References

1. E. C. Koenig, T. J. Frederick, "Formal Analysis for a General Automaton," *Progress in Cybernetics*, Vol. 3, J. Rose, Ed., (Gordon and Breach, New York, 1970) 613-640.
2. E. C. Koenig, T. J. Frederick, "A General System of Interactive Automata: Analysis for Defining the Graph Model." *Cybernetica*, Belgium, Vol. 14, #2 (1971).
3. E. C. Koenig, "Analysis for a General Automaton with Distinguishable Receptors and Effectors," *IEEE Trans.*, SMC, Vol. 5, #1 (1975) 137-140.
4. D. B. Lenat, "CYC: A Large-Scale Investment in Knowledge Infrastructure," *Communications of the ACM*, Vol. 38, #11 (1995) 33-38.
5. B. Schott, T. Whalen, "Modus Ponens, Modus Tollens, and Fuzzy Relations in Goal Directed Inferences." *Proceedings of the 1987 IEEE International Conference on SMC*, Vol. 1 (1987) 173-176.
6. E. C. Koenig, "Some Principles for Robotics Based on General Automata," *Robotica*, Vol. 4, Part 1 (1986).
7. E. C. Koenig, "A Model Knowledge Structure for Robots," *Proceedings of the Annual Conference, 1987 IEEE, Systems, Man, and Cybernetics*, Alexandria, Virginia, U.S.A. (1987).
8. E. C. Koenig, "Parallel Processing Considerations for Interactive Man-Robot Systems," *Systems Analysis-Modeling-Simulation*, Vol. 16, #2 (1994).
9. E. C. Koenig, "Analysis for Correct Reasoning by Robots: Modus Ponens, Modus Tollens," *Proceedings of the 1989 IEEE International Phoenix Conference on Computers and Communications* (1989) 584-589.
10. E. C. Koenig, "Analysis for Correct Reasoning by Robots: Hypothetical Syllogism with Modus Ponens, Modus Tollens," *Proceedings of the 8th International Congress of Cybernetics and Systems*, Hunter College, City University of New York, New York City, NY, U.S.A. (1990).
11. E. C. Koenig, "Analysis for Correct Reasoning in Interactive Man-Robot Systems: Disjunctive Syllogism with Modus Ponens and Modus Tollens," *Artificial Intelligence in Industrial Decision Making, Control, and Automation*, S. G. Tzafestas and M. B. Verbruggen, eds., (Kluwer Academic Publishers, Netherlands, 1995).
12. L. A. Zadeh, "Fuzzy logic and approximate reasoning" (in memory of Grigore Moisil), Synthese 30 (1975) 407-428.
13. L. A. Zadeh, "Fuzzy sets and information granularity," *Advances in Fuzzy Set Theory and Applications*, R. Ragade and R. Yager, eds., (North-Holland, Amsterdam, 1979) 3-18.
14. D. E. Knuth, *The Art of Computer Programming: Fundamental Algorithms*, (Addison-Wesley, Reading, Massachusetts, 1968) 1.

CHAPTER 11

Genetics-Based Learning and Statistical Generalization

Benjamin W. Wah [1] *, Arthur Ieumwananonthachai* [2] *and Ting Yu* [1]

[1] Coordinated Science Laboratory, University of Illinois Urbana, IL 61801, U.S.A.

[2] NonStop Networking Division, Tandem Computers Inc. Cupertino, CA 95014 , U.S.A.

1. Introduction

Heuristics are generally used in many real-world engineering applications ranging from computer aided design, optimization, scheduling and computer communications. Since the relationship between performance and control is unknown in heuristics, some parameters, functions, and procedures are designed either based on user experience or experimentally. These heuristics can usually be improved by automated tuning, machine learning, and generalization. In this chapter, we study the problem of performance generalization of the heuristics learned.

1.1 Terminologies

We define a ***problem solver*** as an algorithm, or more generally, a software package used to solve a problem. A problem solver can be regarded as a black box, with some ***heuristic components*** or ***heuristics*** designed in an ad hoc way, where a heuristic is "A process that may solve a problem but offers no guarantees of doing so" [1]. Heuristic elements may appear in the form of numerical parameters, symbolic formulae, rules, and procedures.

The effectiveness of heuristics is evaluated in terms of their ***performance measures***. In this chapter, we are interested in problem solvers with numerical performance measures; examples of which include the quality of solutions found and the cost of getting them. Since the relationship between parameters controlling heuristic components and application performance measures

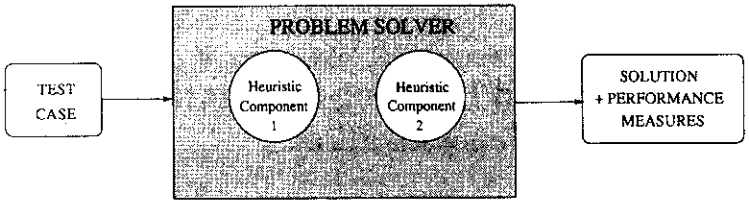

Figure 1: Solving a test case by a problem solver.

is generally very complex, the performance of heuristics is assessed by testing them on *test cases*.

Fig. 1 shows the application of a problem solver to find a solution using a given *test case* (problem instance). Note that in many applications, the number of test cases can be infinitely large.

For instance, consider TimberWolf (version 6) [2,3], a problem solver 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, various cost functions, and several numerical parameters. A test case in this application is a set of VLSI circuit components to be mapped to a physical layout, where a layout is a solution in this case. The placement of these components (cells and wires) must meet certain timing requirements and the laws of physics. The quality of a solution is in terms of the chip area of the final layout, and the cost of finding such a solution is in terms of execution time to find the layout.

1.2 *Generalization*

In this chapter, we study *statistical generalization* for determining the performance of a given set of *heuristic methods* (HMs) found in a heuristics-design process over a problem domain. Generalization usually refers to the process of making or choosing an element that is more general than other elements. In the context of heuristics design, *statistical generalization* is the process of finding or estimating the performance of a given set of HMs over a 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 in the process.

Figure 2 shows the concept of generalization. Suppose a subset of test cases in Subdomain 1 are used in learning, and the performance of n heuristics are evaluated statistically on these test cases. *Statistical generalization* entails

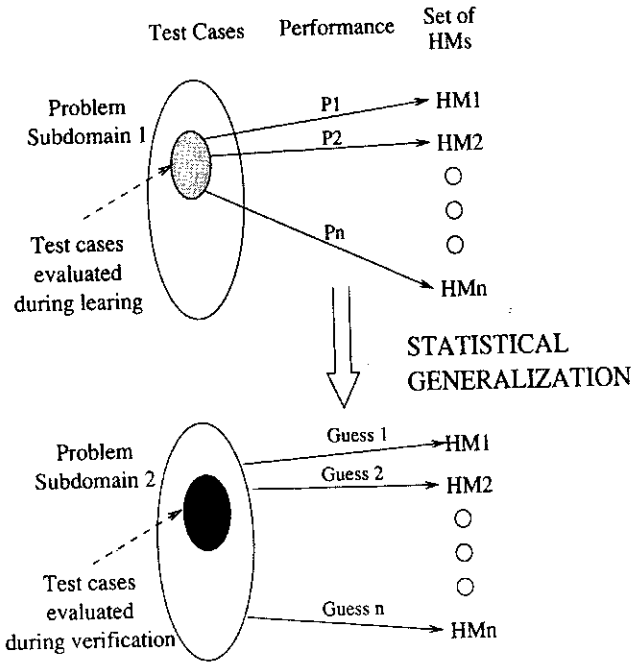

Figure 2: The concept of statistical generalization.

the extension and verification of the performance of these heuristics on unseen test cases and subdomains (say Subdomain 2 in Figure 2).

The primary objective of the generalization process is to find an HM that performs the "best" over the entire problem domain among the given set of HMs. 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 "outperforms" this existing HM over the entire problem domain.

The generalization process is simplified when the performance of each HM shares some common statistical characteristics, such as the *independent and identically distributed (IID)* property, for the entire problem domain. In this case, the performance of an HM over a subset of test cases can be statistically generalized to represent the performance of the HM over the entire problem domain. For example, the sample-mean performance from a subset of test cases can be used to predict the population mean over the entire problem domain.

This IID condition may not be true in complex real-world applications. In

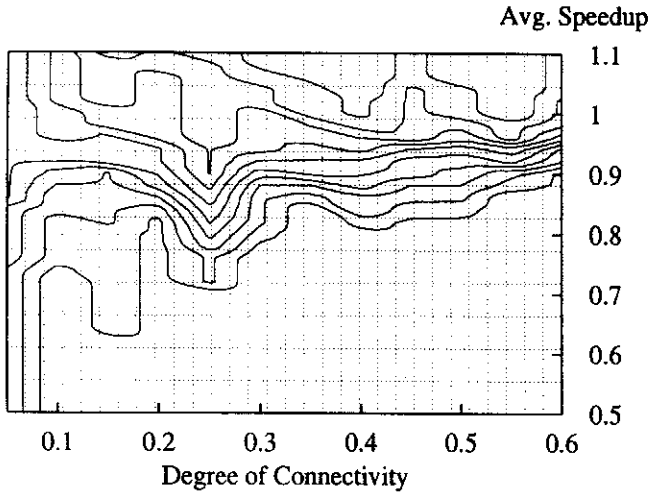

Figure 3: Contour plot showing the distribution of normalized performance values (speedups) of one HM on 15 test cases for solving the vertex-cover problem.

these applications, there may be many regions in the problem domain, each with different performance behavior. To address this issue, we propose to divide the problem domain into smaller subsets called *subdomains* in such a way that the performance of each HM within each subdomain is IID.

Example 1. Consider decomposition heuristics used in a branch-and-bound search to solve vertex cover problems. In this application, 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. 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.

Figure 3 shows the distribution of speedups of an HM with respect to the baseline HM for graphs of various degrees of connectivity. It clearly illustrates that the distributions of speedups are not IID across graphs of different degrees of connectivity, and that speedups of this HM cannot be averaged across all the graphs. ■

The generalization of HMs across multiple subdomains is more difficult because the performance of these HMs may have different statistical characteristics in different subdomains and cannot be combined or compared. Performance evaluation in one subdomain must, therefore, be dealt with separately and independently from other subdomains. Note that generalization only requires finding one HM that performs better than another across all subdomains. It is not necessary that the performance of the HM found be of the same statistical distribution across all subdomains.

Example 2. In testing two heuristic methods HM_1 and HM_2 on two subdomains 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. ■

In the most general case, there are more subdomains than what can be evaluated experimentally. Since computational resources are limited, it will not be possible to evaluate all these subdomains. Hence, generalization can only be made in a weak sense in which related subdomains are clustered together by some other methods, and generalization is only defined on clusters of subdomains.

Under multiple subdomains, our approach in generalization is to first evaluate and compare the performance of HMs in each subdomain. Next, we evaluate the performance of each HM over the entire problem domain and conclude whether the performance of the HM can be generalized. Finally, we pick the HM that works best across all the subdomains.

In the next section, 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 and our learning system *TEACHER*. Finally, we present some experimental results based on our statistical generalization procedure for TimberWolf.

2. Previous Work

Generalization has not been empl asized in most existing genetics-based machine learning systems [4,5,6]. In general, these systems implicitly assume that performance values of HMs share some 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 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 not evaluated fully during learning. In Section 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 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 4.

Note that in this chapter 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. The latter is the type of generalization studied in artificial intelligence [7], which requires more domain knowledge specific to a particular application and problem solver.

3. Performance Evaluation Within One Subdomain

In this section, we first present the performance evaluation of two HMs in one subdomain [8,9,10]. We then explore potential problems in extending this strategy to cases with more than two HMs. Finally, we present a strategy for general performance evaluation within a single subdomain.

3.1 Strategies for evaluating two HMs

In this subsection, we review the strategy for evaluating the performance of two HMs in one subdomain [8,9,10]. This strategy has been developed under the following assumptions:

- The normalized performance values of an HM in a subdomain are IID.
- The average performance of each HM is the objective metric to optimize.

The overall strategy can be divided into three steps.

(1) Symmetric Normalization. To reduce the difference in magnitude of performance values across test cases, it is necessary to normalize the performance values of each test case in order to obtain relative performance measures. In this process, one of the HMs is used as the baseline.

One issue in normalization is the compression of certain range of performance values. Consider the *improvement ratio* that divides the performance of the baseline HM by the performance of the target HM. Here, performance improvements are in the range between 1 and infinity, whereas degradations are in the range between 0 and 1. Hence, when an average improvement ratio is computed, performance improvements and degradations are weighted differently. One immediate consequence is that the conclusion of which HM is better may depend on the baseline HM used.

Example 3. To illustrate such an anomaly, consider the two HMs, HM_1 and HM_2, discussed in Example 2. It is easy to show that HM_1 has a higher average improvement ratio than HM_2 when HM_1 is used as the baseline. On the other hand, HM_2 has a higher average improvement ratio than HM_1 when HM_2 is used as the baseline. This is an obvious contradiction. ■

To avoid anomalies in inconsistent ordering due to the choice of the baseline HM, we should avoid normalization methods that emphasize differently in different ranges of the normalized performance values. One normalization method we have developed earlier [9] is the *symmetric improvement ratio*:

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

where S^+ is the original improvement ratio. The symmetric improvement ratio has the property that improvements are in the range between 0 and infinity, and degradations are in the range between 0 and negative infinity. Further, for two HMs, when we reverse the role of the baseline HM, their symmetric improvement ratios only change in sign. Hence, symmetric improvement ratios avoid the anomaly when using the original improvement ratios.

(2) Multi-objective Learning. When there are multiple performance measures corresponding to multiple objectives, we constrain the average performance of all but one objective measures. This modified goal helps 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) Probability of Win. We evaluate the performance of an HM in each of a set of subdomains. Due to the IID property in each subdomain, the average performance of an HM in a subdomain can be estimated by testing it on a

subset of test cases. We use the sample-mean values as statistical estimations of the population-mean values in deciding whether an HM satisfies or violates a performance constraint or whether it performs better than a baseline HM.

To address uncertainties in using estimated sample means, we have studied a concept called *probability of win*[9], P_{win}, that compares two sample averages and computes the probability that one sample average is larger than another. This is similar to hypothesis testing in which we take random samples to test whether a property of a population is likely to be true or false [11]. Obviously, it may be difficult to test a hypothesis fully by testing the entire population of test cases or by testing only a single random sample.

There are four steps in general hypothesis testing. (a) Specify a significance level α. (b) Specify the testing hypotheses that include both null hypothesis H_0 and alternative hypothesis H_1. (c) Find the corresponding acceptance region using lookup tables. (d) Make decision on the sample value. If the sample falls in the acceptance region, then accept hypothesis H_0 and reject H_1; otherwise, reject H_0 and accept H_1.

The probability of win measures statistically how much better (or worse) the sample mean of one HM is as compared to that of another. It resembles the significance level in general hypothesis testing, but there are two major differences. First, only one hypothesis H: $\mu_1 > \mu_2$ is specified, without the alternative hypothesis. Further, in contrast to hypothesis testing, acceptance confidence is not given in advance but is evaluated based on sample values.

One advantage of P_{win} is that it is between zero and one and is independent of the actual performance difference across subdomains. Hence, it can be used to compare HMs in a uniform way across subdomains.

Consider the performance of HM_i in subdomain j. (For convenience of formulation, subscript j is ignored in the following discussion.) Let μ_i and σ_i be the true mean and true standard deviation of the symmetric improvement ratio defined in (1) with respect to the baseline HM. When n_i samples are taken, we can calculate the sample mean $\bar{\mu}_i$ and sample standard deviation $\bar{\sigma}_i$. By Central Limit Theorem,

$$Pr(\bar{\mu}_i \mid \mu_i, \sigma_i, n_i) \approx \mathcal{N}\left(\mu_i, \frac{\sigma_i^2}{n_i}\right)$$

where, $\mathcal{N}$ is the normal distribution function with mean μ_i and standard deviation $\sqrt{\frac{\sigma_i^2}{n_i}}$. Let t be

$$t = \frac{\bar{\mu}_i - \mu_i}{\bar{\sigma}_i / n_i}$$

Table 1: Examples illustrating how P_{win} changes with increases in number of samples

HM_i	μ_i	σ_i	P_{win} for # of samples		
			5	10	30
HM_1	0.336	1.231	0.652	0.725	0.849
HM_2	-0.129	0.222	0.202	0.097	0.012
HM_3	0.514	0.456	0.940	0.991	1.000

where t has a Student's t-distribution with $n_i - 1$ degrees of freedom when the number of samples is less than 30 and the variance is unknown. The probability that this HM is better than the baseline with mean value 0 is

$$Pr(H \text{ is true}) = Pr\left(t \in \left(-\infty, \frac{\bar{\mu}_i}{\bar{\sigma}_i/\sqrt{n}}\right)\right) \quad (2)$$

$$= \int_{-\infty}^{\frac{\bar{\mu}_i}{\bar{\sigma}_i/\sqrt{n}}} p(t \text{ is } t\text{-distributed})\, dt \quad (3)$$

where the acceptance region of this hypothesis is $\left(-\infty, \frac{\bar{\mu}_i}{\bar{\sigma}_i/\sqrt{n}}\right)$. Note that the right bound of the acceptance region is a random variable that depends on both the sample mean and sample variance.

Example 4. Table 1 illustrates the P_{win} for three HMs. We see that P_{win} of HM_1 increases towards one when the number of samples increases. (HM_1 is better than the baseline.) In contrast, P_{win} of HM_2 reduces to zero when the number of samples is increased. (HM_2 is worse than the baseline.) Last, P_{win} of HM_3 reaches the maximum value 1.0, which means HM_3 is definitely better than the baseline. (HM_3 is better than the baseline but with larger mean and smaller variance as compared to HM_1.) ■

Note that P_{win} considers both the mean and variance. Hence, when P_{win} of an HM is close to 0.5, it is not clear whether the HM is better than or worse than the baseline.

We now show P_{win} of HM_i with respect to the average performance of baseline HM_0. For performance measure J_m in subdomain j, assume that the symmetric improvement ratio of HM_i has sample mean $\hat{\mu}_{i,j,m}$, sample variance

$\hat{\sigma}^2_{i,j,m}$, and $n_{i,j}$ test cases. P_{win} is defined as follows.

$$P_{win}(i,j,m) = F_t\left(n_{i,j}-1, \frac{\hat{\mu}_{i,j,m}}{\sqrt{\hat{\sigma}^2_{i,j,m}/n_{i,j}}}\right) \quad (4)$$

where $F_t(\nu, x)$ is the cumulative distribution function of Student's t-distribution with ν degrees of freedom, and $P_{win}(i,j,m)$ is the probability that the true performance (population mean) of HM_i in subdomain j for performance measure J_m is better than that of baseline HM_0. When $n_j \to \infty$, we have

$$P_{win}(i,j,m) \approx \Phi\left(\frac{\hat{\mu}_{i,j,m}}{\sqrt{\hat{\sigma}^2_{0,j,m}/n_{i,j}}}\right) \quad (5)$$

where Φ is the standard cumulative normal distribution function[12]. J_0 is used to denote the single unconstrained optimization measure.

We define $P_{ok}(i,j,m)$ as the probability that HM_i satisfies a given set of constraints (where constraint J_m, $m = 1, \ldots, k$, is θ_m) in subdomain j:

$$P_{ok}(i,j,m) \approx \min_m P(\mu_{i,j,m} \geq \theta_m) \quad (6)$$

$$\text{where} \quad P(\mu_{i,j,m} \geq \theta_m) = F_t\left(n_{i,j}-1, \frac{\hat{\mu}_{i,j,m}-\theta_m}{\sqrt{\hat{\sigma}^2_{i,j,m}/n_{i,j}}}\right) \quad (7)$$

P_{ok} measures the P_{win} of the worst-violated constraint.

To allow uncertainty in sample means, an HM is considered in violation of a given set of performance constraints when P_{ok} is less than $0.5 + \Delta$. The value of Δ can be controlled based on the HMs desired. For instance, during learning, when HMs that loosely satisfy the performance constraints are needed to compose new HMs, we can allow Δ to be as low as -0.25. This prevents eliminating many potentially good HMs that can help generate better HMs. On the other hand, when we try to generalize HMs learned to one that definitely performs better than the baseline HM, it is necessary to set $\Delta > 0$. An HM whose $P_{ok} > 0.5+\Delta$ with a large Δ has a high degree of certainty that it satisfy the corresponding performance constraint. Care must, however, be taken in setting Δ because it is possible to eliminate an HM that is actually better than the baseline HM when Δ is too high.

P_{win} is a certainty measure that can be used to order HMs based on the unconstrained optimization objective J_0 and baseline HM_0. In this case, HM_i is considered to have better normalized performance than HM_0 when $P_{win}(i,j,0) > 0.5 + \Delta$ for some $\Delta > 0$.

Table 2: Summary of raw performance values for four HMs in performance benchmarking of computers. (An HM represents a computer tested, and a test case is a benchmark program.)

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

3.2 *Strategies for evaluating more than two HMs*

There are two possible approaches to extend the above evaluation strategies for two HMs to the general case with more than two HMs. First, we can perform pair-wise comparison for every possible pair of HMs. This does not require a baseline HM as one of the HMs can be treated as the baseline HM in each comparison. Second, we can select a baseline HM and compare all HMs with respect to the baseline. In this subsection, we present potential problems that can arise with these approaches.

Performance anomalies in pair-wise evaluations

In this approach, each pair of HMs are compared using the strategy described in Section 3.1. Although the ordering between each pair can be determined without anomalies, the orderings of all the HMs are not transitive. In other words, when HM_1 is better than HM_2 and HM_2 is better than HM_3, there is no guarantee that HM_1 is better than HM_3. Consequently, there can be cycles in ordering with $HM_1 \to HM_2 \to HM_3 \to HM_1$, where $HM_a \to HM_b$ denotes that HM_a is better than HM_b in a pair-wise comparison. This anomaly is demonstrated in the following example.

Example 5. Table 2 shows the performance values of four HMs evaluated on three test cases in the performance benchmarking of computers. In this example, an HM represents a computer system, and a test case represents a benchmark program. Each performance value represents the time to execute a benchmark program on a computer system.

Using symmetric improvements defined in (1), we compute the average normalized performance of each HM using different HMs as baselines. Table 3 shows these average normalized performance values.

From this table, we observe that $HM_{75} \to HM_{86}$, $HM_{99} \to HM_{75}$, and $HM_{86} \to HM_{99}$ where each of these orders is based their P_{win} and aver-

Table 3: Summary of the average normalized symmetric improvements for four HMs with different HMs as the baseline.

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	–

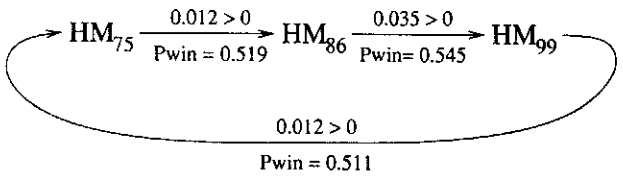

Figure 4: Example of a cycle in pair-wise ordering of HMs (based on the data in Table 3).

age symmetric improvement ratios. These lead to a cycle in which $HM_{75} \rightarrow HM_{86} \rightarrow HM_{99} \rightarrow HM_{75}$ as shown in Figure 4. Note that there is a high degree of uncertainty for these ordered pairs as their P_{win}'s are close to 0.5. ■

Due to possible cycles in pair-wise orderings of HMs when each order is evaluated by the average symmetric improvement ratio and P_{win}, it is not always possible to determine the best HM and the total ordering of HMs over a subdomain for a given set of HMs.

Anomalies when the baseline HM is changed

A different approach to evaluate multiple HMs is to normalize the performance of each HM with respect to a single 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. This anomaly is illustrated in the following example.

Example 6. Based on the data in Example 5, we show in Figure 5 the orderings of the four HMs when a different HM is used as the baseline each time.

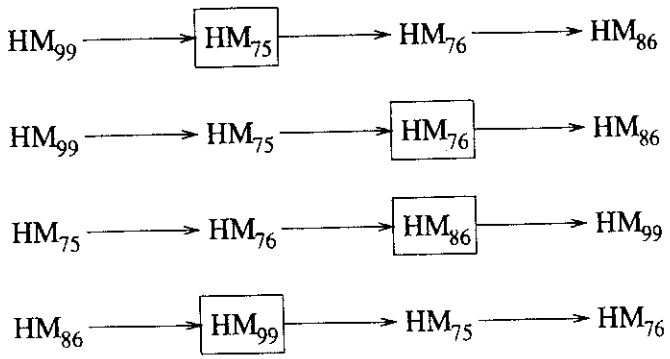

Figure 5: Orderings of HMs based on average normalized performance, each with a different HM as the baseline HM (based on the data in Table 3). The boxed HM is the baseline HM.

We observe that the ordering with HM_{75} as the baseline HM is different from the ordering when HM_{86} is the baseline. In fact, there are three different orderings, depending on the choice of the baseline HM. It is obvious that this approach cannot determine the best HM among the four HMs. ■

Normalization methods without anomalies

We illustrate in this section two normalization methods that do not have the anomalies in performance ordering presented above.

Given the performance values of HM_i, $i = 1, \ldots m$, on a set of test cases j, $j = 1, \ldots n$, the geometric mean of HM_i with respect to the baseline HM_0 is defined as follows.

$$\mu_{i,geom|HM_0} = \sqrt[n]{\frac{\mu_{i,1}}{\mu_{0,1}} \frac{\mu_{i,2}}{\mu_{0,2}} \cdots \frac{\mu_{i,n}}{\mu_{0,n}}} \tag{8}$$

After taking the logarithm of both sides, we obtain:

$$log(\mu_{i,geom|HM_0}) = \frac{1}{n} \sum_{j=1}^{n} [log(\mu_{i,j}) - log(\mu_{0,j})] \tag{9}$$

The advantage of (9) is that a change in the baseline HM does not affect the ordering of HMs found under the original baseline HM. This can be derived

easily as follows.

$$log(\mu_{i,geom|HM_{0'}}) = log(\mu_{i,geom|HM_0}) + \frac{1}{n}\sum_{j=1}^{n}[log(\mu_{0,j}) - log(\mu_{0',j})] \quad (10)$$

when $HM_{0'}$ is used as the baseline instead. Since the terms in the summations in (10) are constants that depend only on the baseline HMs, the ordering of the set of HMs remain unchanged when the baseline HM is switched.

Another example of a normalization function without anomalies in performance ordering is as follows.

$$\mu_{i,raw|HM_0} = \frac{1}{n}\sum_{j=1}^{n}[\mu_{i,j} - \mu_{0,j}] \quad (11)$$

When the baseline HM is switched from HM_0 to $HM_{0'}$, we have

$$\mu_{i,raw|HM_{0'}} = \mu_{i,raw|HM_0} + \frac{1}{n}\sum_{j=1}^{n}[\mu_{0,j} - \mu_{0',j}] \quad (12)$$

Example 7. Using the data in Table 2 and applying (8), we get the following ordering of HMs: $HM_{99} \rightarrow HM_{76} \rightarrow HM_{75} \rightarrow HM_{86}$. On the other hand, applying (11) results in a totally different ordering: $HM_{86} \rightarrow HM_{99} \rightarrow HM_{76} \rightarrow HM_{75}$. ■

Although the two methods we have presented avoid anomalies in the ordering of HMs when the baseline HM is changed, they are not always desirable. Each of these methods places different emphasis on different performance values, resulting in different orderings of HMs. For instance, the geometric mean places less emphasis on large performance values because it takes the logarithms of performance values. In contrast, the method in (11) places more emphasis on raw performance values, resulting in an ordering that may be biased by a few large performance values. We are currently studying other normalization methods that can emphasize or deemphasize certain performance ranges while avoiding anomalies in the ordering of HMs.

3.3 Proposed strategy based on a single baseline HM

In the previous subsection, we have presented anomalies in the ordering of HMs and some methods to cope with these anomalies. A method that avoids anomalies may not always be desirable because it may emphasize certain parts

of the performance range in computing an average measure. We conclude that it may not always be possible to select the "best" HM from a given set of HMs, and our goal in learning and generalization may have to be relaxed to find an HM that is "better" than the best existing HM. This objective can be accomplished by using the best existing HM as the baseline, which can be used consistently throughout learning, verification, and generalization.

There are three advantages when using a single baseline HM.

First, using a common baseline HM allows us to compare all HMs consistently with respect to it and to derive confidence levels P_{win} whether an HM is better than the baseline. This is illustrated in the following example.

Example 8. We continue to use the data in Example 5 as our running example. We assume that HM_{76} is the baseline HM to be improved upon. From Table 3 and Figure 5, we observe that both HM_{75} and HM_{99} have better average normalized performance values than HM_{76} (and $P_{win} > 0.5$ as well).

If $\Delta = 0$, then both HMs would be acceptable based on their P_{win}'s. If our comparison is based only on each HM's sample-mean value, then HM_{99} will be considered a better HM. On the other hand, if the selection is based on P_{win}, then HM_{75} is a better choice.

If $0.4 > \Delta > 0.05$, then only HM_{75} (with P_{win} of 0.981) can be considered better than HM_{76} because P_{win} of HM_{99} (0.519) is too low. In short, HM_{75} is the most robust choice among the given set of HMs that is better than HM_{76}. ■

Second, using a common baseline HM allows us to use it consistently across all subdomains. Consequently, anomalies in orderings due to different baseline HMs in different subdomains cannot happen.

Third, using a common baseline HM allows us to find HMs that are better than the baseline HM. The exact ordering of these HMs may not be critical as our goal in learning and generalization is to distinguish good HMs that perform consistently better than the baseline HM across all subdomains from poor HMs that may not perform well all the time.

3.4 Proposed strategy without a baseline HM

In some applications, there may not be a single baseline HM for performance to be compared. For example, the baseline HM may change under different conditions or may improve as better HMs are found. In this case, it is not feasible to pick a single HM in an ad hoc fashion as the baseline HM, as the

set of HMs that are better than an incumbent HM with respect to one baseline HM may not be the same as the set when a different baseline HM is used.

One possibility in this case is to use the raw unnormalized performance values in computing the average performance measure. However, this is not acceptable in many cases as the distribution of performance values across different test cases may have large variations and unequal distributions, and averaging these non-IID performance values is not valid statistically.

Another method we have developed is to normalize the performance values of HMs on each test case with respect to a test case-specific constant that is invariant as more HMs are evaluated. One possibility here is to use the *median* performance value of all HMs on each test case as the baseline for normalization. Unlike using a baseline HM that may induce a different ordering when the baseline is changed, the median performance is invariant with respect to HMs and test cases in a subdomain. P_{win} can then be computed based on a pseudo HM with median performance in each test case.

Using this normalization method, the performance distributions of all test cases will center around zero. Further tests can then be made to determine whether these distributions are IID.

A potential problem with this approach is the unavailability of the true median performance value of HMs for each test case. Hence, the sample median may have to be used instead. Unfortunately, estimated sample medians are inaccurate during learning because HMs may not be tested adequately, and sample medians are sensitive to the HMs generated. Solutions to this issue are still open at this time.

4. Generalization Across Multiple Subdomains

In this section, we study the problem of statistical generalization over a problem domain with multiple subdomains. Because different subdomains have different statistical behavior, performance from different subdomains must be treated independently and cannot be combined.

There are two assumptions on the strategies presented in this subsection.

- We assume that the set of subdomains used in the design process are *representatives* of all the subdomains in the application. These subdomains behave in a statistically similar fashion to subdomains used in learning and generalization.

- We assume that the relative importance of one subdomain as compared to another is unknown, and that the performance of HMs in subdomains may be dependent. Under these assumptions, we cannot aggregate performance values of HMs across subdomains. Our strategy is to select HMs so that their worst-case performance across all subdomains is better than a minimum level.

The objective of generalization here is to select an HM that is better than the incumbent HM over a problem domain. When there are multiple such HMs, our procedure should attempt to maximize the likelihood of selecting the best HM among the given set.

When there is a baseline HM, we apply the strategy in Section 3.3 to normalize the performance of an HM in a subdomain with respect to the baseline HM. We consider an HM to be better than the baseline when its P_{win} is greater than $0.5 + \Delta$ for that subdomain, while satisfying all other performance constraints ($P_{ok} > 0.5 + \Delta$). Hence, an HM is better than the baseline for the entire domain when it is better than the baseline HM in all selected subdomains. The following are four possible outcomes when comparing the performance of an HM to the baseline HM.

(a) *One HM is better than the baseline in all subdomains.* This HM can then replace the baseline HM in the target problem solver.

(b) *Multiple HMs are better than the baseline in all subdomains.* Here, we should select one HM that maximizes the likelihood of being better than the baseline over the entire domain. This likelihood (or degree of confidence) can be adjusted by increasing Δ in computing P_{ok}. Using a large Δ is equivalent to placing tighter constraints on each performance measure, hence eliminating some potential HMs that are found to be better than the baseline under looser constraints.

(c) *No HM is better than the baseline in all subdomains.* This means that we cannot find $\Delta \geq 0$ such that $P_{ok} > 0.5 + \Delta$. Since no HM is superior to the incumbent, the target problem solver should continue to use the baseline HM.

Alternatively, it is possible to find HMs such that $P_{ok} > 0.5 + \Delta$ where $\Delta < 0$. Such HMs have less certainty in performing better than the baseline across all the subdomains. However, since P_{ok} is based on the worst-case P_{win} across all subdomains, HMs selected this way may still perform better the baseline in some subdomains. Such HMs should be considered as alternatives to the baseline HM.

(d) *The incumbent HM violates some performance constraints.* Since the baseline HM cannot satisfy all the performance constraints, it is acceptable to have HMs that violate some of the performance constraints as well. This results in HMs that give different performance trade-offs. Of course, the HM selected is expected to perform better than the baseline with respect to the

unconstrained performance measure. We do not use P_{win} as the criterion in this case since HM selected may not always perform better than the baseline.

When there is no baseline HM, performance generalization across subdomains is more difficult. We cannot use raw performance values for performance comparison across subdomains as they may depend on the size of test cases used and application characteristics. Our approach in this case is similar to that in a single subdomain. We first normalize the performance of HMs for a specific test case with respect to the median performance of all HMs for this test case. Probabilities of mean with respect to a pseudo HM with the median performance can then be calculated. Generalization across subdomains can be carried out similar to that of applications with baseline HMs.

5. TEACHER - A Genetics-Based System for Learning Heuristics in One Subdomain

In this section, we present the architecture of *TEACHER* (an acronym for Techniques for the Automated Creation of Heuristics), a prototype learning system we have developed to learn improved HMs under resource constraints for a single subdomain [13,14]. Figure 6 shows the architecture of *TEACHER*. Based on the genetics-based machine-learning paradigm, this population-based learning system has five main components:

(a) *Resource Scheduler* that determines the best way to use the available computational resources,

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

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

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

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

In our system, 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 in heuristics generation, performance evaluation, and resource scheduling.

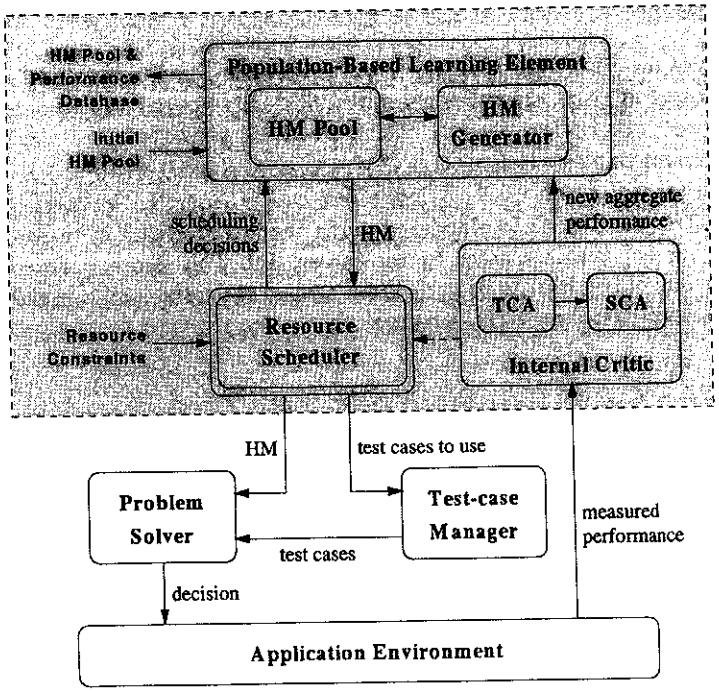

Figure 6: Architecture of population-based learning for one subdomain.

Problem Solver

This component is simply the target problem solver whose heuristics we want to improve. The performance of applying a 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 accepts (a) the specification of the HM to be used in problem solving, and (b) the test case to be solved. It also has a mechanism to return the measured performance of the problem-solving process as feedback to the learning system.

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.

In our current implementation, each HM is evaluated on a predefined sequence of user-specified test cases. When a test case is requested for testing a particular HM, the Test-Case Manager returns the first test case in the sequence that has not been evaluated by the chosen HM. This strategy allows performance data of two HMs to be normalized against each other and is useful when performance data have large variances.

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 are generated to replace existing ones. Several top active HMs are usually retained along with the new HMs while other HMs are removed from the active pool.

The Population-Based Learning Element in *TEACHER* generates new HMs using weak domain-independent operators, such as crossover, mutation, and hill-climbing. These are traditional operators used in genetic algorithms for generating new HMs [15,16]. The process for selecting existing HMs for reproduction is also the same as in traditional genetics-based machine learning.

More advanced generation methods that require additional domain knowledge are left for future study. They are currently not necessary because our application domains are knowledge lean.

Internal Critic

In general, the Internal Critic performs credit assignment [17] that apportions credit and blame on components of an HM using results obtained in testing. 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 feedback between the current and the past decisions. Methods for TCA depend on whether the state space is Markovian: non-Markovian representations often require more complex TCA procedures. The second stage is SCA that translates the (temporally local but structurally global) feedback associated with a decision point into modifications associated with various parameters of the decision process.

Since the knowledge-lean applications considered in our research does not have a world model that relates states, decisions, and feedback signals generated by the learning system or measured in the environment, credit assignment has a much weaker influence on performance improvement. Note that the lack of a world model for credit assignment is the main reason for maintaining competing HMs in our learning system.

In our current prototype, the Internal Critic normalizes the performance value of an HM on a test case against the performance value of the same test case evaluated by the baseline HM. It then updates the fitness value of the candidate HM. which is a heuristic measure to differentiate good HMs from poor ones. Recall that P_{win} is computed for an HM in each subdomain. However, P_{win}'s across subdomains cannot be averaged because they are generally not IID. Our approach is to check the P_{win} of an HM in each subdomain and places penalty on the HM in that subdomain whenever its P_{win} is worse than $0.5 + \Delta$, where $\Delta < 0$. On the other hand, if the HM performs better than the baseline HM in all subdomains, then its fitness value will be computed using symmetric speedups in all the subdomains. This is done mainly to differentiate one HM from another that are both better than the baseline; in this case, we cannot use P_{win} to differentiate these HMs as their P_{win}'s will all approach one.

In short, the equation for computing the fitness value of HM_i in subdomain j is:

$$fitness_i = \begin{cases} \dfrac{1}{1+\sum_{\forall j P_{win}<0.5+\Delta}(0.5+\Delta-P_{win})} & \text{If } P_{win} < 0.5+\Delta \text{ in at least one subdomain} \\ \dfrac{1}{1-\sum_j S^{*}{}_{jm+}} & \text{If } P_{win} > 0.5+\Delta \text{ in all subdomains} \end{cases} \quad (13)$$

Resource Scheduler

This schedules tests of HMs based on the available computational resources. It is critical when tests are computationally expensive. There are two problems in scheduling during each learning phase.

The *sample-allocation problem* involves the scheduling of tests of HMs in a generation, given a fixed number of tests in the generation and HMs to be tested. This problem is known in statistics as the (sequential) *allocation problem* [18,19] and the scheduler, the *local scheduler*.

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

These two problems, as well as the scheduling of tests under multiple performance objectives, are presented elsewhere [20,9,13,21,22,23].

6. Experimental Results

In this section, we present some results we have obtained in learning and generalizing new HMs for VLSI cell placement and routing. In this application, logic components of a circuit are to be placed and routed on a two-dimensional VLSI chip. Layout consists of three steps: gate assignment, placement, and routing. In this research, we apply learning to the placement problem alone. The objective of placement is to minimize the chip area while satisfying constraints such as wire-ability, while the design is within the maximum time delay and number of poly-silicon layers and does not have severe heat source concentration. This problem is NP-hard, and heuristics are generally used.

We use TimberWolf 6.0 as our problem solver. This is a software package based on simulated annealing (SA) to place and route various cells (transistors, resistors, capacitors, wires, etc.) on a piece of silicon [24]. Its operations can be divided into three stages: placement, global routing, and placement refinement. SA is used in TimberWolf to randomly search for possible placements. Although SA converges asymptotically in theory to the global optimum with probability one, the results generated in finite time are usually suboptimal. Hence, there are trade-offs between the size of a layout and the search time in obtaining the layout. One of the control parameters is called *fast_n* that controls the search speed of SA: the larger *fast_n* is, the shorter time SA will run, and, therefore, the worse the result will be.

TimberWolf has five major components: cost function, generation function, initial temperature, temperature decrement function, inner loop criterion, and stopping criterion. Many parameters in these components are tuned manually. To demonstrate the applicability of our learning system, we take the cost function and temperature scheduling parameters as two HMs to learn and to generalize.

The cost function is the objective of the SA search component in TimberWolf. Recall that the goal in placement is to minimize the chip area. However, the actual chip area can only be evaluated after global routing is done, based on a placement generated by the SA search. Hence, the chip area after SA has generated a placement is only estimated heuristically by a cost function in TimberWolf. This cost function is as follows:

$$cost_{baseline} = v1 + v2 * v3 \tag{14}$$

where, $v1$, $v2$ and $v3$ are horizontal wire length, vertical wire length, and

Table 4: Benchmark circuit specifications

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

relative weight between horizontal and vertical wire lengths, with $v3$ set to one. The baseline HM is, therefore, $v1 + v2$. In our experiments, we use the above three variables $v1$, $v2$, $v3$ to construct new cost functions.

Table 4 shows the specifications of six benchmark circuits used to evaluate the performance of HMs: *s298*, *s420*, *primary1*, *primary2*, *struct*, *fract*[25].

As the behavior of an HM can be quite different across different circuits, we treat each circuit as a subdomain by itself. We set *fast_n*, the speed of the annealing process, to 1, 5, and 10, respectively. We further used a fixed sequence of random seeds as a program control parameter. These are needed because whenever a new result is obtained in annealing, a random number is needed to decide whether to keep that result or not. Different random seeds will lead to randomness in the result of each test case.

In our simulations, we used circuits *s298*, *s420*, and *primary1* as our learning subdomains, and set *fast_n* to 10 in the learning process. The best cost function we found is very similar to that in (14).

$$cost_{learned} = v1 + v2 * 2 \tag{15}$$

We compare the performance of both HMs in Figure 7. We evaluate three performance measures based on symmetric improvement ratios, where improvements are measured by reductions in the area of the chip and the cost of obtaining the placement. The average area represents the average chip area after placement and global routing, based on ten runs of TimberWolf using ten random seeds; the minimum area is the minimum chip area achieved; and the average cost is the average execution time in CPU seconds. We see that most of the points are below the horizontal x axis, representing improvements in the area of the layout. However, half of the points are to the right of the y axis, representing increases in execution costs to find the better layouts.

In another set of learning experiments, we learned numerical temperature scheduling parameters in the temperature function. In simulated annealing,

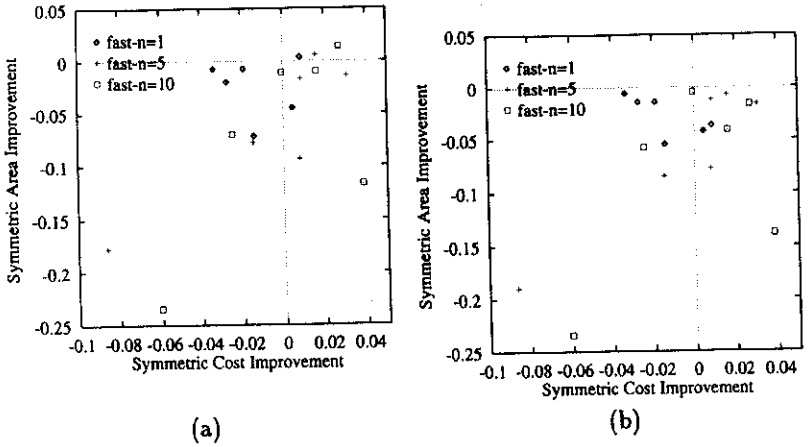

Figure 7: Area-performance of the new cost function learned with respect to the original cost function. (a) average cost/average area over 10 runs, (b) average cost/minimum area over 10 runs.

Table 5: Parameters in TimberWolf (version 6) found by our learning experiments. (Blank entries in the last two columns represent parameters that are not learned in the learning experiments.)

Param.	Range	Step	Meaning	Default	Expt. 1	Expt. 2
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	18.0	10.0416
P_6	50.0 - 99.0	1.0	intermediate temperature finishing point	81.0	71.0	63.6962
P_7	100.0 - 150.0	1.0	low temperature finishing point	125.0	135.0	125.5509
P_8	130.0 - 180.0	1.0	final iteration temperature	155.0	162.8682	147.9912
P_9	0.29 - 0.59	0.01	critical ratio that determines acceptance probability	0.44	0.3457	0.3325
P_{10}	0.01 - 0.12	0.01	temperature for controller turn off	0.06	0.05334	0.1124
P_{11}				125.0	124.8916	
P_{12}				0.025	0.03025	
P_{13}				0.25	0.2290	

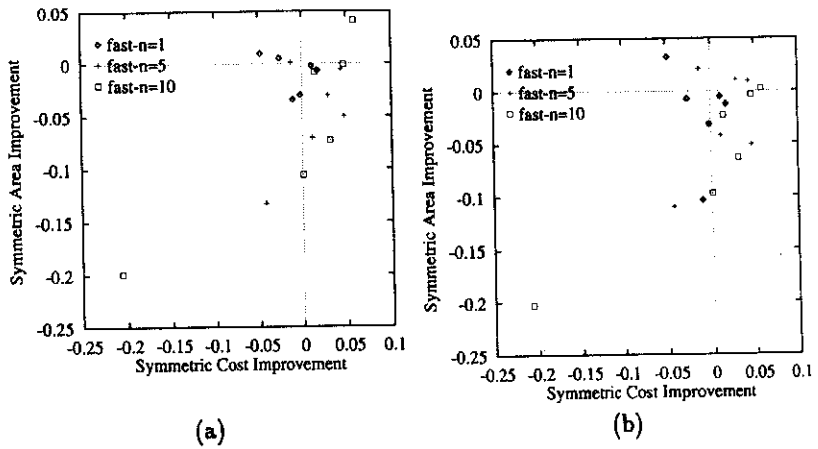

Figure 8: Area-performance of the new temperature parameters learned with respect to the original temperature parameters. (a) average cost/average area over 10 runs, (b) average cost/minimum area over 10 runs.

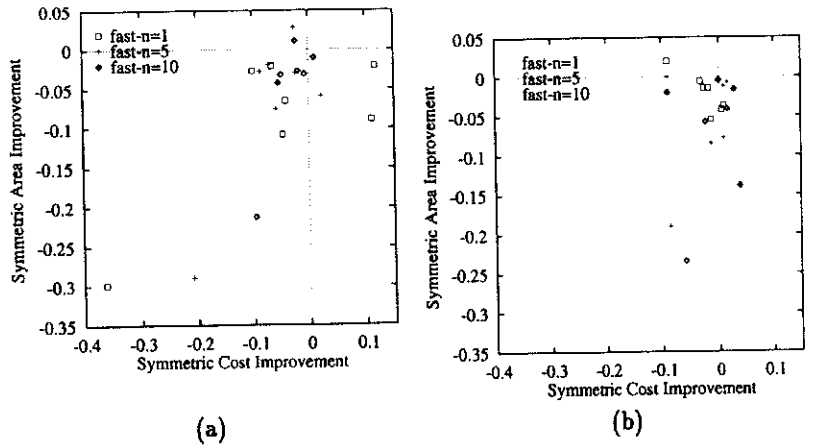

Figure 9: Area-performance of the new cost and temperature parameters learned with respect to the original cost and temperature parameters. (a) average cost/average area over 10 runs, (b) average cost/minimum area over 10 runs.

time is divided into four regions, with different cooling speeds. In the beginning, temperature drops rather fast, and slows down when time increases. We extracted nine parameters from the program, with default values shown in Table 5. The new parameters learned (based on the default cost function (14)) are shown in Table 5 under the column labeled "Expt. 1." Figure 8 shows the area-cost improvements with respect to the original temperature parameters. Although most of the points are below the x axis, many of them are to the right of the y axis. These represent better areas of layout at higher execution costs. These experiments show that tuning the temperature function alone may not lead to significant improvements.

In our last learning experiments, we learned a collection of cost-function and temperature scheduling parameters. The parameters learned are shown in Table 5 under the column labeled "Expt. 2." Note that in these experiments, P_2 is the same as v_3 in (14). Our learning experiments found a value of 0.2315 instead of 2 in (15). Figure 9 shows the area-cost improvements. With respect to the average area, most of the results show improved area and cost, while there are less improvements when the minimum area is evaluated based on ten runs of TimberWolf.

7. Conclusions

In this chapter, we have studied *statistical generalization* for determining the performance of a given set of *heuristic methods* (HMs) found in a heuristics-design process over a problem domain. The objective of statistical generalization is to determine the "best" HM from among a given set of HMs over test cases in a problem domain. Due to difficulties in finding an HM that is superior all the time, the objective in learning and in generalization is relaxed to finding a subset of HMs that are better than the incumbent HM.

One of the major problems in performance generalization is the dependence of performance information on the size of test cases in an application domain. Performance data obtained from test cases of different sizes do not belong to a common statistical distribution and cannot be aggregated. We address this problem by dividing test cases in a domain into subdomains and by normalizing performance data in a subdomain so that normalized data in a subdomain are independent and identically distributed. Performance data in a subdomain can then be aggregated statistically into averages.

We have proposed a statistical method for comparing the performance of HMs. Instead of using sample averages alone, which can be erroneous when sample standard deviations are large, we evaluate a measure called probability of win that depends on both the sample mean and the sample standard deviation. It measures the probability that the population mean performance of one HM is better than the population mean of another. It can also be considered as a normalized performance measure as it values is between 0 and 1.

We have shown anomalies in the ordering of HMs for some normalization methods when the baseline for normalization is changed, and two normalization methods that do not have anomalies. The best choice of a baseline HM is one whose performance is invariant when new HMs are generated in learning. One such HM is one whose performance on a test case is the median performance of all HMs evaluated on this test case.

In evaluating performance of an HM across subdomains, it is difficult to find an aggregate statistical measure as performance data across subdomains can be dependent. Moreover, we do not know the weight that should be placed on each subdomain in evaluating the aggregate measure. For this reason, we have proposed to place a common performance constraint across all subdomains and accept an HM if its performance satisfies the constraint across all the subdomains.

Finally, we have described the architecture of *TEACHER*, our prototype learning system, and experimental results in learning new HMs for TimberWolf. Our results show that automated learning and generalization will lead to improved HMs.

Acknowledgments

We like to acknowledge the support of National Science Foundation Grants MIP 92-18715 and 96-32316 and National Aeronautics and Space Administration Contract NAG 1-613.

References

1. A. Newell, J. C. Shaw, and H. A. Simon. Programming the Logic Theory Machine. In *Proc. 1957 Western Joint Computer Conf.*, pages 230–240. IRE, 1957.
2. C. Sechen. *VLSI Placement and Global Routing Using Simulated Annealing.* Kluwer Academic Publishers, Boston, MA, 1988.
3. C. Sechen and A. Sangiovanni-Vincentelli. The timberwolf placement and routing package. *J of Solid-State Circuits*, 20(2):510–522, 1985.
4. D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2/3):95–100, October 1988.
5. J. R. Koza. *Genetic Programming.* The MIT Press, Cambridge, MA, 1992.

6. C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization. In *Proc. of the Fifth Int'l Conf. on Genetic Algorithms*, pages 416–423, Morgan Kaufman, June 1993. Int'l Soc. for Genetic Algorithms.
7. C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Mach. Learn.*, 13(2/3):189–228, November/December 1993.
8. A. Ieumwananonthachai. *Automated Design of Knowledge-Lean Heuristics: Learning, Resource Scheduling, and Generalization.* Ph.D. Thesis, Dept. of Electrical and Computer Engineering, Univ. of Illinois, Urbana, IL, May 1996.
9. 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. on Knowledge and Data Engineering*, 7(5):763–785, October 1995.
10. A. Ieumwananonthachai and B. W. Wah. Statistical generalization of performance-related heuristics for knowledge-lean applications. *Int'l J. of Artificial Intelligence Tools*, 5(1 2):61–79, June 1996.
11. R. E. Walpole. *Introduction to Statistics.* Macmillan Publishing Co. Inc., New York, NY, 1982.
12. J. L. Devore. *Probability and Statistics for Engineering and the Sciences.* Brooks/Cole Pub. Co., Monterey, CA, 1982.
13. B. W. Wah. Population-based learning: A new method for learning from examples under resource constraints. *IEEE Trans. on Knowledge and Data Engineering*, 4(5):454–474, October 1992.
14. B. W. Wah and A. Ieumwananonthachai. Generalization of heuristics learned in genetics based learning with applications in circuit testing and computer aided design. In Xin Yao, editor, *Evolutionary Computation.* World Scientific Publishing Co. Pte. Ltd., 1996.
15. L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. In J. Carbonell, editor, *Machine Learning: Paradigm and Methods.* MIT press, 1990.
16. J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *Trans. on Systems, Man, and Cybernetics*, SMC-16(1):122–128, January 1986.
17. R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* Ph.D. thesis, Univ. of Massachusetts, Amherst, MA, February 1984.
18. R. E. Bechhofer. A single-sample multiple decision procedure for ranking means of normal populations with known variances. *Ann. Math. Statist.*, 25(1):16–39, March 1954.

19. Y. L. Tong and D. E. Wetzell. Allocation of observations for selecting the best normal population. In T. J. Santner and A. C. Tamhane, editors, *Design of Experiments: Ranking and Selection*, pages 213–224. Marcel Dekker, New York, NY, 1984.
20. A. Ieumwananonthachai, A. Aizawa, S. R. Schwartz, B. W. Wah, and J. C. Yan. Intelligent process mapping through systematic improvement of heuristics. *J. of Parallel and Distributed Computing*, 15:118–142, June 1992.
21. A. N. Aizawa and B. W. Wah. A sequential sampling procedure for genetic algorithms. *Computers and Mathematics with Applications*, 27(9/10):77–82, May 1994.
22. A. K. Aizawa and B. W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, 2(2):97–122, 1994.
23. A. Aizawa and B. W. Wah. Scheduling of genetic algorithms in a noisy environment. In *Proc. Int'l Conf. on Genetic Algorithms*, pages 48–55, Morgan Kaufman, July 1993. Int'l Soc. for Genetic Algorithms.
24. C. Sechen. *VLSI Placement and Global Routing Using Simulated Annealing.* Kluwer Academic Publishers, Boston, MA, 1988.
25. LayoutSynth92. Vlsi layout and synthesis benchmarks. *International Workshop on Layout Synthesis*, 1992.