INTELLIGENT MAPPING OF COMMUNICATING PROCESSES IN DISTRIBUTED COMPUTING SYSTEMS

Arthur Ieumwananonthachai, Akiko N. Aizawa,
Steven R. Schwartz, Benjamin W. Wah and Jerry C. Yan

ABSTRACT

In this paper we present TEACHER 4.1, a system for designing automatically heuristics that map a set of communicating processes on a real-time distributed computing system. The problem of optimal process mapping is NP-hard and involves the optimal placement of processes on the distributed system and the optimal routing of messages from one computer to another. The design of efficient and robust heuristics is often ad hoc and is guided by intuition and experience of the designers. In this paper we develop a statistical method to explore systematically the space of possible heuristics for process mapping. The method operates under a specified time constraint and intends to get the best possible heuristics while trading between the solution quality and the execution time of the process mapping heuristics. Our prototype for process mapping is extended from post-game analysis, a system that uses a set of user-specified rules for generating new mappings. It tunes parameters of these rules and proposes new heuristics for process mapping. Simulations show that there is significant improvement in performance through systematic and automatic exploration of the space of heuristics.

KEYWORDS AND PHRASES. Distributed computing system, heuristics, generate-and-test, loosely coupled computers, process mapping, sequential selection, time constraints.

1. INTRODUCTION

In real-time computing systems, the efficient scheduling of resources is essential in order to meet resource constraints and processing requirements. In such systems, applications executed have somewhat predictable behavior and are often analyzed beforehand. Many formulations of this problem are NP-hard, and optimal mappings are often found in an exponential amount of time and/or space. Heuristics can be applied, but their design is usually ad hoc and depends on the problem instance to be solved and the experience of the designers. Our goal in this paper is to develop a statistical method for exploring systematically the space of possible mapping heuristics. Our method operates under specified time constraints and intends to get the best heuristics while trading between the solution quality of the heuristics and its execution time.

Arthur Ieumwananonthachai, Akiko N. Aizawa, Steven R. Schwartz, and Benjamin W. Wah (contact author, wah@aquinas.csl.uiuc.edu) are with Center for Reliable and High-Performance Computing, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1101 West Springfield Avenue, Urbana, IL 61801. A. N. Aizawa is currently on leave from National Center for Science Information System, Tokyo, Japan. Jerry C. Yan is with Sterling Federal Systems, Inc. and can be reached at MS244-4, NASA Ames Research Center, Moffett Field, CA 94035.

This research was supported in part by National Aeronautics and Space Administration Grant NCC 2-481 and in part by National Science Foundation Grant MIP 88-10584.

Proceedings Supercomputing 91, Albuquerque, NM.

The target problem we study is the mapping of a set of communicating processes derived from an application on a network of loosely coupled computers. Its objective is to determine the mapping of the processes on the given system so that the completion time of the application is minimized. We assume that a process cannot be partitioned at run time, that the mapping is static, that a process is not replicated and executed in multiple computers, that the system is scheduled for one application at a time, and that the processing behavior of the application can be reproduced easily at design time.

One method that finds efficient mappings is the post-game analysis system developed by Yan and Lundstrom [13, 14]. Given a set of communicating processes and a set of input data, the system incrementally changes the process mapping using information collected in previous runs. Based on a set of designer-supplied rules, it (a) formulates multiple optimization subgoals, based on actual timing data gathered during program execution, (b) proposes possible new mappings, and (c) prioritizes and resolves conflicting proposals. The key of the design is that the target application is simulated and its mapping incrementally improved, without employing any abstract program models or any single objective function.

The mappings proposed by post-game analysis can be improved because the rules for proposing and for evaluating mappings are derived based on the experience of the designers. Moreover, these rules do not adapt to conditions of the computing system, such as its hardware configuration, average workload per processor, and mix of computation and communication of processes. In this paper, we extend post-game analysis by developing a set of meta-rules, which systematically generate and test new rules for proposing and for evaluating mappings. As there are infinitely many possible rules to be used in post-game analysis and it is impossible to test each of them, our meta-rules propose a limited set of rules and test them under given time constraints. We develop a statistical model for trading between the number of new rules to be generated and the amount of tests to be performed on the ones generated.

The system studied here is extended from TEACHER (which stands for *TEchniques for Automatic Creation of HEuRistics*). Previous versions of this system has been applied on learning (a) new dominance relations for dynamic programming algorithms [15], (b) good selection and decomposition heuristics for combinatorial searches [7], (c) good parameters for depth perception in stereo vision [11], and (d) new neural network configurations [12]. The current version (version 4) is focused on resource scheduling under time constraints in the generation of new heuristics.

2. PROCESS MAPPING PROBLEM

There have been a number of studies for finding good mappings for a set of communicating processes on a network of computers. The objective often used is the completion time of the set of processes. Approaches to solving this problem can be classified into static and dynamic.

Static strategies are generally deterministic and are applied at design time. These strategies can further be classified as cost based and graph based. Cost-based methods employ simplistic objective functions to evaluate merits [6], while graph-based methods employ graph-theoretic techniques [3]. The general forms of these problems are NP-hard, and their abstract models do not capture adequately the problem behavior. Minimization of actual execution time is not guaranteed.

In contrast, dynamic strategies are non-deterministic and are applied at run time. They are based on partial and probably out-of-date information gathered dynamically. Examples of this approach include load sharing, load balancing, bidding protocols, and Bayesian decision theory. Results developed are generally restricted to independent jobs. One major problem with this approach is that it is very difficult to characterize analytically the behavior of interacting processes. The interactions between the computation time of the processes, the time spent while waiting for messages from other processes (or waiting time[1]), and the time spent while waiting for the processor to be available (or contention time[1]) are too complex to be predicted accurately and studied in advance. This leads to simplified objective functions and models that ignore either the waiting or contention times and characterize workload on a computer by the number of active processes.

Post-game analysis is a method that has been successful in addressing some of the difficulties discussed above. This method iteratively refines the current mapping of processes on a network of computers using information collected in between program executions. The processes are executed (or simulated) first using a random mapping. Based on data collected during program execution, heuristics (or rules) are applied to propose changes to this mapping in hoping that the execution time will be reduced. Each heuristic represents an independent line-of-reasoning concerned with one particular optimization subgoal for reducing program execution time. It is used to propose "actions" that are partial descriptions of the transformations needed in order to reduce execution time. The actual transformations needed are generated based on these proposals. The program is then executed (or simulated) using the new mapping. The entire process is repeated again until no new changes are proposed by the heuristics. To aid in experimenting with different machine architectures, a simulation package called AXE was developed [13]. AXE allows flexible specification of software and hardware, and simplifies the collection of statistics during the simulation.

The type of architectures that post-game analysis is designed for are known as *ensemble architectures*. They consist of homogeneous and regularly connected processing elements (or sites). Each site is autonomous, and has its own processor, local memory large enough to hold any number of processes, and operating system that handle sending, receiving, and routing of all messages. Examples of architectures modeled include Hypercube multicomputers and distributed systems connected by local area networks.

Each distributed program is represented as a collection of autonomous processes that are created either at compile time or at run time. The processes communicate by messages and have no shared data. This model encompasses programs using divide-and-conquer strategy, blackboard problem-solving paradigms, and programs utilizing pipelining.

The key to improving the performance of the post-game analysis is on the development of heuristics that assess improvements achieved and migration actions to be taken next. Currently, the designer must supply rules for proposing and evaluating mapping decisions. Using meta rules, we can systematically and automatically propose new rules that may lead to better mappings in less time. Moreover, these meta rules can propose rules that generate mappings adapted to the architecture configuration, the mix between communication and computation of processes, and the application characteristics. In the following sections, we describe our proposed modification to post-game analysis.

3. GENERATE-AND-TEST FRAMEWORK UNDER RESOURCE CONSTRAINTS

The problem of finding a set of heuristics that maximize the performance of the mappings generated by post-game analysis can be expressed as a search through the space of possible heuristics. The search is characterized by an objective to be optimized and a set of constraints to be satisfied.

Informally, the *objective function* of such a search is to maximize the quality of the heuristics; that is, to perform the best tradeoff between the quality of the mapping of a given problem instance and the time spent by the post-game analysis in deriving the mapping. This objective has to be specified by the users. Examples of objectives include maximizing the average or worst case of a function of quality and time. Since there is no good model on how the heuristics affect the objective function, we cannot compute but can only measure the objective-function value based on the heuristics and the problem descriptions. Likewise, there are no well-defined constraints that define the search space of the heuristics. Consequently, this search is considered to have an ill-defined objective as well as ill-defined constraints.

Due to the large heuristic space and the large amount of time required for the evaluation of a heuristic set, it is impractical to enumerate the objective values of all heuristics. A common approach to solving this kind of problems is to ask knowledgeable experts select a set of good heuristics. However, the set of possible heuristics in this problem is extremely large, and parameters that affect their performance may be dependent and unknown to the experts. On the other hand, performing naive experimentation by computers is not feasible, as the search space is too large.

A desirable solution is to employ expert knowledge in guiding the search for better heuristics, and utilize high-speed computers to perform computation-intensive experiments. The generate-and-test framework studied in this paper uses expert knowledge (in the form of meta-rules) to generate new heuristics (in the form of rules used in post-game analysis) and utilizes high-speed computers to test the quality of the heuristics using benchmark test cases.

1. The contention time depends on the processor scheduling strategy, while the waiting time depends on the routing strategy.

3.1. Possible Solution Methods

The solution to the problem discussed above lies in finding an efficient and automated method for generating and evaluating heuristics.

The problem of discovering new heuristics was first addressed by Pearl [9], who explores the paradigms that heuristics are discovered by consulting simplified models of the problem domain. Pearl's scheme involves systematic refinement and deletion of constraints from the original problem specification until a semi-decomposable model is identified. The solution to the latter constitutes a heuristic to the former.

Many paradigms for machine learning can be applied as well. Machine learning strategies studied in the literature include rote learning, learning by instruction, learning by deduction, learning by analogy, and learning by induction [8].

In order to select the learning paradigm for our problem, the characteristics of the problem must be studied. The key characteristics for the process mapping problem are:

- *Knowledge-lean domain*: there is little knowledge for generating good candidate heuristics;
- *Explicit feedback*: the effect of applying a heuristic on a test case can be assessed quickly;
- *Scalability*: performance of heuristics scale reasonably well between small and large test cases;
- *Large solution space* and *limited resources*.

Due to these characteristics, learning by induction, and in particular, learning by experimentation, is particularly suitable for learning new heuristics. In this domain, resource scheduling is very important in the learning process. Note that other learning methods, such as learning by deduction and learning by analogy, do not work well for knowledge-lean problems.

In this paper we focus on studying the generate-and-test mechanism under a fixed time constraint. The major problem addressed is on the scheduling of resources for systematically exploring the space of possible heuristics so that the best heuristics can be obtained when time is exceeded. The tradeoff is on the number of heuristics to test and the amount of testing performed on each within the time limit. Our heuristics generator is somewhat primitive at this point and can be enhanced in the future through Pearl's method [9].

3.2. Generate-and-Test Framework

This framework is an application of a generate-and-test strategy to search an ill-defined search space under resource constraints (see Figure 3.1). It consists of three main parts: the *candidate generator* generates new candidates to be considered, the *evaluator* applies the candidate to the target problem and records the performance of this candidate, and the *scheduler* makes decisions about the best way to use the resources. A candidate is a point within the search space, which in this case is a set of heuristics for the post-game-analysis system.

To avoid spending a large amount of time on poor candidate, the evaluation process is divided into small sub-tests called *quanta*. This allows the system to perform additional tests on candidates only if they demonstrate some merits during previous quanta. During a quantum of time, tests are performed on the candidate selected using test cases randomly generated or supplied by the users. There are two possibilities here: either choose relatively small test cases so that they can be tested fully within the quantum of time, or choose large test cases dictated by the application requirements to actually test the candidate. In the latter case, the evaluation of the candidate on the test case within the quantum may not be completed. Its advantage is that the candidate is tested under realistic conditions; however, its drawback is that if the evaluation of the test case is not completed at the end of the assigned quantum, it is difficult to assess which candidate to test next. Since in the process mapping problem candidates scale reasonably well between small and large test cases, we test the candidate selected using small test cases and update the statistical behavior of the candidate at the end of the quantum.

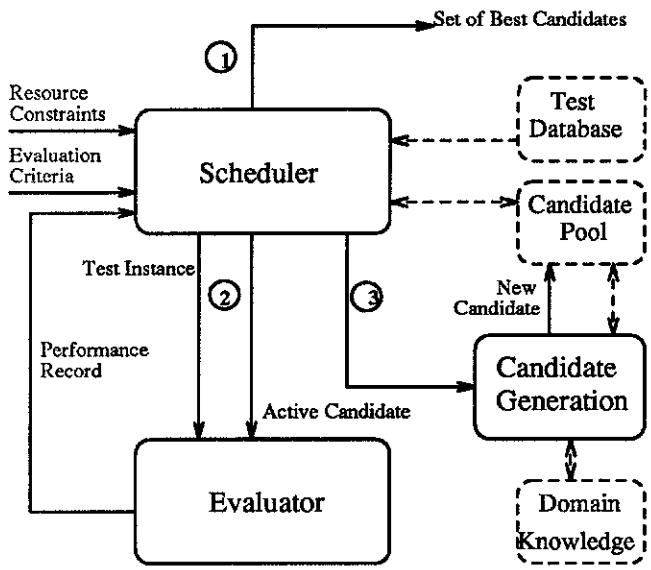

Figure 3.1. Generate-and-Test Framework.

In other application domains, the behavior of candidates may not scale well. For example, in designing an artificial neural network for an application [12], it may not be possible to assess the quality of a configuration at the end of a quantum, as the network is too large to be tested fully. It is also not possible to test the patterns on a small network and generalize the network to a larger one. Consequently, the decision made at the end of a quantum are heuristic rather than statistical, and may be prone to errors.

At the end of a quantum, a decision is made by the scheduler to perform one of the following actions: (1) select the next candidate to test from the candidate pool, (2) generate a new candidate to be placed in the candidate pool and possibly remove an existing one from the pool, and (3) select a set of best candidates and stop learning when time is expended. The decision between choices (1) and (2) is made based on the current performance of the candidates in the pool and how much evaluation has been done on each. One simple method for determining when to generate new candidates for the pool is to simply generate new ones whenever existing ones have been evaluated to within a statistical confidence.

If the decision is made to pursue choice (1) or (3), then the candidate is selected based an *evaluation criterion*. This consists of two parts: the *goodness* and the *guidance* functions.

The goodness function is an estimator of the value of the objective function. It is used for selecting a candidate in the pool that most likely performs the best if the learning process were to be stopped at this time. It is needed because candidates may not be fully evaluated to within a statistical confidence when learning is terminated.

The guidance function is used for selecting the candidate to test in the next quantum if learning were to be continued. Its goal is to choose the candidate that will maximize the probability that the candidates with the highest objective values also have the highest goodness values. The choice is not always to select the most promising candidate to test because a candidate may show less promise with the limited tests performed but will become better with more tests. Moreover, with limited resources, it may be necessary to explore more candidates early in the learning process and focus on a limited set of promising ones as time is running out. A statistical model that addresses this tradeoff is shown in Section 4.

If the decision is made to pursue choice (2), then a new candidate has to be generated. The candidate generator must be intelligent in guiding the generation of new and hopefully better candidates. It should use past performance of existing candidates in the pool as well as any domain knowledge it has. This part of the framework can be incrementally improved using higher-level learning strategies such as learning by deduction, learning by explanation, and relaxation methods [9]. Generate-and-test is not applicable to improving the candidate generator because the overhead of testing a new meta-rule could be extremely high. We assume that candidate generation is driven by a fixed set of meta-rules in this paper. The learning of new meta-rules will be studied in the future.

4. STATISTICAL SELECTION FOR FINDING THE BEST CANDIDATE UNDER TIME CONSTRAINTS

The problem of finding the best candidate by performing a series of tests has traditionally been known as the *sequential selection problem*. The following section details what work has been done with this problem in the past and the difficulties that have been encountered. We also describe our proposed approach to address these problems and demonstrate the effectiveness of our technique.

4.1. Problem Formulation

In our case we are faced with chosing the best candidate from a pool of candidates, each of which is associated with a set of performance values. This problem is formally stated as follows: given a set of populations consisting of normally distributed numbers with unknown means and variances, select the one with the highest population mean by testing a certain number of samples from these populations.[2] In our case, the populations are candidate heuristics, and the numbers comprising the elements of the populations are the performance values associated with applying the heuristic to a problem instances.[3] Making one pick from a population is analogous to testing the candidate on one problem instance. The goal is to chose the candidate with the highest mean[4] within a given number of tests.

2. Population mean and variance are properties of a population. They can be represented by the sample mean and variance if limited samples are drawn from the population or if the population is infinite in size.

3. A problem instance (or test case) is represented by the specification of the set of communicating processes to be mapped, the hardware on which the processes are mapped, the input values to the distributed processes, and the set of initial random mappings for the processes. The performance of a problem instance is averaged over the set of initial random mappings.

4. The highest mean is used as the objective here. As discussed in Section 5.1, other objectives may be specified.

4.2. Related Methods

Two naive approaches for this problem are: (1) the *random/round-robin* method, which takes samples from each population in turn, and (2) the *greedy* method, which takes samples from the population that currently has the maximum sample mean. However, there are problems with both methods. The problem with round-robin is that the number of picks made from each population is the same. In our case, it seems unnecessary to demand we test the worst candidate just as much as the best, ignoring how initial tests may quickly demonstrate the disparity in performance between them. On the other hand, the greedy method considers only candidates that look good at the current stage and might discard the best candidate at an early stage. This could result in poor performance.

Work related to this area was pioneered by Bechhofer in 1954 [1]. The result of his method allowed the tester to determine the minimum amount of picks necessary to know which population is the best to within a certain degree of confidence. Many extensions have been proposed by researchers to accommodate various tradeoffs and other goals of selection. Some papers deal with the case of an unequal number of samples [2]. Optimal algorithms to this problem do not exist.

The major problem with applying these traditional methods to our case is that the emphasis in the past has been on the stopping criteria for testing a finite set of populations rather than on finding the best population within a given time constraint. The traditional sequential selection problem dictates that at each time step all the populations are sampled once. Following this, a measure is calculated to determine if the desired degree of confidence has been attained or if testing should continue. None of the previous methods have considered the case where there are possibly more populations than total testing time allowed, or when one knows *a priori* how many picks can be made. Both of these conditions are highly relevant in our case, because we know our deadlines and have far more options than we could possibly pursue.

4.3. Proposed Approach

In this section we first propose an overall guidance strategy that combines individual guidance strategies. Following this we discuss a minimum-risk guidance strategy to overcome the worst-case scenario of the greedy method while maintaining the desirable property of avoiding extensive tests on inferior candidates.

4.3.1. Multistage Testing

Due to the limited available testing time, we formulate the overall guidance strategy, $G(T)$, as a series of stages, $G_i(t_i,g_i,k_i)$, where i ranges from 1 to the number of stages. Each stage is charterized by a triplet consisting of (a) t_i, the duration of the stage, (b) g_i, the particular guidance strategy to be used for the stage, and (c) k_i, the number of candidates to be

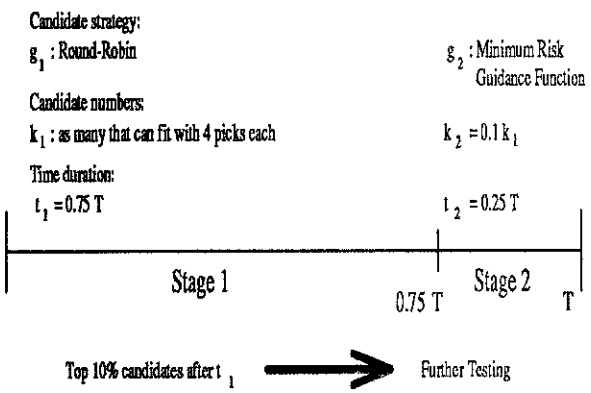

Figure 4.1. Multistage Testing Procedure.

considered for testing during this stage. The resulting combination gives an overall testing strategy that takes into account limited resources like testing time, and can deal with situations in which there are so many candidates that it would be impossible to test them all just once.

The parameter values for each stage are calculated heuristically based on knowledge of the problem. Factors that affect these settings include the size of each population, the total amount of testing time, and the number of possible candidates. Currently, we have focused on two-stage testing. The first stage consists of a round-robin strategy, and the second is a minimum-risk procedure of our own design. The two-stage procedure is illustrated in Figure 4.1. Intuitively, the two stages correspond to coarse initial testing to weed out unworthy candidates followed by a careful evaluation to select the best.

Our heuristic for setting the stage parameters is to spend the first 75% of the time performing round-robin tests at 4 tests per candidate, $G_1(0.75T, RR, 0.75T/4)$. The number of candidates is then selected accordingly. The second stage then selects a subset of the best performers for continued experimentation. The size of this subset is set to allow each remaining candidate to be tested to within a statistical confidence. Initial strategies tested in the second stage include (1) *round-robin*, (2) *greedy*; and (3) *minimum risk* (see next section).

Method (2) is based on an estimation of population means, and method (3) is based on an estimation of both the population means and population variances. Thus these methods rely on a small amount of *a priori* test results from the first stage. We have made all stages explicit in order to study the performance tradeoffs necessary for providing a robust solution.

4.3.2. Minimum Risk Guidance Strategy

We propose the minimum-risk guidance strategy to overcome the worst-case problems of the greedy method without losing the desirable property of avoiding extensive tests on inferior candidates. We define a risk function that is expressed as the expected square-error loss of the mean of each population, weighted by the probability of selection calculated using a joint t-distribution. This guidance strategy selects the next candidate so that the weighted risk is locally minimized.

Formally, our minimum-risk guidance is defined as follows. Let m be the number of populations. For population i, let μ_i be its population mean, σ_i be its population standard deviation, $\bar{X}_i$ be its sample mean, S_i be its unbiased sample standard deviation, and n_i^k be the number of samples tested in stage k. If L_i, the estimation loss for population i, is expressed using the squared-error function, then

$$L_i(\mu_i, \bar{X}_i) = (\mu_i - \bar{X}_i)^2 \tag{4.1}$$

When the candidate with the currently maximum sample mean is selected in stage k, the expected estimation loss (or risk) is expressed using the following equation.

$$E[L^k(\mu_i, \bar{X}_i)] = \sum_{i=1}^{m} P_i^k \, E[L_i(\mu_i, \bar{X}_i)] \tag{4.2}$$

where P_i^k represents the likelihood that population i is the best. Because σ_i's are unknown in our case, the variable $\mu_i - \bar{X}_i/(S_i/\sqrt{n_i^k - 1})$ has Student's t distribution with degree of freedom $(n_i^k - 1)$ and variance $(1 + 2/(n_i^k - 3))$, $n_i^k > 3$. This leads to the following equation.

$$E[L_i^k(\mu_i, \bar{X}_i)] = \sum_{i=1}^{m} P_i^k \left(1 + \frac{2}{n_i^k - 3}\right) \frac{\sigma_i^2}{n_i^k} \tag{4.3}$$

Eq. 4.3 shows that uncertainty about σ_i increases the risk of estimation by the factor $2 / (n_i^k - 3)$. P_i^k is also defined using t distribution as follows,

$$P_i^k = \int \prod_{i \neq j} F_t\left(n_i^k - 1, \frac{y - \mu_i}{\sigma_i / \sqrt{n_i^k - 1}}\right) dF_t\left(n_j^k - 1, \frac{y - \mu_j}{\sigma_j / \sqrt{n_j^k - 1}}\right) \tag{4.4}$$

where $F_t(v, x)$ is the cumulative distribution function of the t distribution.

The policy to be taken in stage k is the one which, under the constraint that only one of n_j^k can be increased by 1, minimizes Eq. 4.3 in stage $(k+1)$. In the special case when we know the "best" population, the choice of this strategy is simple: only the "best" population is tested, because $P_j = 1$ if $\bar{X}_j$ is the "best" one, and 0 for other populations.

The actual value of P_i^k is calculated based on the information we currently have in stage k. We use unbiased estimators $\bar{X}_i$ and S_i for μ_i and σ_i, respectively. This implies that we need at least four samples from each population in order to apply this guidance strategy.

4.4. Experimental Results

In this section we present some Monte Carlo simulation results to show the usefulness of our guidance strategies. Distributions of populations are generated from the distributions of target problems studied in Section 6. To avoid biasing our judgement of candidate quality, we select a random problem instance from our problem set for testing.

Figures 4.2 and 4.3 show the Monte Carlo simulation results of the *rank* of the selected population averaged over 100 iterations for each guidance method. Rank 100 is the best. In Figure 4.2, $T = 533$ time units and for two-stages methods, $t_1 = 0.75T$. Round-robin search (RR) is used in the first stage, and one of minimum-risk (MR), round-robin, and greedy (GR) searches is used in the second stage. They are compared with one-stage round-robin. All two-stage methods perform better than the one-stage round-robin method. In Figure 4.3, we set T equal to 160 units and t_1 to 40 units. In this case, the one-stage round-robin search performs very poorly.

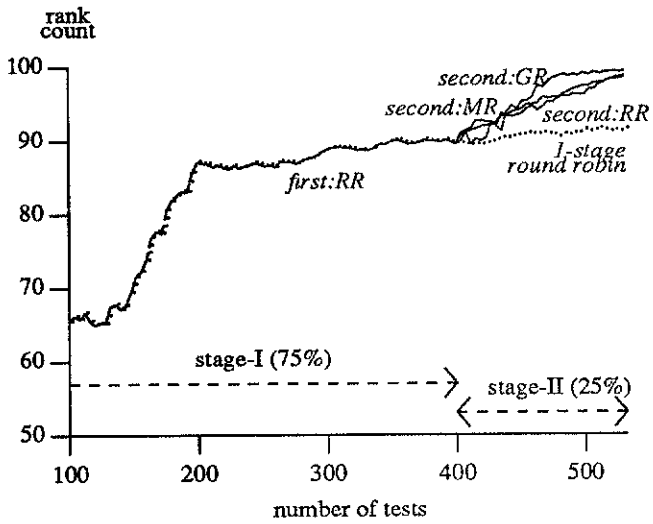

Figure 4.2. Average rank of selected candidate for T = 533 time units and $t_1 = 0.75T$.

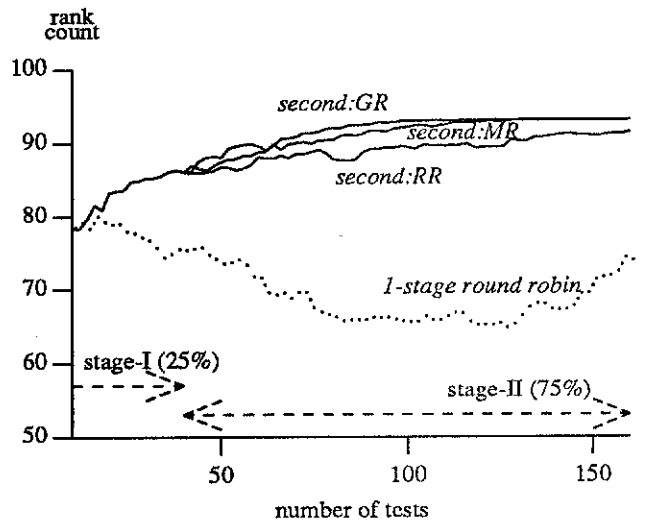

Figure 4.3. Average rank of selected candidate for T = 160 time units and $t_1 = 0.25T$.

A greedy search in the second stage seems to perform better than a minimum-risk search. The reason for this is that in our problem the variances of populations are relatively small and the statistical estimation of the population means is rather accurate. Hence, a greedy method has less chance to get into the wrong direction. Information on the percent deviation from the value of the best as a function of time also demonstrates this tendency.

Figure 4.4 shows an example in which the greedy method performs poorly. In this case, ten normally distributed populations are generated: nine of these have small variances (N(0.02, 0.035)), and one has a better sample mean but a much larger

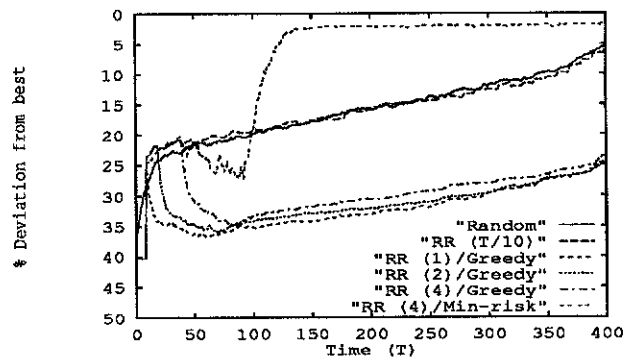

Figure 4.4. Example when greedy selection fail.

variance (N(0.04, 0.2)). The minimum-risk method is found to be superior in identifying early in the process the best candidate, while the other methods are misled by the distributions with small variances.

From these results, we select strategy $G(T)$ = $\{G_1(0.75T, RR, 0.75T/4), G_2(0.25T, MR, k_2)\}$ for our current implementation discussed in Section 6.

5. LEARNING HEURISTICS FOR POST-GAME ANALYSIS

In this section, we present our implementation of the generate-and-test framework discussed in Section 3 for finding new heuristic candidates. We first discuss formally our objective of the framework, namely, the kind of heuristic candidates we are seeking, and briefly discuss the heuristics used in post-game analysis, the performance evaluator, and the candidate generator.

5.1. Objective of Learning Process

The objective of our learning process is to develop a good heuristic that lead to better process mappings. There are two conflicting factors that affect the goodness of a heuristic: its execution time (or cost), and the quality of its solution, and a tradeoff on them must be performed. The following two boundary cases are unrealistic: (a) unlimited time for the heuristic to execute and hence the best mapping, (b) zero time for the heuristic to execute and hence zero cost.

We select an objective to be used in our learning process. Its specification depends on the application requirements. Let $c(h,v)$, $t(h,v)$, and $q(h,v)$ be the cost, execution time, and quality of the heuristic candidate h and target mapping problem v under consideration. Each of these three values are averaged over a set of initial random mappings.

We first define $c(h,v)$ as a piecewise continuous function with a discontinuity at t_{max}, where t_{max} is the maximum time that a heuristic is allowed to execute before a penalty is imposed. Hence,

$$c(h,v) = \begin{cases} 1 & t(h,v) \le t_{max} \\ c_r\,[t(h,v) - t_{max}] + 1 & t(h,v) > t_{max}, \end{cases} \qquad (5.1)$$

where c_r is the relationship between cost and execution time when t_{max} is exceeded. The reason for choosing t_{max} is to

avoid the degenerate case when $c(h,v)$ and $t(h,v)$ are both zeroes.

We define $q(h,v)$, the quality of a heuristic candidate for mapping problem v averaged over a set of initial random mappings as the reciprocal of the completion time of the processes mapped on the distributed system averaged over a set of initial random mappings using heuristic h. That is, the quality is higher when the processes mapped complete sooner.

$Q(L)$, the objective of our learning process L, is to find a heuristic that maximizes the average quality-cost ratios, that is,

$$Q(L) = \max_{h} \sum_{v} \frac{q(h,v)}{c(h,v)} \qquad (5.2)$$

Note that we choose the average quality-cost ratio to maximize in Eq. 5.2. This implies that we place equal weights on each problem instance.[3] Note that the information on the set of representative problem instances and their weights on the final performance must be supplied by the system designers.

5.2. Representing Post-Game Heuristics as Frames

The original rules used in post-game analysis are coded as programming constructs that are not easy to modify by the learning system. To simplify the modification of these rules, they are represented as frames in our implementation.

There are three types of heuristics in post-game analysis, which represent the three stages of the system. The first type of heuristics is used in the performance-evaluation stage for generating proposals based on independent optimization subgoals. The second type of heuristics is used in the priority-assessment stage for determining the order in which sites and processes are processed based on proposals related to them. The final type of heuristic is used for checking the feasibility of a move created from the highest-priority site and process.

One basic need for all three types of heuristics is to be able to represent *expressions*. An expression defines the relationship between the result and values of various variables. In this case, each expression is stored in the form of an expression tree. Each intermediate node represents an operator and each leaf node represents a variable or constant. The set of operators include arithmetic, relational, and logical operators, and additional pre-defined functions. It is evaluated by a post-order traversal of the expression tree. At each intermediate node, the children are evaluated first, then the values from the children are combined using the specified operator. Note that the logical operators are evaluated with short-circuit evaluation.

Each *proposal-generation heuristic* is represented as a record composed of four fields.

- Reason field specifies the motivation behind the heuristic.
- Quantifier field specifies the scope of the heuristic.
- Condition field is an expression specifying the condition in which proposals are generated.
- Action list specifies the proposals to be generated. Each action is a record containing the action type, parameters for the proposal, and an expression specifying the weight of the proposal.

Each heuristic is applied according to its quantifier field. The condition expression is evaluated for each subject specified by the quantifier. When the expression returns a non-zero value, each action in the action list generates a proposal. Each proposal has a weight computed by evaluating the weight expression.

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

Feasibility heuristics can be represented as a collection of expressions. Each expression represents a condition in which the move would be infeasible. For each move considered by post-game analysis, an expression is evaluated. If any expression return a non-zero value, then the corresponding move is rejected.

Because heuristics are represented as frames, they can be read in from a data file. For this learning framework, a set of heuristics is used to represent a *candidate*, a point within the search space. Each set of heuristics is stored in a file which can be easily assimilated by post-game analysis.

5.3. Candidate Evaluator

This component is used to evaluate the performance of heuristic candidates on problem instances. The performance of each candidate is found by using the heuristic candidate to find the mapping for all problem instances in the test database.[3] As the performance of each heuristic candidate evaluated on the same problem with different initial mappings can vary widely, it is necessary to test each problem instance enough to get a good confidence on the value of the average performance. This is done by testing each test case until the 95% confidence interval (based on the Student's t-distribution) of the average performance across the different initial mappings are within 5% of the average value.

The results from each evaluation of a heuristic candidate for a problem instance with an initial mapping are recorded and used to compute the performance. In our implementation, we collect the cost of the best mapping found, the amount of CPU time used by post-game analysis, the number of post-game iterations, the number of moves done, and the amount of improvements of the best mapping over the initial mapping.

The evaluator evaluates each candidate one *quantum* at a time. Depending on the size of the quantum, one or more problem instances can be solved with respect to a number of initial random mappings. With a small quantum size, more candidates can be examined; however, the assessment of the performance of the candidate tested at the end of a quantum may be difficult. With a large quantum size, more accurate assessment can be made on the candidate tested; however, the maximum number of quanta is reduced, as the total learning time is fixed. Tradeoffs on the quantum sizes are discussed in Section 6.

5.4. Candidate Generator

The generation of new post-game heuristics is a difficult problem. First, a heuristic is composed of both numeric and symbolic parts. It is much more difficult to manipulate symbolic rules than numeric values. In addition, the set of possible heuristics is ill-defined. It is possible that the current set of heuristics, which was designed manually, might be missing

some important and unknown components. It would require a substantial amount of domain knowledge to discover them in an automated fashion. At present, we only consider the existing heuristics and transform them into new ones by a few well-defined operators.

Due to the limited amount of time available for learning, the candidate generator should utilize the performance of previous candidates in generating new ones. Currently, we have investigated two methods: random and greedy. In a random approach, a new candidate is generated by a random perturbation from the current candidate. In a greedy one, the generator tries to follow the direction of the greatest improvement in performance, based on performance of candidates already generated. We plan to explore other methods of candidate generation in the future.

Our current implementation is based on a rule-based system. It is flexible enough to allow additional domain knowledge, new operators, and new rules to be added incrementally. It consists of a set of assertions called the *working space* and a set of rules. A rule consists of two parts: a list of conditions and a list of actions. When all conditions within a rule are met, the rule fires, and all actions within the action list are attempted. A condition can be either a pattern or a function. A pattern is considered met by finding a unification between the pattern and an assertion within the working space, while a function is considered met when it returns a non-zero value. Variable assignments and wildcards can be used in patterns and functions. There are three types of actions: delete an existing assertion used in the condition part from the working space, add a new assertion to the working space, and execute some functions. Actions can use values of variables assigned in the condition part of the rule.

There are presently four operators for transforming a candidate, *i.e.*, moving it from one point to another point in the heuristics space. Three operators apply to proposal generation heuristics: change the threshold of the conditional expression, remove an existing heuristic, and duplicate an existing heuristic with a different threshold value. The other operator selects each priority-assessment expression from one of the six possible combinations. Each operation can be specified as (OP, H, δ), where OP is the operator, H is the heuristic subject to the operator, and δ is the parameter set of the operator. We plan to add additional operators in the future and include probabilistic uncertainties in the feasibility heuristics.

A transformation, $T(C_i, C_j)$, is a sequence of operations $\{(OP_k, H_k, \delta_k)\}$, $k=1, \cdots$, that transform a point in the heuristics space, C_i, into another point, C_j. In other words, candidate C_j is created by applying transformation $T(C_i, C_j)$ to candidate C_i. Two transformations, $\{(OP_k^1, H_k^1, \delta_k^1)\}$ and $\{(OP_k^2, H_k^2, \delta_k^2)\}$, are said to be *similar* if $OP_k^1 = OP_k^2$, for all k. If, in addition to this, H_k^1 is also the same as H_k^2 for all k, then the two transformations are said to be *in the same direction*.

Let $Relate(C)$ be the set of all candidates C_i such that the distance between C and C_i in the heuristics space, *i.e.*, the number of operations to transform C into C_i, is less than some predefined constant d_{min}. The operations carried out in the meta-rules for candidate generation can be divided into three stages. They are described briefly as follow.

(1) Given a candidate as a starting point, find C_{inc}, the best candidate that a new candidate can be created from.

Table 6.1a. Test of Normal Distribution with t_{max} = 600 seconds. A total of 100 candidates, each with 20 points, were tested. At α = 0.2, Kolmogorov-Smirnov test with 20 samples requires the value to be less than 0.23, while Geary Test at α = 0.05 requires the value to be greater than 0.915 [4].

Test	Min	Avg	Max	# Fail/Tot
Kolmogorov-Smirnov ($\alpha = 0.2$, KS < 0.23)	0.079	0.142	0.252	1/100
Geary Test ($\alpha = 0.05$, G > 0.915)	0.784	0.943	1.049	27/100

Table 6.1b. Test of Normal Distribution with t_{max} = 180 seconds.

Test	Min	Avg	Max	# Fail/Tot
Kolmogorov-Smirnov ($\alpha = 0.2$, KS < 0.23)	0.051	0.136	0.220	0/100
Geary Test ($\alpha = 0.05$, G > 0.915)	0.840	0.977	1.106	15/100

- Let C_{max} be the candidate with the highest difference in performance from C_{inc} among all candidates in the set defined by $Relate(C_{inc})$.
- If C_{max} has better performance than C_{inc}, then use C_{max} as C_{inc}. As a result, C_{max} after this step will have equal or worse performance as C_{inc}.
- If no candidate in $Relate(C_{inc})$ has performance among the top 2/3 of all candidates, then use C_{best}, the incumbent candidate with the highest performance, as C_{inc}.

(2) Find transformation with the highest performance increase.

- Base on C_{max} found in step (1), find $T_{new} = T(C_{max}, C_{inc})$, the transformation that causes the greatest increase in performance within the heuristics space around C_{inc}.

(3) Generate the new candidate by applying one of the following cases.

- If there are no existing candidates *in the same direction* as T_{new}, then $C_{new} = Apply(T_{new}, C_{inc})$. Finish.
- If there are very few points in the neighborhood of C_{inc}, then $C_{new} = Apply$(Random Operations, C_{inc}). Finish.
- If there are many points in the neighborhood of C_{inc}, then find T *similar* to T_{new} such that there are no candidates *in the same direction* as T, and set $C_{new} = Apply(T, C_{inc})$. Finish.

6. EXPERIMENTAL RESULTS

To verify our method for learning heuristics for post-game analysis, we train the post-game system using a target problem based on a divide-and-conquer paradigm. In this problem, each node does some computation for a random amount of time and sends a message to each of its child nodes to start computation. It then waits until it receives results from its descendents before reporting to its parent. There are a total of 105 processes that are mapped to a 3-by-3 mesh architecture. 20 problem

Table 6.2. Result of Learning for Different t_{max}. Time allowed for learning is 400 quanta for the first stage and 133 quanta for the second stage. (Total time = 533).

t_{max} (sec)	Orig. Perf.	Performance (1300*$Q(L)$)	Average Post-Game Exec. Time (orig=62.30)	Average Mapping Prob. Comp. Time (orig=1595.57)
20	0.1425	0.6965	14.81	1880.75
30	0.3066	0.7074	24.99	1636.07
40	0.4291	0.7722	24.99	1636.07
50	0.4962	0.7881	24.99	1636.07
60	0.5479	0.7913	24.99	1636.07
70	0.6039	0.7955	32.38	1622.93
90	0.6900	0.8061	37.17	1595.16
120	0.7356	0.8137	37.17	1595.16
180	0.7888	0.8176 (best)	37.17	1595.16
		0.8141 (3)	40.38	1595.72
300	0.8087	0.8194 (best)	57.75	1587.90
		0.8183 (3)	42.09	1594.94
450	0.8140	0.8235 (best)	96.12	1567.05
		0.8209 (3)	57.75	1587.90
600	0.8167	0.8290	96.12	1567.05
900	0.8182	0.8317	96.12	1567.05
∞	0.8182	0.8332 (best)	96.12	1567.05
		0.8195 (14)	59.51	1592.65

Note 1. If our learning framework does not find the actual best candidate for a given t_{max}, then the best candidate is shown in first row, and the actual result on the second row with the rank of the candidate indicated in parenthesis.

Note 2. The execution time taken by post-game analysis averaged over all problem instances tested using the original heuristics of Yan and Lundstrom [14] is 62.30 secs. The average completion time of the mappings found by the original heuristics is 1595.57 secs.

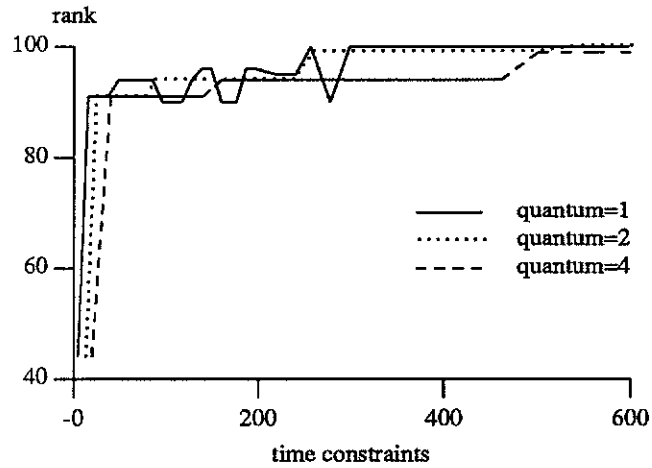

Figure 6.1. Performance of heuristics learned as a function of quantum size and learning time allowed.

Table 6.3. Result of applying heuristics learning to a distributed blackboard program. Each candidate in the table corresponds to the candidate with the best performance for a particular value of t_{max}. The corresponding t_{max} is listed in the first column.

Candidate (t_{max})	Average Post-Game Exec. Time	Average Mapping Problem Comp. Time
original	1056.17	29655.22
t_{max} = 20	60.77	35718.67
t_{max} = 30,40,50,60	113.57	31063.76
t_{max} = 70	594.17	30083.52
t_{max} = 90,120,180	1254.08	29279.45
t_{max} = 300	182.56	30454.57
t_{max} = 450,600,900,∞	1310.04	30056.83

instances with CPU times drawn from the same distribution are included in the test set.

The 20 problem instances are evaluated for 100 different candidate heuristics. Each set of 20 points is then tested for normality using both the Kolmogorov-Smirnov Test and the Geary Test [4]. The results summarized in Table 6.1 demonstrate the validity of the assumption.

The time limit for the first learning experiment is 533 time units, where each time unit corresponds to the time for evaluating a candidate over one problem instance with 31 initial random mappings. We choose one quantum of evaluation to be equivalent to one time unit. Our system use the 2-stage selection method discussed in Section 4. The first stage uses 75% of the total time and tests as many candidates as possible such that each candidate is tested in four time quanta. The second stage uses the minimum-risk-selection method to evaluate the top 10% of the candidates from the first stage until time runs out. Using this strategy, 100 candidates can be tested. In contrast, if each candidate were evaluated in 20 time quanta, then only 26 candidates can be test in the time limit. The initial candidate pool consists of 24 predefined heuristic sets. The candidate generator creates new candidates as old ones are tested.

The result in Table 6.2 shows the tradeoffs between the performance of the target mapping problem and the execution time of the heuristic candidate. Recall that t_{max} is the threshold beyond which a linear cost is associated with the mapping cost ($c_r = 1$ in Eq. (5.1)). We apply our learning system to learn new heuristics using the objective of quality-cost ratio defined in Eq. (5.2) for each given t_{max}. We also exhaustively evaluate the 100 candidates and rank them. The results in Table 6.2 show that in most cases our selection method finds the best candidate. We have also indicated the rank of the candidate found in case we did not find the best one. Using the quality-cost measure, our system consistently produces better heuristics than than the original heuristics of Yan and Lundstrom [14].

Table 6.2 also shows for each t_{max} the time required by post-game analysis to find the mapping and the resulting completion time of the target problem. These results indicate that there is a tradeoff between these two quantities. It is important to choose the heuristic candidate that best fits the application requirements. An automated system for learning heuristics is

essential here, as it shows alternatives to users and allows them to make a better choice.

A second experiment was performed to measure the effects of varying the quantum size and the time limit. Figure 6.1 depicts the results for various learning times and 3 quantum sizes. The graph shows the trend of increasing performance of candidates selected at the end of learning period when the duration of learning increases. This is true for all quantum sizes. The fluctuation in performance decreases as the quantum size increases because more tests can be performed in a larger quantum and there is less chance for error. However, the performance for the same learning duration decreases as the quantum size increase because less candidates can be tested.

A third experiment was carried out to demonstrate the generality of the heuristics learned. We apply the candidates that provide the best performance for various t_{max}'s to a problem instance of mapping 115 processes in a distributed blackboard program on a 3-by-3 mesh architecture. This program has a very different behavior from the divide-and-conquer program used to learn the heuristics. The performances over 31 random initial mappings are shown in Table 6.3.

Results in Table 6.3 show that the general trend of candidate performance follows the same pattern as the performance of the original test database. The only exceptions are the candidate for $t_{max} = 300$, and for $t_{max} = 450$, 600, 900, and ∞. From these limited experiments, generalization of learning results seems possible at least to a limited extent. Further experimentation on generalization will be carried out in the future.

7. FUTURE WORK

Future work on this research include (a) the application of our learning system to a variety of target mapping problems, (b) the refinement of meta-rules in the candidate generator, (c) the normalization of results when test cases are not identically distributed, (d) studying the effect of using different heuristics under different resource conditions, (e) the dynamic learning of heuristics during run time based on past profile, and (f) the detection of conditions in which performance is not improved with increased learning and the methods to cope with them.

REFERENCES

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

[2] J. O. Berger and J. Deely, "A Bayesian Approach to Ranking and Selection of Related Means With Alternatives to Analysis-of-Variance Methodology," *J. of the American Statistical Association*, vol. 83, no. 402, pp. 364-373, American Statistical Association, June 1988.

[3] S. H. Bokhari, "On the Mapping Problem," *Trans. on Computers*, vol. C-30, no. 3, pp. 207-214, IEEE, Mar. 1981.

[4] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*, Brooks/Cole Publishing Company, Monterey, CA, 1982.

[5] E. J. Dudewicz and J. O. Koo, *The Complete Categorized Guide to Statistical Selection and Ranking Procedures*, American Sciences Press, Inc., Columbus, OH, 1982.

[6] V. M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," *Trans. on Computers*, vol. C-37, no. 11, pp. 1384-1397, IEEE, Nov. 1988.

[7] M. B. Lowrie and B. W. Wah, "Learning Heuristic Functions for Numeric Optimization Problems," *Proc. Computer Software and Applications Conf.*, pp. 443-450, IEEE, Chicago, IL, Oct. 1988.

[8] R. S. Michalski, "Understanding the Nature of Learning: Issues and Research Directions," in *Machine Learning: An Artificial Intelligence Approach*, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, vol. II, Morgan Kaufmann, Los Altos, CA, 1986.

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

[10] K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *Trans. on Computers*, vol. C-38, no. 8, pp. 1110-1123, IEEE, Aug. 1989.

[11] S. R. Schwartz, *Resource Constrained Parameter Tuning Applied to Stereo Vision*, M.Sc. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, August 1991.

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

[13] J. C. Yan, *Post-Game Analysis--A Heuristic Resource Management Framework for Concurrent Systems*, Ph.D. Dissertation, Dept. Elec. Eng., Stanford Univ., Dec. 1988.

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

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