contrast, the larger the *slow-n* is, the longer amount of time SA will run. Of course, only one of these parameters can be used in a single experiment to control the running time.

TimberWolf has six major components: *cost function*, *generate function*, *initial temperature*, *temperature decrement*, *equilibrium condition*, and *stopping criterion*. Many parameters in these components have been tuned manually. However, their settings are generally heuristic because we lack domain knowledge for setting them optimally. In Table 6.12, we list the parameters we have focused on in our experiments. Our goal is to illustrate the power of our learning and generalization procedures and to show improved quality and reduced cost

Table 6.12: Parameters in TimberWolf (version 6) used in our HM in learning and in generalization.

Param.	Range	Step	Meaning	Default	Learned
P_1	0.1 - 2.5	0.1	vertical path weight for estimating the cost function	1.0	0.9584
P_2	0.1 - 2.5	0.1	vertical wire weight for estimating the cost function	1.0	0.2315
P_3	3 - 10	1	orientation ratio	6	10
P_4	0.33 - 2.0	0.1	range limiter window change ratio	1.0	1.2987
P_5	10.0 - 35.0	1.0	high temperature finishing point	23.0	10.0416
P_6	50.0 - 99.0	1.0	intermediate temperature finishing point	81.0	63.6962
P_7	100.0 - 150.0	1.0	low temperature finishing point	125.0	125.5509
P_8	130.0 - 180.0	1.0	final iteration temperature	155.0	147.9912
P_9	0.29 - 0.59	0.01	critical ratio that determines acceptance probability	0.44	0.3325
P_{10}	0.01 - 0.12	0.01	temperature for controller turn off	0.06	0.1124
P_{11}	integer	1	seed for the random number generator	see Table 6.9	

for the placement and routing of large circuits, despite the fact that only small circuits were used in learning and in generalization.

6.4.2 Experimental results

In our experiments, we used seven benchmark circuits [120] whose specifications are shown in Table 6.13. Here, we have only studied the application of TimberWolf to standard-cell placement, though other kinds of placement (such as macro/custom-cell placement and gate-array placement) can be studied in a similar fashion. In our experiments, we used *fast-n* values of 1, 5, and 10, respectively. We first applied TEACHER to learn good HMs for circuits *s298* with *fast-n* of 1, *s420* with *fast-n* of 5, and *primary1* with *fast-n* of 10, each of which was taken as a learning subdomain. We used a fixed sequence of ten random seeds (P_{11} in Table 6.12 with values from Table 6.9) in each subdomain to find the statistical performance of an HM. Each learning experiment involved 1000 applications of TimberWolf divided into ten generations. Based on the best 30 HMs (ten from each subdomain), we

Table 6.13: Benchmark circuits used in our experiments [120].

Cell Name	Cells	Nets	Pins	Implicit Feedthru
fract	124	163	454	0
s298	133	138	741	98
s420	211	233	1488	112
primary1	766	1172	5534	0
struct	1888	1920	5407	0
primary2	3014	3817	12014	0
industrial1	2271	2597	—	—

applied our generalization procedure from Section 5.4.1 to obtain one generalized HM. This generalized HM and the default HM are shown in Table 6.12.

Table 6.14 compares the quality (average/maximum area of chip) and cost (average execution time) between the generalized HM and the default HM on all seven circuits with *fast-n* values of 1, 5, and 10, respectively. (The cost for finding the minimum area is ten times the average cost.)

Figure 6.5 plots the quality (higher quality in the y-axis means reduced chip area averaged over ten runs using the defined random seeds) and cost (average execution time of TimberWolf) between the generalized HM and the default HM on all seven circuits with *fast-n* values of 1, 5, and 10, respectively. Note that all performance values in Figure 6.5 are normalized with respect to those of a *fast-n* of 10 and that the positive (resp., negative) portion of the x-axes shows the fractional improvement (resp., degradation) in computational cost with respect to the baseline HM using a *fast-n* of 10 for the same circuit. Each arrow in this figure points from the average performance of the default HM to the average performance of the generalized HM.

Among the 21 test cases, the generalized HM has worse quality than that of the default in only two instances (both for circuit *fract*) and has worse cost in 4 out of 21 cases. We see in Figure 6.5 that most of the arrows point in a left-upward direction, implying improved quality and reduced cost. Note that these experiments are meant to illustrate the power

Table 6.14: Comparison of cost-quality trade-offs between our generalized HM and the default HM on seven benchmark circuits and three *fast-n* values over ten runs. (All performance values except the defaults with *fast-n*=10 are in the form of symmetric improvements with respect to the default performance values of *fast-n*=10. Percentage improvement in quality means percentage decrease in chip area. Boldface indicates performance values worse than the corresponding values of the default HM.)

Circuit	Performance Measure	*fast-n*=10		*fast-n*=5		*fast-n*=1	
		Default	Generalized	Default	Generalized	Default	Generalized
s298	Avg. Quality	655706	0.021	0.008	0.038	0.021	0.049
	Max. Quality	640668	0.037	0.038	0.060	0.032	0.064
	Avg. Cost	30.05	−0.062	0.392	0.361	3.284	3.205
s420	Avg. Quality	858110	0.027	0.012	0.040	0.033	0.066
	Max. Quality	850662	0.071	0.030	0.067	0.046	0.078
	Avg. Cost	44.59	−0.091	0.420	0.307	3.268	3.069
fract	Avg. Quality	87524	0.065	0.058	**0.029**	0.096	**0.084**
	Max. Quality	77248	0.177	0.162	0.114	0.183	0.201
	Avg. Cost	32.65	−0.040	0.346	0.313	2.502	2.424
primary1	Avg. Quality	3542722	0.108	0.075	0.156	0.155	0.191
	Max. Quality	3413835	0.150	0.106	0.184	0.187	0.244
	Avg. Cost	300.08	−0.045	0.267	0.197	2.525	2.501
struct	Avg. Quality	2029258	0.299	0.226	0.581	0.666	1.020
	Max. Quality	1894878	0.368	0.293	0.728	0.774	1.179
	Avg. Cost	1174.23	−0.266	0.044	−0.136	1.505	1.286
primary2	Avg. Quality	17930315	0.089	0.142	0.209	0.280	0.334
	Max. Quality	17111600	0.167	0.200	0.287	0.329	0.363
	Avg. Cost	2291.71	**0.111**	0.192	**0.218**	2.210	2.044
industry1	Avg. Quality	16337967	0.021	0.037	0.057	0.065	0.076
	Max. Quality	15855016	0.048	0.052	0.077	0.078	0.088
	Avg. Cost	1676.05	**0.117**	0.276	0.191	2.375	**2.408**

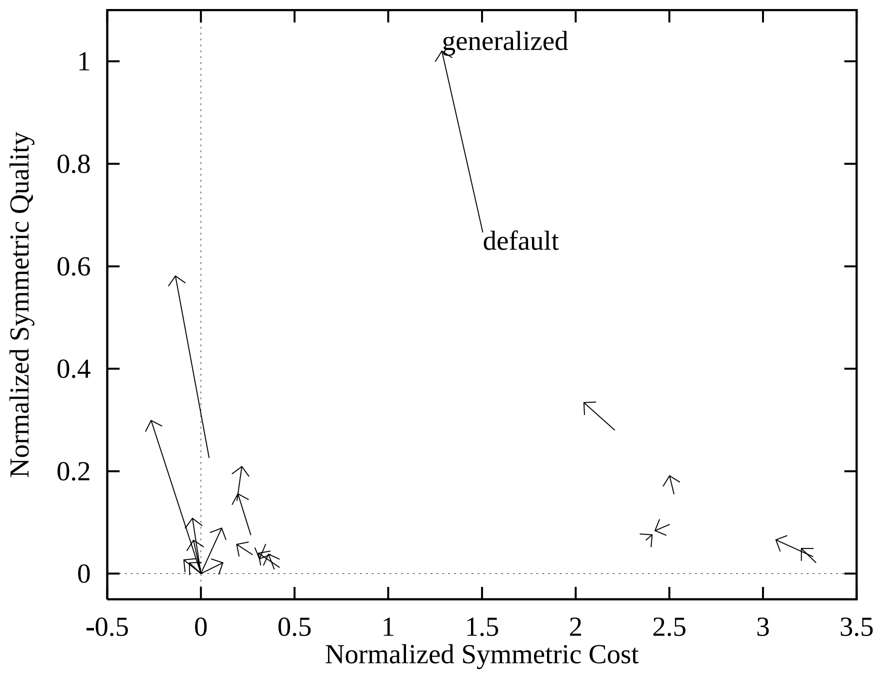


Figure 6.5: Comparison of normalized average performance between the default and the generalized HMs. The plots are normalized with respect to the performance of the baseline HM on each circuit using *fast-n* = 10. (See Eq. (2.8) for definition of normalized symmetric values.)

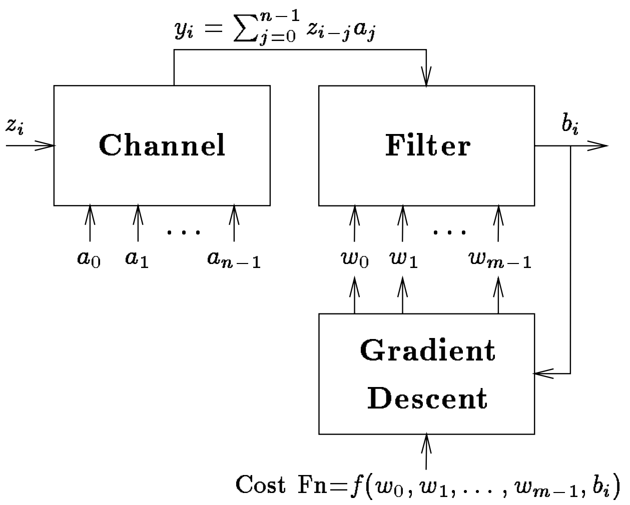

$y_i = \sum_{j=0}^{n-1} z_{i-j} a_j$
z_i
Channel
Filter
b_i
a_0 a_1 $\ldots$ a_{n-1}
w_0 w_1 $\ldots$ w_{m-1}
Gradient Descent
Cost Fn$=f(w_0, w_1, \ldots, w_{m-1}, b_i)$

Figure 6.6: Blind equalization process for recovering the input data stream for an n^{th} order channel and an m^{th} order filter.

of our generalization procedure. We expect to see more improvement as we learn other functions and parameters in TimberWolf.

6.5 Blind Equalization

The final problem solver is a gradient descent algorithm to find a set of weights of an FIR filter in order to minimize convergence time, number of accumulated errors, and complexity of the filter [19]. The heuristics involved are the initial weights of the descent algorithm and the cost function defined in the weight space for the descent algorithm.

We have applied our TEACHER system to design the cost function for blind equalization. The goal is to minimize the number of accumulated errors for a sequence of input data corrupted in transmission (Figure 6.6). The equalization process is equivalent to adjusting the weights of an FIR filter in order to minimize the value of a cost function (by gradient descent). In our experiments, the cost function is defined in terms of the weights of the filter and the current output of the filter.

Table 6.15: Summary of third-order channels used in our blind equalization experiment. Note that the channel weights can be in any order for a given subdomain.

Channel	Channel Weight Specification		
	A	B	C
1	0.8704	0.3482	0.3482
2	0.8704	0.3482	-0.3482
3	0.8193	0.4916	0.2950
4	0.8193	-0.4916	0.2950
5	0.7893	0.4341	0.4341
6	0.7893	0.4341	-0.4341
7	0.7908	0.5140	0.3322
8	0.7908	0.5140	-0.3322
9	0.5774	0.5774	0.5774
10	0.5774	0.5774	-0.5774

In this application, we define a test case as multiple random sequences of data of fixed length passing through a fixed channel and a blind equalizer with a fixed set of random initial weights. Each sequence of data is generated by a random number generator with an equal probability of each data value being zero or one. Each test case specifies a random seed for generating the first data value for the first sequence.

We further define all test cases with the same channel specification as belonging to one subdomain. In our experiments, we attempt to cover the entire spectrum of all possible third-order channels from relatively easy ones ($|a_i| > \sum_{i \neq j} |a_j|$ where a_i is the i^{th} weight of the channel) to the hardest one ($a_i = a_j$ for all i and j). All ten channels used in our experiment are shown in Table 6.15.

Table 6.16 shows the average symmetric improvement in terms of the number of accumulated errors, HM_{base} (CMA 2-2) [19], and the new HM found after learning and generalization.

Table 6.16: Summary of average symmetric improvements in terms of the number of accumulated errors for the learned cost function over ten subdomains. (b_i in Figure 6.6 is the instantaneous value of b.)

Average Sym. Improvement				Orig.	New
Avg.	Std. Dev.	Max.	Min.	HM	HM
0.153	0.395	0.694	−0.465	$b^3 - b$	$4b^3 - 2b^2 Sign(b) - b$

6.6 Summary

In this chapter, we have applied our automated system for designing knowledge-lean heuristics to several real-world applications. The targeted problem solvers include:

(a) process mapping using PGA,

(b) branch-and-bound search on several combinatorial optimization problems,

(c) CRIS and GATEST for generating test-patterns for VLSI circuits,

(d) TimberWolf for placement and routing of VLSI circuits, and

(e) blind equalization using gradient descent approach.

The experiments shown in this section cover a wide variety of situations. The forms of HMs include a set of numerical parameters (CRIS and TimberWolf), a symbolic function (blind equalizer, GATEST fitness function, and B&B decomposition heuristic), and a set of symbolic rules with numerical threshold values (PGA). Both single-objective and multiple-objective cases are covered in our experiments. We also study the case in which the target problem solver has an incumbent HM and the less frequent case in which the target problem solver does not have an incumbent HM.

Our results consistently show the effectiveness of our heuristics-design system for providing new HMs that improve the performance of their problem solvers. For all of the applications, we have discovered HMs that are comparable or are better than the incumbent HM (when it exists). Our results are especially significant when we consider the fact that incumbent HMs for these problem solvers have been extensively hand-tuned and have been applied to solve many application problems.

7. CONCLUSIONS AND FUTURE WORK

7.1 Summary of Work Accomplished

In this thesis, we have developed an automated system for designing heuristics for knowledge-lean problem solvers using genetics-based machine-learning methods. There are many applications and problem solvers that fit the characteristics targeted by our system. The heuristics for these applications and problem solvers are relatively difficult to design and can have a large impact on the problem-solving process.

To summarize, we have made the following contributions in this thesis:

- We have identified five key issues that must be addressed in the development of an automated system for designing knowledge-lean heuristics. These issues include: the decomposition and integration of problem-solving components, the classification of a problem domain, the generation of new heuristics, the performance evaluation of heuristics, and the statistical generalization of learned heuristics (Chapter 3).

- We have developed and implemented a systematic framework for designing knowledge-lean heuristics using various strategies to address the above key issues. This framework is modular so that each key issue is addressed independently. In addition, we have isolated the functionalities that are application and problem-solver dependent so that we can easily interface our framework to new applications and problem solvers.

TEACHER, the implementation of our heuristics-design framework, has been applied to develop new heuristics for several applications (Chapter 3).

- We have developed a method for partitioning a problem domain into smaller subsets, called *subdomains*, when there is different statistical performance behavior in different parts of the problem domain. Within each subdomain, all performance values are *independent and identically distributed (IID)* and can be combined and compared with one another. Across different subdomains, performance values must be treated separately and independently (Chapters 2 and 3).

- We have identified the various problems that must be addressed in the performance evaluation of heuristics within a subdomain and have developed a systematic approach to address these issues. First, a normalization method can be applied to reduce performance variations and provide relative performance measures. Second, when there are multiple performance measures, all but one measure should be transformed into constrained measures. Third, the true performance of each HM over a subdomain is statistically estimated, based on sample means and an incomplete subset of test cases. The uncertainty involved in this statistical estimation can also be computed using the *probability-of-win* (P_{win}) measure (Chapters 2 and 5).

- We have studied methods to find good HMs that can generalize to the entire problem domain, including unlearned subdomains. Using a range-independent measure called probability of win for comparing against the current incumbent HM, we can compare and order heuristics across problem subdomains in a uniform manner. The worst-case performance of an HM over a given set of subdomains has been selected as an indicator of that HM's performance over the entire problem domain (Chapter 5).

- We have studied strategies to schedule resources for tests in the heuristics-design process. An improvement over previous strategies is that our strategy is nonparametric and does not rely on the underlying performance distribution of heuristics. We have also proposed a scheduling strategy to cope with one or more learning objectives. Our results show that scheduling is important when tests are expensive and test results are noisy (Chapter 4).

- Using TEACHER, we have found better heuristics for process mapping, branch-and-bound search, test-pattern generation in circuit testing, VLSI cell placement and routing, and blind equalization. These experiments cover many different situations including different forms of HMs (*e.g.*, a set of numerical parameters, a symbolic function, and a set of symbolic rules), multiple performance objectives, and whether an incumbent HM exists for the target problem solver (Chapter 6).

7.2 Future Work

In this section, we present some possible avenues for research in the future.

- *Decomposition and Integration of Problem-Solver Components.* We plan to integrate this issue into our framework for designing knowledge-lean heuristics. The investigation of this issue is finally possible after we have developed effective solutions to the other issues in the heuristics-design process. We plan to study various methods for decomposing the problem solver into smaller groups and their interactions when heuristics for each group are learned sequentially.

- *Classification of Problem Domains.* An automated method for identifying problem subdomains for learning and subspaces for generalization is desirable. Since such demarcation is generally vague and imprecise, we plan to apply fuzzy sets to help define subdomains and subspaces. Fuzzy logic can also help identify heuristics that can be generalized, especially when there are multiple objectives in the application.

- *Generation of New HMs.* Better and possibly domain-dependent methods for generating new HMs should be investigated and integrated into TEACHER. Studies on different methods for selection will also be investigated.

- *Evaluation of HMs.* We plan to develop a better normalization method that can avoid anomalies when more than two HMs are considered. In addition, we plan to study other metrics for performance evaluation besides the average metrics studied in this thesis. One such metric is the median metric, which is more robust to extreme performance values and less sensitive to normalization methods.

- *Generalization of Learned HMs.* We need to find better generalization strategies when there is no incumbent HM or when the incumbent HM is not acceptable. Using the worst normalized performance values over multiple subdomains may not be meaningful since different subdomains may have entirely different ranges of normalized performance values. In addition, we must investigate the robustness of our probability-of-win measure, P_{win}.

- *Applications of TEACHER.* Finally, we plan to carry out learning on more applications. The merits of our system, of course, lie in finding better heuristics for real-world problems, which may involve many contradicting objectives. Our experience in this thesis is on an application with two objectives. To cope with applications with many objectives, we need to extend our scheduling and generalization strategies.

REFERENCES

[1] G. F. DeJong and R. J. Mooney, "Explanation-based learning: An alternative view," *Mach. Learn.*, vol. 1, no. 2, pp. 145–176, 1986.

[2] R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans," *Artif. Intell.*, vol. 3, pp. 251–288, 1972.

[3] J. J. Grefenstette, C. L. Ramsey, and A. C. Schultz, "Learning sequential decision rules using simulation models and competition," *Mach. Learn.*, vol. 5, pp. 355–381, 1990.

[4] R. Keller, "Defining operationality for explanation-based learning," in *Proc. National Conf. on Artificial Intelligence*, Seattle, Washington, AAAI, Inc., June 1987, pp. 482–7.

[5] J. E. Laird, P. S. Rosenbloom, and A. Newell, "Chunking in SOAR: The anatomy of a general learning mechanism," *Mach. Learn.*, vol. 1, no. 1, pp. 11–46, 1986.

[6] S. Minton, J. G. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil, "Explanation-based learning: A problem solving perspective," in *Machine Learning: Paradigms and Methods*, J. Carbonell, Ed., Cambridge, MA: M.I.T. Press, 1990, pp. 63–118.

[7] T. M. Mitchell, P. E. Utgoff, and R. B. Banerji, "Learning by experimentation: Acquiring and refining problem-solving heuristics," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds., Tioga, 1983.

[8] T. Mitchell, R. Keller, and S. Kedar-Cabelli, "Explanation-based generalization: A unifying view," *Mach. Learn.*, vol. 1, no. 1, pp. 47–80, 1986.

[9] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev.*, vol. 3, pp. 210–229, 1959.

[10] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," in *Machine Learning: Paradigms and Methods*, J. Carbonell, Ed., Cambridge, MA: M.I.T. Press, 1990.

[11] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 4, pp. 4–22, Apr. 1987.

[12] L. A. Rendell, "A new basis for state-space learning systems and a successful implementation," *Artif. Intell.*, vol. 20, pp. 369–392, 1983.

[13] B. G. Buchanan and E. H. Shortliffe, *Rule-Based Experts Programs: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley, 1984.

[14] E. Rich and K. Knight, *Artificial Intelligence.* New York, NY: McGraw Hill, 1991.

[15] D. G. Saab, Y. G. Saab, and J. A. Abraham, "CRIS: A test cultivation program for sequential VLSI circuits," in *Proc. of Int. Conf. on Computer Aided Design*, Santa Clara, CA, IEEE, Nov. 8-12 1992, pp. 216–219.

[16] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing.* Boston, MA: Kluwer Academic Publishers, 1988.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability.* San Francisco, CA: Freeman, 1979.

[18] J. C. Yan and S. F. Lundstrom, "The post-game analysis framework–developing resource management strategies for concurrent systems," *IEEE Trans. Knowl. Data Eng.*, vol. 1, pp. 293–309, Sept. 1989.

[19] S. Haykin, *Blind Deconvolution.* Englewood Cliffs, NJ: Prentice Hall, 1994.

[20] P. Mehra and B. W. Wah, *Load Balancing: An Automated Learning Approach.* River Edge, NJ: World Scientific Publishing Co. Pte. Ltd., 1995.

[21] S. R. Schwartz and B. W. Wah, "Automated parameter tuning in stereo vision under time constraints," in *Proc. Int. Conf. on Tools for Artificial Intelligence*, IEEE, Nov. 1992, pp. 162–169.

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

[23] M. H. Romanycia and F. Pelletier, "What is a heuristic?," *Comput. Intell.*, vol. 1, pp. 47–58, 1985.

[24] A. Newell, J. C. Shaw, and H. A. Simon, "Programming the logic theory machine," *Proc. 1957 Western Joint Computer Conf.*, pp. 230–240, 1957.

[25] L.-C. Chu, "Algorithms for combinatorial optimization in real time and their automated refinement by genetic programming," Ph.D. dissertation, Dept. of Electrical and Computer Engineering, Univ. of Illinois, Urbana, IL, May 1994.

[26] C. L. Ramsey and J. J. Grefenstette, "Case-based initialization of genetic algorithms," in *Proc. of the Fifth Int. Conf. Genetic Algorithms*, Int. Soc. for Genetic Algorithms, June 1993, pp. 84–91.

[27] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley Pub. Co., 1989.

[28] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Mach. Learn.*, vol. 3, pp. 95–100, Oct. 1988.

[29] J. R. Koza, *Genetic Programming.* Cambridge, MA: M.I.T. Press, 1992.

[30] R. Weinberg, "Computer simulation of a living cell," Ph.D. dissertation, Univ. of Michigan, Ann Arbor, MI, 1970.

[31] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-16, pp. 122–128, Jan. 1986.

[32] R. E. Mercer, "Adaptive search using a reproductive meta-plan," Unpublished master's thesis, Univ. of Alberta, Edmonton, CA, 1977.

[33] R. E. Mercer and J. R. Sampson, "Adaptive search using a reproductive meta-plan," *Kybernetes*, vol. 7, pp. 215–228, 1978.

[34] H. A. Simon, "A comparison of game theory and learning theory," *Psychometrika*, vol. 21, pp. 267–272, 1956.

[35] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "Sequential circuit test generation in a genetic algorithm framework," in *Proc. Design Automation Conf.*, ACM/IEEE, June 1994.

[36] B. W. Wah, A. Ieumwananonthachai, L. C. Chu, and A. Aizawa, "Genetics-based learning of new heuristics: Rational scheduling of experiments and generalization," *IEEE Trans. Knowl. Data Eng.*, vol. 7, pp. 763–785, Oct. 1995.

[37] C. Sechen and A. S. Vincentelli, "The timberwolf placement and routing package," *J. of Solid-State Circuits*, vol. 20, no. 2, pp. 510–522, 1985.

[38] B. W. Wah and L.-C. Chu, "Combinatorial search algorithms with meta-control: Modeling and impl ementations," *Int. J. Artif. Intell. Tools*, vol. 1, pp. 369–397, Sept. 1992.

[39] J. C. Yan, "Post-game analysis–A heuristic resource management framework for concurrent systems," Ph.D. dissertation, Dept. Elec. Eng., Stanford Univ., Dec. 1988.

[40] J. C. Yan, "New "post-game analysis" heuristics for mapping parallel computations to hypercubes," in *Proc. Int. Conf. on Parallel Processing*, vol. II, Boca Raton, FL, CRC Press, Aug. 1991, pp. 236–242.

[41] A. Ieumwananonthachai and B. W. Wah, "TEACHER – an automated system for learning knowledge-lean heuristics," Tech. Rep. CRHC-95-08, Center for Reliable and High Performance Computing, Coordinated Science Laboratory, Univ. of Illinois, Urbana, IL, Mar. 1995.

[42] A. Ieumwananonthachai, A. Aizawa, S. R. Schwartz, B. W. W. h, and J. C. Yan, "Intelligent process mapping through systematic improvement of heuristics," *J. Parallel Distrib. Comput.*, vol. 15, pp. 118–142, June 1992.

[43] A. Ieumwananonthachai, A. N. Aizawa, S. R. Schwartz, B. W. Wah, and J. C. Yan, "Intelligent mapping of communicating processes in distributed computing systems," in *Proc. Supercomputing 91*, Albuquerque, NM, ACM/IEEE, Nov. 1991, pp. 512–521.

[44] J. L. Devore, *Probability and Statistics for Engineering and the Sciences.* Monterey, CA: Brooks/Cole Pub. Co., 1982.

[45] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1. New York, NY: John Wiley & Sons, Inc, 1968.

[46] N. R. Ullman, *Statistics: An Applied Approach.* Lexington, MA: Xerox College Publishing, 1972.

[47] J. E. Walsh, *Handbook of Nonparametric Statistics*, vol. 1. Princeton, NJ: D. Van Nostrand Company, Inc., 1962.

[48] B. W. Wah, A. Ieumwananonthachai, and Y. C. Li, "Generalization of heuristics learned in genetics based learning," in *Genetic Algorithms and Pattern Recognition*, (S. K. Pal and P. Wang, Eds.), CRC Press, to be published.

[49] V. Kumar and L. N. Kanal, "A general branch and bound formulation for understanding and synthesizing and/or tree search procedures," *Artif. Intell.*, vol. 21, no. 1-2, pp. 179–198, 1983.

[50] E. L. Lawler and D. W. Wood, "Branch and bound methods: A survey," *Oper. Res.*, vol. 14, pp. 699–719, 1966.

[51] B. W. Wah, "Population-based learning: A new method for learning from examples under resource constraints," *IEEE Trans. Knowl. Data Eng.*, vol. 4, pp. 454–474, Oct. 1992.

[52] E. Kreyszig, *Introductory Mathematical Statistics: Principles and Methods.* New York, NY: John Wiley & Sons, Inc., 1970.

[53] J. E. Walsh, *Handbook of Nonparametric Statistics*, vol. II. Princeton, NJ: D. Van Nostrand Company, Inc., 1965.

[54] B. W. Wah, A. Ieumwananonthachai, S. Yao, and T. Yu, "Statistical generalization: Theory and applications (plenary address)," in *Proc. Int. Conf. on Computer Design*, Austin, TX, IEEE, Oct. 1995, pp. 4–10.

[55] F. W. Gembicki, "Vector optimization for control with performance and parameter sensitivity indices," Ph.D. dissertation, Case Western Reserve University, Cleveland, OH, 1974.

[56] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* Englewood Cliffs, NJ: Prentice-Hall Inc., 1982.

[57] H. R. Neave and P. L. Worthington, *Distribution-Free Tests.* London, UK: Unwin Hyman, 1988.

[58] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed. Cambridge, UK: Cambridge University Press, 1992.

[59] W. G. Bulgren, *Discrete System Simulation.* Englewood Cliffs, NJ: Prentice Hall, Inc., 1982.

[60] A. Ieumwananonthachai and B. W. Wah, "Statistical generalization of performance-related heuristics for knowledge-lean applications," in *Proc. Int. Conference on Tools for Artificial Intelligence*, Houston, TX, IEEE, Nov. 1995, pp. 174–181.

[61] A. Ieumwananonthachai and B. W. Wah, "Statistical generalization of performance-related heuristics for knowledge-lean applications," *Int. J. Artif. Intell. Tools, Archit. Lang. Algorithms*, vol. 4, (accepted to appear) 1996.

[62] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 2. New York, NY: John Wiley & Sons, Inc, 1971.

[63] B. G. Tabachnick and L. S. Fidell, *Using Multivariate Statistics*, 2nd ed. New York, NY: Harper and Row, 1989.

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

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

[66] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing.* Boston, MA: Kluwer Academic Pub., 1987.

[67] R. S. Michalski, "Understanding the nature of learning: Issues and research directions," in *Machine Learning: An Artificial Intelligence Approach*, vol. II, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds., Los Altos, CA: Morgan Kaufmann Publishers Inc., 1986, pp. 3–25.

[68] T. M. Mitchell, P. E. Utgoff, B. Nudel, and R. Banerji, "Learning problem-solving heuristics through practice," in *Proc. 7th Int. Joint Conf. on Artificial Intelligence*, Los Altos, CA, William Kaufman Inc., 1981, pp. 127–134.

[69] D. Sleeman, P. Langley, and T. M. Mitchell, "Learning from solution paths: An approach to the credit assignment problem," *The AI Magazine*, vol. 3, pp. 48–52, 1982.

[70] M. Minsky, "Steps toward artificial intelligence," in *Computers and Thought*, E. A. Feigenbaum and J. Feldman, Eds., New York: McGraw-Hill, 1963, pp. 406–450.

[71] T. G. Dietterich and B. G. Buchanan, "The role of critic in learning systems," Tech. Rep. STAN-CS-81-891, Stanford Univ., CA, Dec. 1981.

[72] R. Desimone, "Learning control knowledge within an explanation-based learning framework," in *Progress in Machine Learning*, I. Bratko and N. Lavrac, Eds., Cheshire, UK: Sigma Press, 1987, pp. 107–119.

[73] G. W. Ernst and M. M. Goldstein, "Mechanical discovery of classes of problem-solving strategies," *J. Assoc. Comput. Mach.*, vol. 29, pp. 1–23, Jan. 1982.

[74] A. Barr and E. A. Feigenbaum, *The Handbook of Artificial Intelligence*, vols. 1-3. Los Altos, CA: William Kaufmann Inc., 1981.

[75] R. G. Smith, T. M. Mitchell, R. A. Chestek, and B. G. Buchanan, "A model for learning systems," in *Proc. 5th Int. Joint Conf. on Artificial Intelligence*, Los Altos, CA, William Kaufman Inc., Aug. 1977, pp. 338–343.

[76] P. Langley, "Learning to search: From weak methods to domain-specific heuristics," *Cogn. Sci.*, vol. 9, pp. 217–260, 1985.

[77] T. M. Mitchell, "Learning and problem solving," in *Proc. 8th Int. Joint Conf. on Artificial Intelligence*, Los Altos, CA, William Kaufman Inc., Aug. 1983, pp. 1139–1151.

[78] B. Widrow, N. K. Gupta, and S. Maitra, "Punish/reward: Learning with a critic in adaptive threshold systems," *IEEE Trans. Syst, Man Cybern.*, vol. SMC-3, no. 5, pp. 455–465, 1973.

[79] A. H. Klopf, "Drive-reinforcement learning: A real-time learning mechanism for unsupervised learning," in *Proc. Int. Conf. on Neural Networks*, vol. II, IEEE, 1987, pp. 441–445.

[80] R. S. Sutton and A. G. Barto, "Toward a modern theory of adaptive networks: Expectation and predicition," *Psych. Review*, vol. 88, no. 2, pp. 135–170, 1984.

[81] D. B. Lenat, "Theory formation by heuristic search; the nature of heuristics II: Background and examples," *Artif. Intell.*, vol. 21, pp. 31–59, 1983.

[82] A. L. Samuel, "Some studies in machine learning using the game of checkers II–recent progress," *IBM J. Res. Dev.*, vol. 11, no. 6, pp. 601–617, 1967.

[83] R. J. Williams, "On the use of backpropagation in associative reinforcement learning," in *Proc. Int. Conf. on Neural Networks*, vol. I, IEEE, July 1988, pp. 263–270.

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

[85] D. Nguyen and B. Widrow, "The truck backer-upper: An example of self-learning in neural networks," in *Proc. Int. Joint Conf. on Neural Networks*, vol. II, IEEE, 1989, pp. 357–363.

[86] S. Chien, J. Gratch, and M. Burl, "On the efficient allocation of resources for hypothesis evaluation: A statistical approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, pp. 652–665, July 1995.

[87] J. Heitkötter and D. Beasley, Eds., *The Hitch-Hiker's Guide to Evolutionary Computation: A List of Frequently Asked Questions (FAQ).* USENET: comp.ai.genetic, 1995. Anonymous FTP: rtfm.mit.edu:/pub/usenet/news.answers/ai-faq/genetic/.

[88] W. M. Spears, K. A. DeJong, T. Bäck, D. Fogel, and H. de Garis, "An overview of evolutionary computing," in *Proc. European Conf. on Machine Learning*, New York, NY, Springer-Verlag, 1993, pp. 442–459.

[89] K. DeJong, "Learning with genetic algorithms: An overview," *Mach. Learn.*, vol. 3, pp. 121–138, Oct. 1988.

[90] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 1, fundamentals," *Univ. Comput.*, vol. 15, no. 2, pp. 58–69, 1993.

[91] A. K. Aizawa and B. W. Wah, "Scheduling of genetic algorithms in a noisy environment," *Evol. Comput.*, vol. 2, no. 2, pp. 97–122, 1994.

[92] J. M. Fitzpatrick and J. J. Grefenstette, "Genetic algorithms in noisy environments," *Mach. Learn.*, vol. 3, pp. 101–120, Oct. 1988.

[93] D. E. Goldberg, K. Deb, and J. H. Clark, "Genetic algorithms, noise, and the sizing of populations," *Complex Syst.*, vol. 6, 1992.

[94] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 2, research topics," *Univ. Comput.*, vol. 15, no. 4, pp. 170–181, 1993.

[95] K. DeJong and W. Spears, "On the state of evolutionary computation," in *Proc. Fifth Int. Conf. Genetic Algorithms*, Int. Soc. for Genetic Algorithms, June 1993, pp. 618–623.

[96] C. Z. Janikow, "A knowledge-intensive genetic algorithm for supervised learning," *Mach. Learn.*, vol. 13, pp. 189–228, November/December 1993.

[97] S. W. Wilson and D. E. Goldberg, "A critical review of classifier systems," in *Proc. of the Third Int. Conf. Genetic Algorithms*, Int. Soc. for Genetic Algorithms, June 1989, pp. 244–255.

[98] G. Robertson and R. Riolo, "A tale of two classifier systems," *Mach. Learn.*, vol. 3, pp. 139–160, Oct. 1988.

[99] S. F. Smith, "Flexible learning of problem solving heuristics through adaptive search," in *Proc. Int. Joint Conf. on Artificial Intelligence*, Morgan Kaufmann Publishers Inc., 1983, pp. 422–5.

[100] K. A. DeJong, W. M. Spears, and D. F. Gordon, "Using genetic algorithm for concept learning," *Mach. Learn.*, vol. 13, pp. 161–188, November/December 1993.

[101] C.-C. Teng and B. W. Wah, "An automated design system for finding the minimal configuration of a feed-forward neural network," in *Proc. Int. Conf. on Neural Networks*, IEEE, June 1994, pp. III–1295 – III–1300.

[102] M. B. Lowrie and B. W. Wah, "Learning heuristic functions for numeric optimization problems," in *Proc. Computer Software and Applications Conf.*, Chicago, IL, IEEE, Oct. 1988, pp. 443–450.

[103] C. F. Yu and B. W. Wah, "Learning dominance relations in combinatorial search problems," *IEEE Trans. on Softw. Eng.*, vol. SE-14, pp. 1155–1175, Aug. 1988.

[104] A. Aizawa and B. W. Wah, "Scheduling of genetic algorithms in a noisy environment," in *Proc. of the Fifth Int. Conf. Genetic Algorithms*, Int. Soc. for Genetic Algorithms, July 1993, pp. 48–55.

[105] A. N. Aizawa and B. W. Wah, "A sequential sampling procedure for genetic algorithms," *Comput. Math. Appl.*, vol. 27, pp. 77–82, May 1994.

[106] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, Univ. of Massachusetts, Amherst, MA, Feb. 1984.

[107] R. E. Bechhofer, "A single-sample multiple decision procedure for ranking means of normal populations with known variances," *Ann. Math. Stat.*, vol. 25, pp. 16–39, Mar. 1954.

[108] Y. L. Tong and D. E. Wetzell, "Allocation of observations for selecting the best normal population," in *Design of Experiments: Ranking and Selection*, T. J. Santner and A. C. Tamhane, Eds., New York, NY: Marcel Dekker, 1984, pp. 213–224.

[109] R. E. Bechhofer, A. J. Hayter, and A. C. Tamhane, "Designing experiments for selecting the largest normal mean when the variances are known and unequal: Optimal sample size allocation," *J. Stat. Plan. Inference*, vol. 28, pp. 271–289, 1991.

[110] T. S. Ferguson, *Mathematical Statistics: A Decision Theoretic Approach.* NY: Academic Press, 1967.

[111] B. K. Ghosh and P. K. Sen, Eds., *Handbook of Sequential Analysis.* New York, NY: Marcel Dekker, Inc., 1991.

[112] E. Lloyd, Ed., *Statistics*, vol. 6 of *Handbook of Applicable Mathematics.* New York, NY: Wiley & Sons Ltd., 1984.

[113] L.-C. Chu and B. W. Wah, "Optimization in real time," in *Proc. Real Time Systems Symp.*, IEEE, Nov. 1991, pp. 150–159.

[114] E. L. Crow, F. A. Davis, and M. W. Maxfield, *Statistics Manual.* New York, NY: Dover Publications, 1960.

[115] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization," in *Proc. of the Fifth Int. Conf. Genetic Algorithms*, Int. Soc. for Genetic Algorithms, June 1993, pp. 416–423.

[116] F. Brglez, D. Bryan, and K. Kozminski, "Combinatorial profiles of sequential benchmark circuits," in *IEEE Int. Symp. Circuits Syst. Proc.*, May 1989, pp. 1929–1934.

[117] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *European Design Automation Conference*, 1991, pp. 214–218.

[118] C. Sechen and A. S. Vincentelli, "Timberwolfe3.2 : A new standard cell placement and global routing package," in *Proc. 23rd Design Automation Conference*, IEEE/ACM, 1986, pp. 432–439.

[119] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[120] LayoutSynth92, *International Workshop on Layout Synthesis.* ftp site: mcnc.mcnc.org in directory /pub/benchmark, 1992.

VITA

Arthur Ieumwananonthachai received his B.S. degree in electrical engineering and computer science from the University of Washingtion, Seattle, WA, in 1986, and his M.S. degree in computer science from the University of California, Los Angeles, CA in 1988. He will receive the degree of Doctor of Philosophy in electrical and computer engineering from the University of Illinois at Urbana-Champaign in May 1996. After completing his doctoral degree, he will take a position as Software Designer with the NonStop Networking Division within Tandem Computers Incorporated in Cupertino, California.

His research interests include computer networks, distributed systems, and machine learning.