Statistical Generalization: Theory and Applications *

Benjamin W. Wah, A. Ieumwananonthachai, Shu Yao, and Ting Yu

Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
1308 West Main Street, Urbana, IL 61801, USA
{wah, arthuri, yshu, yuting}@manip.crhc.uiuc.edu

Abstract

In this paper, we discuss a new approach to generalize heuristic methods (HMs) to new test cases of an application, and conditions under which such generalization is possible. Generalization is difficult when performance values of HMs are characterized by multiple statistical distributions across subsets of test cases of an application. We define a new measure called probability of win and propose three methods to evaluate it: interval analysis, maximum likelihood estimate, and Bayesian analysis. We show experimental results on new HMs found for blind equalization and branch-and-bound search.

1 Introduction

In this paper we present the theory and applications of statistical generalization, which involves methods to statistically assess the performance of heuristics and conditions under which one *heuristic method* (HM) performs better than another.

An application problem is generally specified by its problem variables and constraints under which solutions are sought. Given a *test case*, or an application problem instance, a *problem solver* is applied to find a solution to the test case. In general, solutions can be conditions to guarantee correct operations, or can be numerical values of *performance measures*. In this paper we are interested in the class of problem solvers with numerical performance measures. Such measure are generally inter-related and are statistical in nature. Examples of performance measures include quality of solutions and cost of getting them.

*Research supported by National Science Foundation Grant MIP 92-18715 and National Aeronautics and Space Administration Grant NAG 1-613.

Plenary address, Architecture Track, IEEE International Conference on Computer Design, October 1995.

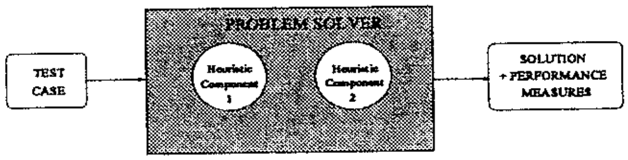


Figure 1: Solving a test case by a problem solver.

A problem solver generally has *heuristic components* or *heuristics* that were designed in an ad hoc fashion (see Figure 1). Newell, Shaw, and Simon defined heuristics as "A process that may solve a problem but offers no guarantees of doing so" [1]. Since the relationship between parameters controlling heuristic components and application performance measures is generally very complex, the performance of heuristics is assessed by testing them on test cases.

There are many ways to represent heuristics; examples include a collection of numeric parameters, symbolic formulae, rules, and procedures. The following are examples of applications and their heuristics.

- *Process mapping.* This involves mapping a set of communicating processes on a distributed-memory multi-computer system. The goals of the problem solver are to minimize the completion time of the processes mapped and the cost of finding such mappings. An example of a problem solver is Post-Game Analysis (PGA) [2], which is a simulation-based method to find the best mapping. Heuristics used in PGA include those for proposal generation, priority assessment, transformation generation, and feasibility test.

- *Load balancing in distributed systems.* The goals are to minimize the completion time of an incoming job while minimizing the overhead of collect-

1063-6404/95 $4.00 © 1995 IEEE

ing workload information. An example problem solver is CONDOR [3] with heuristics to evaluate workload, threshold to start process migration, and threshold to accept incoming processes.

- *Blind equalization.* One variation of the 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 [4]. The heuristics involved are the initial weights of the descent algorithm and the cost function defined in the weight space for the descent algorithm.

- *Finding a minimum-cost tour in a directed graph.* An example problem solver is a branch-and-bound algorithm. This has heuristics to select a partially expanded node for expansion, decide how to decompose the partially expanded node selected into descendants, prune unpromising nodes from further expansion, and terminate the search. The goal is to find a minimum-cost tour while minimizing the cost to find it.

Methods to generate new heuristics can be classified as knowledge rich and knowledge lean. In knowledge-rich applications, domain knowledge is used to decide what new heuristics to generate, based on performance measured on past heuristics. In contrast, in knowledge-lean applications, syntactic and application-independent operators, such as cross-over and mutation, are used to generate new heuristics.

In applications studied in this project, we have used operators that have domain-specific and domain-independent components. For instance, in designing decomposition heuristics for a branch-and-bound search to solve a traveling salesman problem, one needs to first identify domain-specific attributes in constructing the decomposition function. Examples of these attributes include length of the current partial tour, minimum length to complete the current partial tour, and number of neighboring cities not visited. Based on these domain-specific attributes, possible decomposition functions can be constructed by domain-independent operators, such as cross-over and mutation, to form a formula to be evaluated at run time.

Generalization is a process of finding good HMs that perform well on test cases not evaluated before. It is an important problem because many algorithms used in engineering applications are heuristic in nature, and, due to limited time, only a small subset of test cases can be evaluated during their design.

Generalization is a problem that has been studied extensively in artificial intelligence [5, 6, 7]. Past approaches generally focus on syntactic and semantic aspects of generalization that extends heuristics learned to new situations. Statistical approaches to generalization are usually restricted to using statistical measures in decision procedures. Our approach here is distinguished from previous approaches in identifying good heuristics that may have different but unknown statistical performance distributions across different subsets of test cases.

In the next section, we examine alternative goals in generalization and define domains and subdomains. We also study issues on multi-objective trade-offs and normalization. In Section 3, we describe parallel learning, define the concept of probability of win, and show three statistical methods to compute it. In Section 4, we present some preliminary results of our method.

2 Design Goals and Performance Issues

2.1 Domains and Subdomains

Given an application problem consisting of a collection of test cases, the first task in learning and generalization is to classify the test cases into *domains* such that a unique HM can be designed for each. This classification step is domain specific and should be carried out by experts in the area.

For instance, in designing a measure to evaluate workload in a multi-computer system, experts in parallel processing will indicate that workload measure for a network of workstations is different from workload measure of a distributed-memory massively parallel processing system. Consequently, we can consider these two architectures as two different domains.

Given a domain, a possible goal in learning and generalization is to find a new HM that always performs better than a baseline HM for all test cases, where the *baseline HM* (or HM_{base}) is the best HM known so far. This goal is difficult to achieve when there are infinitely many test cases in the domain.

Another possible goal is to find a new HM that performs better than HM_{base} on the average across all test cases in the domain. This may not always be possible as different heuristics may have different distributions of performance over different subsets of test cases. This phenomenon is illustrated in Table 1 that shows the average and maximum fault coverages of two HMs used in a test-pattern generator to test sequential circuits. The data indicate that we cannot average the fault coverages across the two circuits as the performance values of HM_{101} across the two circuits do not belong to one distribution.

Table 1: Maximum and average fault coverages of two HMs used in a test-pattern generator with different random seeds.

Circuit	HM	Maximum FC	Average FC
S444	101	60.3	28.5
	535	86.3	84.8
S1196	101	94.9	94.2
	535	93.6	93.1

Table 2: Kolmogorov-Smirnov test, with 0.01 significance interval, of nodes expanded in a branch-and-bound search using a specific decomposition HM to solve a vertex cover problem.

Degree of Connectivity	Graph Size	Number of Samples	D_n	D_n^α
0.1	30	30	0.17	0.19
	40	30	0.11	0.19
	30 + 40	60	0.15	0.13
0.1	30	30	0.17	0.19
0.2		30	0.19	0.19
0.1 + 0.2		60	0.18	0.13

A third possible goal is to find a new HM that performs better than HM_{base} on the average for all subdomains of test cases. In this case, a *subdomain* is a partitioning of the domain of test cases such that performance values of one HM in a subdomain are independent and identically distributed (iid). Under this condition, it is meaningful to compute the average performance across test cases in a subdomain. The advantage of classifying test cases into subdomains is that we only need to evaluate a HM on a small number of test cases in a subdomain in order to assess its performance across all test cases in the subdomain.

Table 2 shows the results of normality tests of a HM used in solving a vertex-cover problem.[1] We have applied the Kolmogorov-Smirnov method for normality test with unknown mean and variance [8]. Tests with significance level of 0.01 pass marginally when graphs of sizes 30 and 40 are grouped into one subdomain but fail when grouping graphs of 0.1 and 0.2 degrees of connectivity. Hence, we can save time by testing the HM on smaller graphs (as graphs of sizes 30 and 40 belong to the same subdomain), but graphs with 0.1 and 0.2 average degrees of connectivity must both be tested as they belong to different subdomains.

Note that the Kolmogorov-Smirnov method only tests whether two random variables belong to one common distribution. Tests for independence may also need to be carried out [9]. Further, note that the iid property may depend on the performance measure used and the particular HM involved. Hence, it is important to test the iid property when new HMs are generated or when different performance measures are used.

In short, the goal of learning and generalization in an application is to find a new HM that performs better than HM_{base} on the average for all performance measures involved, across all test cases in a subdomain, and across all subdomains in the domain.

2.2 Performance Issues within a Subdomain

There are two performance issues involved in designing HMs in a subdomain. The first involves tradeoffs among different objectives, and the second is concerned with normalization of performance values.

When an application has multiple performance objectives, one common method to identify good HMs is to design a parametric function of the performance measures and to find a HM that optimizes the function. For instance, in an application with quality and cost measures, one possible objective in designing good HMs is to find a HM that maximizes the average quality-cost ratio. We have shown in our earlier work that such may lead to anomalous and often undesirable conclusions [10, 11]. In fact, we can arbitrarily show that one HM is better than another by creating a new function of the objective measures. The problem is generally known as the *multi-objective optimization problem* [12].

One way to address this problem that we have studied [11] is to constrain all but one objective measures and to optimize the unconstrained measure. To apply this method, we start with loose constraints on the constrained measures. If these constraints can be satisfied while optimizing the unconstrained measure, then we can tighten the constraints. The process is repeated until we cannot find feasible HMs that satisfy all the constrained measures. This approach is similar to that of Fonseca and Fleming [7].

Another important issue in evaluating performance in a subdomain is normalization. Normalization is applied to overcome dependence of performance values on problem sizes. Normalization is generally done with respect to HM_{base}.

[1] The vertex-cover problem entails finding the smallest subset of vertices of an undirected graph to cover all its edges in such a way that at least one end of each edge emanates from one of the included vertices. The graph is characterized by its degree of connectivity which measures the probability that an edge exists between two nodes.

Traditional normalization methods may emphasize one performance range while deemphasizing another. For instance, suppose we normalize execution times of a HM with respect to those of HM_{base} to get speedups (SU). Assume that after evaluating the HM on two test cases, we get SU's of 5 and 0.2, respectively; that is, the HM has a speedup of five for the first test case and a slow down of five for the second. The average speedup is 2.6, which obviously is not what is desired.

The problem here is that speedup is in the range between one and infinity, whereas slowdown is between zero and one. Consequently, slowdowns are deemphasized in computing the average speedup.

One way to solve this problem is to compute a symmetric speedup when testing HM_i on test case j.

$$SU_{sym}^{i,j} = \begin{cases} SU^{i,j} - 1 & \text{if } SU^{i,j} \geq 1 \\ 1 - \frac{1}{SU^{i,j}} & \text{if } 0 \leq SU^{i,j} < 1 \end{cases} \quad (1)$$

where $SU^{i,j}$ is the normal way of computing speedup when HM_i is tested on test case j. The average symmetric speedup of HM_i is, therefore,

$$SU_{sum}^{i} = \frac{\sum_j SU_{sym}^{i,j}}{\#\ of\ test\ cases} \quad (2)$$

Note that the symmetric speedup of HM_{base} is zero as speedups of HM_{base} are always one.

Using the definition of symmetric speedup, our earlier example of a HM evaluated on two test cases with speedups of 5 and 0.2 will have symmetric speedups of 4 and -4, respectively. Hence, the average symmetric speedup is 0.

3 Generalization across Subdomains

In this section, we discuss issues and approaches on generalizing new HMs across multiple subdomains. We assume that a HM generator has generated a HM (using domain-specific and domain-independent knowledge), and the task of generalization is to decide whether this HM should be kept. We further assume a generate-and-test learning paradigm similar to that of genetic algorithms [11] in which learning time is divided into generations. In each generation, the HM generator uses HMs retained at the end of the last generation to generate new HMs for the current generation. The operators used by the HM generator include mutation and cross-over. Due to space limitation, we will not describe the learning procedure here.

3.1 Probability of Win and Parallel Learning

To know whether a HM generated is better than HM_{base}, the HM must be tested across all subdomains from which test cases were drawn. As mentioned in the last section, the performance of test cases

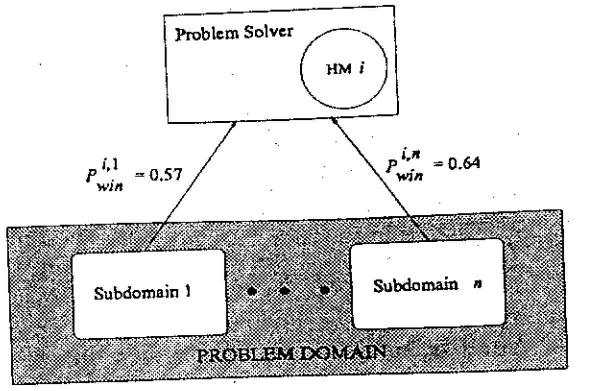


Figure 2: Parallel learning: verifying that a HM satisfies constraints on probabilities of win across multiple subdomains.

may not belong to one common distribution. Consequently, statistical measures cannot be computed for performance values across subdomains. In this section, we present a new measure called probability of win that evaluates performance within a subdomain. Note that we are dealing with a single unconstrained performance measure.

To define the probability of win, consider a collection of sample symmetric normalized performance values of HM_i with respect to HM_{base} in subdomain j. Here, P_{win}^{i}, the *probability of win* of HM_i, is the probability that μ_i (or the actual average symmetric normalized performance of HM_i with respect to that of HM_{base}) is larger than zero. That is, when P_{win} is 0.5, the chance for μ_i to be larger than zero is the same as the chance for μ_i to be smaller than zero.

To make sure that HM_i is better than HM_{base} for all subdomains, we want to ensure that its P_{win} is better than a threshold probability τ in every subdomain. That is, in subdomain j,

$$P_{win}^{i,j} = P(\mu_{i,j} > 0 \mid \hat{\mu}_{i,j}, \hat{\sigma}_{i,j}, n_{i,j}) > \tau, \quad (3)$$

where $\mu_{i,j}$, $\hat{\mu}_{i,j}$, $\hat{\sigma}_{i,j}$, and $n_{i,j}$ are, respectively, the actual average symmetric normalized performance, sample average symmetric normalized performance, sample standard deviation of symmetric normalized performance, and number of samples that HM_i has been tested. For simplicity, index j is not shown in the rest of this paper.

Figure 2 illustrates the case when HM_i is tested across n subdomains, it is is accepted if its probability of win is greater than τ for all subdomains.

τ in Eq. (3) is the threshold probability. When τ is 0.5, it means that the probability for μ_i to be greater than zero is larger than the probability for μ_i to be less than zero. This threshold is deemed too low when only a small number of tests are performed, as it will be difficult to determine whether HM_i is better than HM_{base} or not. On the other hand, when τ is close to 1 and HM_i satisfies Eq. (3), then HM_i is very likely to be better than HM_{base}.

To ensure that HM_i is better than HM_{base}, we need to set τ to be larger than 0.5. In the beginning when little is known about the application, we can set τ to be close to 0.5. If this constraint can be satisfied, we can then increase τ in order to get better HMs. The τ should be increased in such a way that a reasonable fraction of the new HMs generated will have P_{win}'s greater than the threshold probability. Note that many HMs may need to be generated before τ can be increased.

By repeating the generate-and-test learning algorithm until $P_{win} > \tau$, we will obtain a set of HMs that are better than HM_{base}. After learning is completed, the HMs obtained are further verified in detail across all subdomains, including those used in learning.

3.2 Evaluating Probability of Win

In this section, we present three methods to evaluate probabilities of win: interval analysis, maximum likelihood estimate, and Bayesian analysis. Consider a HM with μ_i and σ_i that represent, respectively, the mean and standard deviation of symmetric normalized performance in n_i samples. Recall that HM_{base} has zeroes for both its mean and standard deviation of symmetric normalized performance. We make the following assumptions in our analysis.

- By Central Limit Theorem,

$$P(\hat{\mu}_i \mid \mu_i, \sigma_i, n_i) \approx \mathcal{N}\left(\mu_i, \frac{\sigma_i^2}{n_i}\right). \qquad (4)$$

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

- The actual average symmetric normalized performance of all HMs at any time has a normal distribution with mean μ_0 and standard deviation σ_0, respectively. That is,

$$\mu_i \approx \mathcal{N}(\mu_0, \sigma_0^2) \qquad (5)$$

These two assumptions must be verified on the application performance before applying the following methods. In general, μ_0, σ_0, and σ_i can only be estimated.

Interval Analysis. The following variable

$$T = \frac{\hat{\mu} - \mu}{\sqrt{\hat{\sigma}^2/n}} \qquad (6)$$

is approximately t-distributed with $n-1$ degrees of freedom. Therefore,

$$P_{win} = P(\mu_i > 0) = P\left(T < \frac{\hat{\mu}_i}{\sqrt{\hat{\sigma}_i^2/n_i}}\right). \qquad (7)$$

When $n \to \infty$, we have

$$P(\mu_i > 0) \approx \Phi\left(\frac{\hat{\mu}_i}{\sqrt{\hat{\sigma}_i^2/n_i}}\right). \qquad (8)$$

where Φ is the standard cumulative normal distribution function.

As an example, consider the three HMs whose actual mean and standard deviation of symmetric normalized performance measures are shown in Table 3. The average P_{win} obtained for 5, 10 and 30 samples are very consistent, showing $P_{win} > 0.5$ when a HM performs better than HM_{base}.

Note that P_{win} does not necessarily order HMs by their mean performance. HM_i with μ_i very close to zero but a very small σ_i can have P_{win} close to one. On the other hand, HM_j with μ_j much larger than zero but very large σ_j may have smaller P_{win}. Further, note that Eq. (7) does not take into consideration μ_0 and σ_0 in the derivation and that P_{win} will be close to 0.5 when n is small.

Maximum Likelihood Estimates. Based on the assumptions stated earlier in this subsection, the maximum likelihood estimate of μ_i is

$$\mu_{mle} = \frac{\hat{\mu}_i \sigma_0^2 + \frac{\mu_0 \sigma_i^2}{n_i}}{\sigma_0^2 + \frac{\sigma_i^2}{n_i}} \qquad (9)$$

By using such an estimate, we can find a set of HMs whose μ_{mle}'s are greater than zero across all subdomains. Note that the HMs evaluated can be ordered by their μ_{mle}'s.

Table 3 illustrates the average μ_{mle} for different number of samples drawn for each HM. Here, μ_0, σ_0, and σ_i are estimated from the samples drawn of the three HMs. These examples illustrates that μ_{mle} predicts the correct sign of μ. They also show better predications as more samples are drawn.

Bayesian Analysis. The general form of P_{win} estimated using Bayesian analysis is as follows.

$$\begin{aligned} &P(\mu_i \mid \hat{\mu_i}, \sigma_i, n_i, \mu_0, \sigma_0) \\ &\quad = \frac{P(\hat{\mu_i} \mid \mu_i, \sigma_i, n_i, \mu_0, \sigma_0) P(\mu_i)}{\int_{-\infty}^{\infty} P(\hat{\mu_i} \mid \mu_i, \sigma_i, n_i, \mu_0, \sigma_0) P(\mu_i)\, d\,\mu_i} \\ &\quad = \frac{e^{-a\,\mu_i^2 + b\,\mu_i + c}}{\int_{-\infty}^{\infty} e^{-a\,\mu_i^2 + b\,\mu_i + c} d\,\mu_i} = \mathcal{N}\left(\frac{b}{2a}, \frac{1}{2a}\right) \end{aligned} \qquad (10)$$

Table 3: Examples illustrating (a) average P_{win} found by interval analysis, (b) average μ_{mle} found by maximum likelihood estimation, and (c) average P_{win} found by Bayesian analysis for different number of samples drawn. μ_0 and σ_0 in cases (b) and (c) were estimated from the sample means of the three HMs.

HM_i	μ_i	σ_i	Interval Analysis			Max. Likelihood Estimates			Bayesian Analysis		
			Avg. P_{win} for # Samples			Avg. μ_{mle} for # Samples			Avg. P_{win} for # Samples		
			5	10	30	5	10	30	5	10	30
HM_1	0.336	1.231	0.652	0.725	0.849	0.300	0.287	0.345	0.718	0.786	0.876
HM_2	−0.129	0.222	0.202	0.097	0.012	−0.113	−0.120	−0.126	0.212	0.114	0.020
HM_3	0.514	0.456	0.940	0.991	1.000	0.466	0.474	0.502	0.949	0.993	1.000

where $a = \frac{1}{2}\left(\frac{n_i}{\sigma_i^2} + \frac{1}{\sigma_0^2}\right)$, $b = \left(\frac{\hat{\mu}_i}{\sigma_i^2/n_i} + \frac{\mu_0}{\sigma_0^2}\right)$ and $c = \left(\frac{-\hat{\mu}_i^2}{2\sigma_i^2/n_i} - \frac{\mu_0^2}{2\sigma_0^2}\right)$. Using Eq. (3), we have

$$P_{win}^i = P(\mu_i > 0 \mid \hat{\mu}_i, \sigma_i, n_i, \mu_0, \sigma_0) = \Phi\left(\frac{b}{\sqrt{2a}}\right)$$

Note that μ_0, σ_0 and σ_i must be estimated before the above equation can be applied. These estimates may not be accurate because each HM is only tested a few times during each iteration of the generate-and-test learning algorithm.

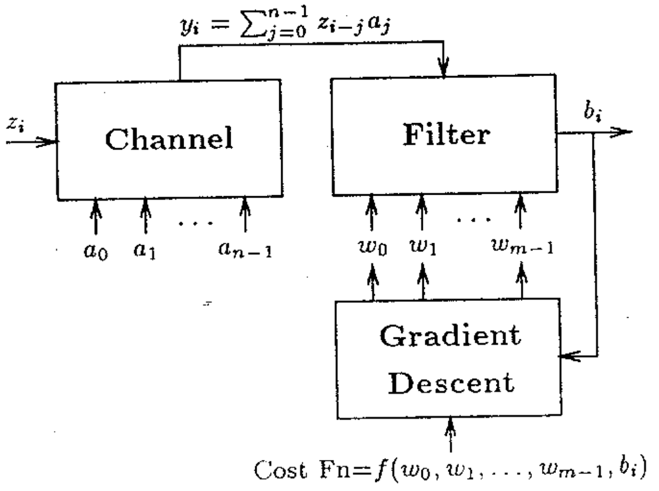
Continuing with the previous example, Table 3 shows the average P_{win}'s estimated using Bayesian analysis. The results show that the average P_{win}'s are higher than those of interval analysis. This happens because μ_0 is larger than zero.

4 Experimental Results

In this section, we summarize some experimental results on blind equalization and branch-and-bound search. The learning algorithm used in each subdomain is a genetic algorithm. Due to space limitation, results on other applications are not shown.

(a) We have applied the proposed generalization procedure in a genetic algorithm to learn 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 3). The equalization process is equivalent to adjusting the weights of a FIR filter in order to minimize the value of a cost function (using gradient descent). In our experiments, the cost function is defined in term 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 blind equalizer with a fixed set of random initial weights. 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



$y_i = \sum_{j=0}^{n-1} z_{i-j} a_j$
z_i
Channel
Filter
b_i
a_0 a_1 ... a_{n-1}
w_0 w_1 ... w_{m-1}
Gradient Descent
Cost Fn=$f(w_0, w_1, \ldots, w_{m-1}, b_i)$

Figure 3: Blind equalization process for recovering input data stream for n-th order channel and m-th order filter.

($|a_i| > \sum_{i \neq j} |a_j|$ where a_i is the i-th weight of the channel) to the hardest one ($a_i = a_j$ for all i and j).

Table 4 shows the average symmetric improvements in terms of number of accumulated errors, HM_{base} (CMA 2-2) [4], and the new HM found after learning and generalization.

(b) We have also applied the proposed generalization procedure in a genetic algorithm to learn new decomposition HMs used in a branch-and-bound search for solving the vertex-cover (VC), traveling-salesman (TSP) and knapsack (KS) problems. Here, a decomposition HM expands a partially expanded node selected for expansion into descendant nodes using operators to translate (or expand) the search node. The operators used are based on measurable attributes in the selected node.

In solving a TSP, an active node is one whose route has been determined for a subset of the cities, and a decomposition HM expands a selected active node into descendants by picking one of the unvisited cities

Table 4: Summary of average symmetric improvements in terms of number of accumulated errors for the learned cost function over ten subdomains. (b_i in Figure 3 is the instantaneous value of b.)

Average Sym. Improvement				Orig. HM	New HM
Avg.	Std. Dev.	Max.	Min.		
0.153	0.395	0.694	−0.465	$b^3 - b$	$4b^3 - 2b^2 Sign(b) - b$

Table 5: Summary of average symmetric speedups of learned decomposition HMs (l: number of uncovered edges or live degree of a vertex; n: average live degree of all neighbors; Δl: difference between l from parent node to current node; c: length of current partial tour; m: minimum length to complete current tour; p: profit of object; w: weight of object).

Appl.	Avg. Sym. Speedup			Orig. HM	New HM
	Learn.	Gen.	Valid.		
VC	0.012	0.068	0.088	l	$1000 \times l + n - \Delta l$
TSP	0.188	0.080	0.231	c	$m\ c$
KS	0	0	0	p/w	p/w

and expanding it into two descendants, one visiting the city next and one that does not. In VC, given a set of partially covered graph, a decomposition HM selects one of the unselected vertices to expand into two descendants, one including the selected vertex and one that does not. Finally, in KS, a decomposition HM selects one of the unselected objects to be included or not included in a knapsack with fixed capacity. Due to space limitation, we do not show results of other experiments [11].

Table 5 shows the average symmetric speedups, the original HM used in each problem, and the new HM found after learning and generalization. An interesting point of the HM found for solving VC is that l is the primary index, and $n - \Delta l$ is the secondary index.

5 Conclusions

In this paper we have described an important problem encountered in experimenting heuristics to solve application problems, namely, how to evaluate statistically that one heuristic method performs better than another. We have introduced the concept of probability of win and have present some preliminary results on its evaluation.

References

[1] A. Newell, J. C. Shaw, and H. A. Simon, "Programming the Logic Theory Machine," in *Proc. 1957 Western Joint Computer Conf.*, pp. 230–240, IRE, 1957.

[2] J. C. Yan and S. F. Lundstrom, "The Post-Game Analysis framework–developing resource management strategies for concurrent systems," *IEEE Trans. on Knowledge and Data Engineering*, vol. 1, pp. 293–309, Sept. 1989.

[3] M. L. Litzkow, M. Livny, and M. W. Mutka, "Condor - a hunter of idle workstations," in *Proc. 8th Int'l. Conf. Distributed Computer Systems*, pp. 104–111, IEEE, 1988.

[4] S. Haykin, *Blind Deconvolution.* Englewood Cliffs, NJ: Prentice Hall, 1994.

[5] D. A. Waterman, "Generalization learning techniques for automating the learning of heuristics," *Artificial Intelligence*, vol. 1, no. 1/2, pp. 121–170, 1970.

[6] T. M. Mitchell, "Generalization as search," *Artificial Intelligence*, vol. 18, pp. 203–226, 1982.

[7] 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*, pp. 416–423, Morgan Kaufman, June 1993.

[8] H. W. Lilliefors, "On the Kolomogorov-Smirnov test for normality with mean and variance unknown," *American Statistical Association Journal*, pp. 399–402, June 1967.

[9] W. Feller, *An Introduction to Probability Theory and its Applications*, vol. 1. New York, NY: Wiley, 1968.

[10] B. W. Wah, "Population-based learning: A new method for learning from examples under resource constraints," *IEEE Trans. on Knowledge and Data Engineering*, vol. 7, pp. 454–474, Oct. 1992.

[11] 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*, vol. 7, Oct. 1995.

[12] F. W. Gembicki, *Vector Optimization for Control with Performance and Parameter Sensitivity Indices.* PhD thesis, Case Western Reserve University, Cleveland, OH, 1974.