From: Proceedings of the Twelfth Annual International Computer Software & Applications Conference, pg. 443-450

LEARNING HEURISTIC FUNCTIONS FOR NUMERIC OPTIMIZATION PROBLEMS

Matthew Lowrie and Benjamin Wah
Coordinated Science Laboratory
University of Illinois
1101 West Springfield Avenue
Urbana, IL 61801

ABSTRACT

Numeric optimization problems have arisen in a wide variety of fields, from economics to physics. Learning strategies are becoming more practical and can be seen as a strong method for increasing the efficiency of systems as well as acquiring new concepts and relations. In this paper, learning techniques are applied to enhance the efficiency of computer solution to numeric optimization problems. The goal of the paper is to provide a framework for approaching learning of heuristic functions for numeric optimization problem solutions. The outline of a system, Techniques using Experimentation for Acquisition and Creation of HEuRistics 2.0 (TEACHER 2.0), for learning heuristic functions is presented. The system is unique in that it combines many learning techniques into one coherent system. It can be a powerful learning system as it allows generation of heuristics based on an amalgamation of learning techniques and strategies. The value of the system is illustrated by an example where TEACHER 2.0 learns a new heuristic which is superior to the typical heuristic for that problem domain.

KEYWORDS AND PHRASES: Branch and bound, frames, guidance function, heuristic searches, knapsack problem, learning methods, numeric optimization, transformation function.

I. INTRODUCTION

Since the invention of computers, the applications where they can be applied have expanded rapidly. This can be seen to be the result of both the expanding capabilities of computers, and better understanding of programs along with continual improvement of software development environments. In the future, computers may continue to expand their processing abilities, and the size as well as class of solvable problems should continue to grow. A powerful technique to promulgate this expansion employs machine learning. A system capable of learning and adjusting to more efficiently solve problems has a clear advantage over a system that does not learn [21].

In this paper, the design of learning systems to enhance processing efficiency are considered. The specific form of processing discussed is efficient search algorithms. The class of problems, where the method for design of learning systems is applied, is numeric optimization problems. These problems continue to be an important application for computer solution and arise in a great diversity of fields, from economics and business to physics and computer science. Automated techniques for acquiring knowledge which makes this form of processing more efficient can be extremely valuable. Learning of dominance relations has been explored in earlier work on the TEACHER 1.0 system [25]. This research, TEACHER version 2.0, represents design of a learning system applied to learning other heuristic functions.

The approach employed in this paper to developing a machine learning system is outlined in Figure 1. This approach to learning is specific to the development of systems for increasing the efficiency of computation algorithms. The way this strategy is mapped into the presentation of the paper is as follows. In order to identify the specific goals of the learning system, the issues involved in solving numeric optimization problems are presented in Section II.A. The motivation for using numeric optimization as a testing ground for learning heuristics is given in Section II.B. The final step in defining the learning problem is identification of system objectives. This is done in Section II.C. The ultimate goals of the learning system include enhancement of the efficiency of processing and an increase in the size of problem solvable on existing computers.

Prior to developing the framework of the learning system, it is highly desirable to identify existing learning techniques and strategies. This identification includes analysis of their potential uses in the learning system. This is the topic of Section III. The emphasis of this section is not on reviewing well known learning techniques, but on analyzing the utility of existing learning techniques for numeric optimization problems, and on developing an alternative philosophy that enables the development of a learning system that does not rely on only one learning technique.

Once the strategies and techniques have been discussed and evaluated, the design of a system for performing this task is presented. Section IV begins with a discussion of problem representation, and follows a strategy for development of a complete learning system. The system is unique due to its utilization of many different learning strategies and paradigms. The potential value of TEACHER 2.0 is demonstrated by its application to the 0/1 knapsack problem. TEACHER 2.0 is shown to be capable of discovering new heuristics that are superior to the heuristic usually employed for that problem. The new heuristics enable solution of problems 20% larger (20% more variables), for a given amount of computer resources.

II. DEFINITION OF THE LEARNING PROBLEM

In this section, the domain and goals of the learning system are developed. The end-goal of the learning system is to enhance the efficiency of searches, and extend the size of solvable problems which use search techniques. This statement of a goal is not sufficient to begin designing a learning system as it says nothing about the problem domain, characteristics, and how search efficiency may be increased. This section, therefore, is devoted to analysis of how this end-goal may be approached.

II.A. Search Solutions to Numeric Optimization Problems

Numeric optimization problems may be represented as integer programming problems. A linear integer programming problem may be written in the form: Maximize: $\bar{c}^T \cdot \bar{x}$; Subject to: $A \cdot \bar{x} = \bar{b}$, $\bar{x} \geq 0$ and integer, where $\bar{x}$ denotes a vector x, and $\bar{Z}$ represents a two-dimensional

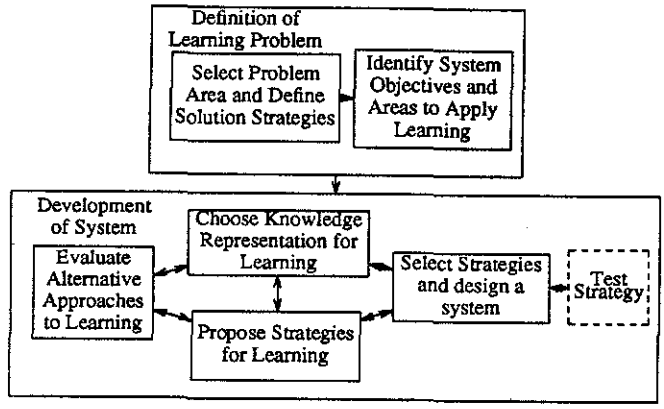

Figure 1. Approach to Design of Learning System

Research was partially supported by National Aeronautics and Space Administration Grant NCC 2-481 and National Science Foundation Grant MIPS 85-19649. Research of M. B. Lowrie was also supported by a Ph.D. Scholarship from AT&T Bell Laboratories.

Computer Software and Applications Conference, IEEE, October 1988.

matrix Z. Nonlinear integer programming problems have a similar representation, but have objective and/or constraint equations which are not linear. The techniques in this paper are discussed in terms of linear numeric optimization problems.

There are many ways to solve integer programming problems. The basis for these solutions may employ enumeration, Bender's Decomposition, cutting planes, or group theory [5]. In this paper, emphasis is placed on search techniques, which are based on the concept of enumeration. In the search, the intial problem is transformed by a transformation function, or branching rule, into subproblems. Subproblems are branched on until a solution is found.

In the design of a heuristic search, the heuristic seeks to exploit the characteristics of the problem domain. Consequently, the greater the restrictions in a class of problems for which heuristics are sought, the greater potential for finding effective heuristics. Such numeric optimization problems include the cargo-loading problem, the 0/1 knapsack problem (a restricted version of cargo-loading), numerous scheduling problems, inventory problems, etc.

For discussion purposes, only one of these problems will be referred to: the 0/1 knapsack problem (0/1KP). The 0/1KP is defined as: Maximize: $\vec{v}^T \cdot \vec{x}$, subject to: $\sum_{i \in U} w_i \cdot x_i \leq K$, and, $x_i = 0,1$. In this representation, x_i is 1 if item i should be placed in the knapsack, and 0 otherwise.

A technique for solving optimization or semioptimization search problems is branch and bound (B&B) search [9,16]. Search strategies such as A*, AO*, SSS*, B*, alpha-beta, and dynamic programming are specific instances of the general form of B&B algorithms [9]. B&B algorithms consist of the *transformation function* defined above, a technique for *selecting* the next node for expansion (guidance function), and a *pruning mechanism*. The original pruning functions involved only upper and lower bound functions. The lower bound function is the best value of a solution known to be achievable. Usually, the lower bound is just a known solution. An upper bound function is a value which the value of the best solution achievable from that node cannot be better than. Nodes which have upper bounds less than a known lower bound may be pruned. Pruning functions may also take the form of dominance relations.

II.B. Numeric Optimization as a Problem Domain

Numeric optimization problems are a natural research environment for studying learning strategies. The techniques currently employed are well researched, and a good deal of experience has been gained. The formal nature of mathematical areas of research creates the advantage of easy, or at least provable and formal, evaluation of the results of the system. The relations employed by a numeric optimization problem solving system are comprised of mathematical functions. These functions are also well understood, "well-behaved" and natural for implementation on a digital computer. While research in this area is natural, advances in learning and search techniques may still be applicable to less formal domains.

Mathematical topics had received such great attention by researchers since the time of Pythagoras. The attention of many researchers not only indicates the importance of the field, but lends a formalism to the environment which is advantageous. In addition, learning new concepts in such an environment will clearly demonstrate the value of a learning system -- the ability to add knowledge to the knowledge readily achievable to man alone.

The learning technique employed by a computer, at least in this case, utilizes the strengths of a computer which man lacks. These strengths include 100% accurate memory, and the ability to evaluate potentially millions of expressions per second. Although man is (and may always be) the greatest thinker, some of the advantages of his intuition can be countered in this way. Since the computer will rely on different strengths, it is natural to hope that the computer will "discover" strategies which man has not found by his superior intuition.

In the design of a heuristic search, the heuristic seeks to exploit the characteristics of the problem domain. Consequently, the greater the restrictions in a class of problems for which heuristics are sought, the greater potential for finding effective heuristics. For this reason, classes of numeric optimization problems which have restricted domains are sought. Such problems include the cargo-loading (also referred to as resource allocation problems), the 0/1 knapsack problem (which is a restricted version of the cargo-loading problem), the scheduling problem, the inventory problem, etc.

II.C. System Objectives

In this section, the objectives of the research are more formally discussed. The end-goal of the system is enhancement of performance of search solutions. The domain for which this task is explored encompasses numeric optimization problems. The intermediate goals, and technique for measuring the performance of the search is discussed first. This is followed by an analysis of how these goals may be approached. This Section is not a discussion of how the system will perform learning. Instead, it is a discussion of the goals and types of knowledge the system will try to acquire in order to achieve those goals.

Measuring Search Performance. On a real computer system, the performance of a search can be identified with the time necessary for completion of a problem. The learning system may not, in general, be performed on the same system as the solution of the problem is to be performed. In addition, it may be desirable to acquire knowledge that will promote efficiency on many computer systems.

The usual and most obvious measure for determining the efficiency of a search is the number of nodes expanded by the search. In fact, search complexity is usually measured as the relation of the number of nodes expanded to the problem size. The number of nodes expanded by a search technique is certainly a strong measure of search performance. Each expanded node requires processor time for expansion, and must at some point be stored in memory.

It has been shown that a bound function must be quite accurate to avoid exponential number of nodes expanded, on the average. Analytical results have been obtained for when this time is exponential and when it is polynomial, on the basis of the accuracy of the heuristic bound function [18].

Unfortunately, number of nodes expanded is not the only relevant indicator of search performance. The communication behavior is a factor that also greatly impacts search performance. In searches implemented on a single processor system with hierarchical memory, memory behavior is the facet of communication behavior that affects performance. In a best-first search, all active nodes must be stored in immediate memory since any of them may be selected as the next node to be expanded. In a depth-first search, this memory requirement is not as strict since memory access is much more local. Depth-first searches require only linear amount of memory. As a result, depth-first search can perform some searches in less time even though best-first search expands fewer nodes [22]. In parallel search, communication of nodes and pruning information is the constraint on performance [23].

Another standard by which a heuristic may be measured concerns the complexity of the processing at each node. The more complex a heuristic is, the greater knowledge it should possess. The greater knowledge should be reflected by fewer nodes requiring expansion in the search [8]. Thus, very knowledgeable heuristics are desirable. More complex heuristics, however, may require greater resources. Thus, a simpler heuristic may be able to investigate more potential solutions in a lesser amount of time [1].

It would be desirable to be able to evaluate heuristics directly in terms of these goals. The only way to do this is to try the heuristic on a large number of problem instances of the type *and size* of the end application of the heuristic. Let N denote the problem size. Typically, N represents the number of variables in the problem, e.g. the number of items in the 0/1KP. Solution of problems with large N may require too much computation time to allow the use of this technique to evaluate the large number of potential heuristics. This is especially significant in the case of heuristics that do not effectively trim the number of nodes evaluated. One alternative is to use the heuristic being evaluated on random problems, and measuring how many of these problems it can solve within a certain constraint such as number of nodes expanded. Another alternative is to measure the performance of the heuristic on a large number of problems with smaller N. The final alternative is to develop measures which allow evaluation of the heuristic without performing complete searches. An example of this would be measuring the mean error of a bound function on a set of sample nodes. The problem of evaluating heuristics is considered in greater detail in Section IV.

Learning Targets: The system explores three types of heuristics: bound function, guidance function, and transformation function. Each of these functions has a strong effect on the performance of a search. In general, search performance is a function of all three of these functions. Learning of these three functions is by no means the only way to enhance

Table 1. Targets for Learning System.

Function Type	Usual Form	Possible Parameters	Description and Comments
Bounding Function	Mathematical Expression	Node Description	Composed of an expression or algorithm for computation. May be composed of +, −, *, /, max, min, $\sum$, $\prod$, $\cap$, $\cup$, etc. Since heuristic is appropriate for all n (size of problem), operations are usually on sets of elements such as $\sum$, and $\cap$. Is subject to monotone and admissibility restrictions.
Guidance Function	Mathematical Expression	Node Description, Search History, Penalty Functions	Same as bounding function, but without monotone and admissibility restrictions. Non set operations are more common when incoporating penalty functions.
Transformation Function	i such that: <constraint> & <constraint> ...	Node Description, Search History, Penalty Functions	Is restricted to an integer value or subset of variables to branch on. May be modeled as $f(g_1(Node), g_2(Node), ...)$, where f selects a g_i (such as max or min), and g sub i (Node) represents an evaluation of the value of using a transformation: i (such as branching on variable i in the 0/1KP).

search performance. Learning of dominance relations is another technique that was explored by the TEACHER 1.0 system [25].

The quality of both the heuristic bound function and the heuristic guidance function greatly affect the performance of search solutions to numeric optimization problems. The bound function determines the pruning power of B&B searches. The guidance function determines the behavior and efficiency of the search when it is implemented on a real computer system, as well as influencing the number of nodes expanded in search strategies containing a depth-first component. The transformation function also strongly influences search behavior and the number of nodes expanded. In general, search performance is a function of these three search parameters.

A learning system should take advantage of the form that the function takes. The forms of these three functions are demonstrated in Table 1. In guidance and transformation functions, search history and penalty functions incorporated *as parameters* of the function have not been explored in detail in the literature, but are promising areas for the learning system to explore.

III. LEARNING HEURISTIC FUNCTIONS

In this Section alternative strategies for approaching the problem of learning heuristic functions for numeric optimization problems are discussed. Once the available strategies and properties of heuristics have been detailed, an evaluation of the potential uses of these techniques in a learning system may be performed. In Section III.B. an alternative approach to learning is developed. The approach is valuable because it enables development of a system that takes advantage of many learning techniques.

Table 2. Application of General Learning Techniques.

Learning Technique	Example	Use
Instruction	TIERE-SIAS NANO-KLAUS	Basis of most known heuristics; too restrictive of a paradigm for learning known heuristics; incorporation of facility to take advice is desirable.
Deduction	PLANNER FOO	Requires unit to propose alternatives for verification; useful for verification of bounding functions.
Analogy	CARL	Only useful for problems with similar base; cannot derive functions based on unknown concepts;
Induction/ Example	SPARC ARCH LEX	Restricts learner to a narrow focus; not a preferred strategy in general; neural networks can be applied for learning guidance functions.
Induction/ Clustering	Cluster	Data is already clustered, a way of describing the cluster is sought; additionally, obtaining attributes for clustering is difficult.
Induction/ Quant. Anl.	BACON ABACUS	Useful for deriving relationships between attribute and real values.
Induction/ Theory Form	AM EURISKO	Useful for deriving attributes and relations.
Induction/ Incremental	INC. AQ ARCH	System should base decisions on a large set of search data; incremental learning technique is preferred.

III.A. Learning Techniques Applied to Numeric Optimization Heuristics

Learning techniques in the past have been categorized as being based on: rote, instruction, deduction, analogy, and induction. Induction is further categorized on the basis of induction type [2, 13, 14]. In Table 2, the techniques, some example systems using the technique, and their potential application are enumerated.

Techniques specific to learning heuristic functions include: analogy [4], constraint alteration [6, 18], heuristic rules [10, 11], probabilistic estimation [19], systematic exploration [24], genetic learning [20], and explanation-based [15]. Constraint alteration is a powerful technique which formulates heuristics by simplifying the problem. This is done by either adding or removing constraints. Heuristic rules guide the search for a heuristic with if <cond> then <do> rules. The uses of these learning techniques are summarized in Table 3.

III.B. An Alternative Approach to Learning Heuristics

Learning strategies in the past have focused on using one technique for learning heuristics. An example is the "use" of constraint relaxation to derive linear programming as a heuristic for integer programming. A set of inputs, such as the problem description and analogous problems, are input. A single learning technique, such as analogy or constraint relaxation, is employed. The result is a potential heuristic.

An *attribute* is defined as a formula or relation which describes a problem instance. For example, linear programming is an *attribute* for the integer programming problem. The goal of the system, then, is to generate attributes which effectively perform a heuristic function, such as a guidance function. The approach to learning developed in this paper is depicted in Figure 2. The environment inputs the problem description, any possibly analogous problems, known heuristics, etc. The learning system then uses multiple learning techniques to derive attributes. These attributes enter a pool, which are also potential heuristics. Attributes may be combined with the environmental inputs or other attributes and used as inputs to multiple learning techniques which generate new attributes. Intuitively, this technique is very strong. The results of different learning strategies can be combined using a third learning strategy. In general, attributes are potential heuristics which may be derived using an amalgamation of many learning techniques. Examples of learning which use this approach will be presented in Section IV.

Table 3. Application of Heuristic-Specific Learning Strategies.

Learning Technique	Use
Constraint Alteration	A useful technique for learning most types of heuristics. Lagrange relaxation is not as appropriate for bounding functions due to admissibility restrictions.
Heuristic Rules	Useful technique. Programmed heuristic rules may limit the types of functions the system can learn, so other techniques should be incorporated.
Probability Estimation	Derivation of probability on basis of representation may be difficult; can be useful for evaluating heuristics.
Systematic Exploration	Useful technique based on learning by induction; Derivation of terms and attributes is difficult, since there may be an infinite number of ways to derive them.

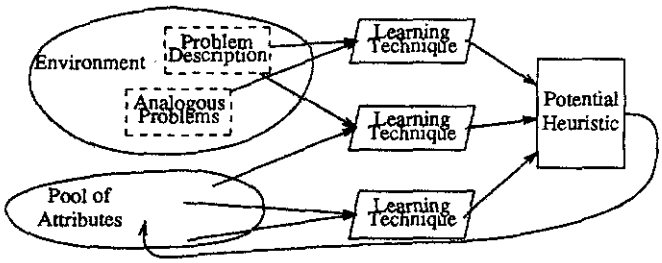

Figure 2. Approach to Learning Heuristics.

IV. TEACHER 2.0: A System for Learning Heuristic Functions

TEACHER 2.0 is a framework for construction of learning systems. The targets of a TEACHER 2.0 system are numeric heuristic functions to increase efficiency of search solutions to numeric optimization problems. The presentation of TEACHER 2.0 is organized in the following way. First, the overall learning strategy employed is reviewed in Section IV.A. The knowledge representations for learning employed in TEACHER 2.0 are then discussed in Section IV.B. The system organization is then presented in Section IV.C. Finally, an example of how the system performs learning is given in Section IV.D.

IV.A. Learning Strategy

The learning of effective heuristics is seen to be a very difficult topic. As a result, the most powerful learning system possible is desired. For this reason, the TEACHER 2.0 system seeks to utilize all of the available learning strategies, where they are most effective. The philosophy of design for TEACHER 2.0 was outlined in Section III.B.

In the TEACHER 2.0 system, a hierarchy of learning tasks is discerned. This hierarchy is composed of three levels: attribute generation, attribute management, and evaluation. The *attribute generation level* is composed of the learning techniques that generate potential heuristic functions. The *attribute management level* is concerned with managing the pool of attributes as well as the scheduling of attribute generation. Finally, the *evaluation level* compares and computes the efficiency and performance of the attributes. In order to evaluate potential heuristic functions, the system actively generates sample instances of the problem.

Each type of heuristic may require different techniques for learning. For example, the form of a bound and transformation function may differ. The system should utilize this during its learning process. The forms of the three types of heuristic functions were detailed in Table 1.

IV.B. Knowledge Representations

There are three types of knowledge representations the system must be concerned with: representation of the domain problem (e.g. 0/1KP), representation of sample searches generated by the system, and representation of the search space of potential heuristic functions. Each of these will be considered separately.

Representation of Domain Problem: A useful representation for learning problems in the mathematical domain is frame representations. Frame representations were used in the AM and EURISKO systems [11,12]. More significantly, frames were utilized in TEACHER 1.0 for numeric optimization problems [24,25]. Many of the numeric optimization problems had similar representation in a frame environment. For example, the 0/1KP, scheduling, and inventory problems can be represented efficiently in frames, as depicted in Table 4.

The slots in Table 4 are sufficient to describe the problem and the search space. An additional slot should be allocated providing a procedure for expansion of a node. Since many of the problems that the system might work on will have heuristics, additional slots could be allocated for known heuristics. Since the system will be generating sample search trees of the problem, it might also be desirable to provide range and random generation functions for generation of problem instances.

Representation of Sample Searches: Sample searches are used in TEACHER 2.0 for evaluation of candidate heuristics. The system actively generates example problems. The frame representation can also be used for representation of sample searches [24,25]. As in Table 4, slots can be allocated to variables, except that values for the variables would be included. An additional slot can be included for the nodes of the search. Each node of the search would have slots describing that node of the search.

Table 4. Frame-Like Representations for Numeric Optimization Problems.

Frame Slots	Problems		
Name	0/1 Knapsack	Scheduling	Inventory
Variables	x_i=0,1: 0: i not in sack 1: i in sack	ρ_t: location of task t in schedule	x_i, y_i: amount bought, sold in period i
Properties	value: v_i weight: w_i	Execution Time: γ_t Deadline: d_t Late Penalty: w_t	Sale Price: s_i Buy Price: b_i
Constants	Capacity: K # of Items: n	# of Tasks: n	Init. Invent.: v # of Periods: n
Objective	Maximize $\sum_{i=1}^{n} x_i v_i$	Minimize $\sum_{t=1}^{n} (\rho_t+\gamma_t) w_t$	Maximize $\sum_{i=1}^{n} x_i s_i - y_i b_i$
Constraints	$\sum_{x_i \in U'} w_i \le K$	all $t \in T$: $\rho_t+\gamma_t \le d_t$	$v+\sum_{i=1}^{j} x_i - y_i \le B$, $y_j \le v+\sum_{i=1}^{j-1} x_j - y_j$ all $i,j \in (1...n)$

The technique used for evaluation of an attribute depends on the type of heuristic the system is endeavoring to learn. Examples of these will be discussed specifically in Section IV.

Representation of Search Space of Heuristics: As in Figure 2, each term can be considered to be an element in a pool of attributes. Each attribute is stored as a record. An example of an Attribute Record for a transformation function for the 0/1 KP would be as follows.

Attribute Record	
Name	ATT2
Form	$\max_i: v_i/w_i$
Evaluation	(Evaluation Tuple)
Used In Attributes	ATT4, ATT6
Contains Attributes	ATT1, ATT8

This record is just an example of some of the fields that may be used. Additional fields may be incorporated in a TEACHER 2.0 system to assist scheduling of the search and other operations (see Section IV.D).

The pool of attributes is a tree-like data structure that classifies the attributes on basis of its evaluation tuple. The evaluation tuple is based on the type of heuristic being learned, and the relative importance of the objectives of the learning system.

IV.C. System Overview

The techniques for learning and representing knowledge and data in TEACHER 2.0 have been presented. In this section, an overview to the way that these techniques are mapped into a software architecture is provided. The relation of the major components of TEACHER 2.0 are detailed in Figure 3. The functional description of the components follows. The Attribute Generators are the primary learning elements in the TEACHER 2.0 system. Each Generator systematically explores possible heuristics, in the order determined by the Manager. The Manager determines this order heuristically. Each attribute is considered to be a potential heuristic. Each

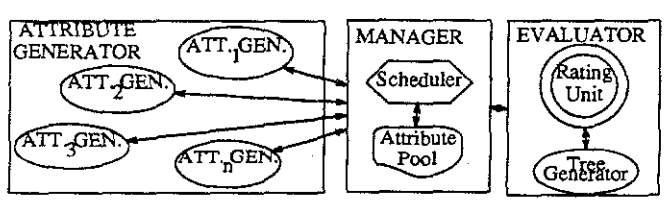

Figure 3. TEACHER 2.0 Software Architecture

Table 5. Example Attribute Generators.

Attribute Generator	May Be Used For	Comments
Advice Taking	All*	Allows user to input known heuristics, or functions which the user would like explored; merely forwards user input to the Manager.
Combinator	All	Generates *secondary* attributes only; combines attributes on the basis of the evaluation field of the attribute records in order to enhance the measure(s) of the evaluation field.
Decomposition	All	Generates *secondary* attributes only; decomposes attributes into their subcomponents; also proposes terms in problem representation as attributes.
Permutor	All	Generates *secondary* attributes only; takes as input a single attribute and modifies it to generate a new attribute; for example, changing v_i+w_i to $v_i^2+w_i$.
Analogy	All	Performs two functions; proposes attributes on the basis of analogous problems; proposes attributes on the basis of attributes discovered by other (or this) Generator.
Relaxation	Bound Guidance	Proposes attributes which are discovered through relaxation of some constraint; relaxation techniques suffer the disadvantage of generating 'relaxed' problems which may not be solvable; generates attributes on a greedy algorithm way; for example 0/1KP guidance function, selection of consecutive items which maximize (v_i/w_i), until the capacity constraint is violated.
Penalty	Guidance Transformation	Proposes penalty terms to be associated with erratic memory behavior, such as backtracking; for example, use of heuristic rules to generate penalty for selection of nondepth-first node.
History	Guidance Transformation	Proposes attributes which are based on changes which occur or accumulate from parent to child nodes; has not been explored in the use of guidance functions; TEACHER 2.0 presents a new avenue for exploring guidance functions which incorporate search history; an example is the change in bound function from parent to child.

*Relevant to learning of bound, guidance, and transformation functions

attribute is evaluated by the Evaluator and given a set of scores on its performance.

ATTRIBUTE GENERATORS: The Attribute Generators are responsible for derivation of attributes on the basis of the problem description frame. It should not be restricted to any one technique for learning. Analogy, deduction, instruction, relaxation techniques, and other schemes may all be used as Attribute Generators. Some example Attribute Generators are presented in Table 5.

Table 5 provides descriptions of some *techniques* for learning attributes. The fact that some of these techniques are appropriate for learning more than one type of heuristic function does not mean that the same generators may be used for each of the different functions. In general, the form of heuristic functions is determined by the restrictions under which it is constrained. Thus, a Combinator Generator employs a similar learning technique for both guidance and transformation functions, but the specific rules it uses are different.

Each Generator uses a different type of knowledge about generation of attributes. Representation of that knowledge is important. TEACHER 2.0 uses two forms of representing this processing knowledge: procedural, and heuristic rules.

A *procedural* module in an Attribute Generator follows a set of actions, procedures, or rules which are explicitly stated in the program. An example would be code in the Decomposition Generator to propose each entity variable as an Attribute (in the appropriate form, i.e. $max\,(entity_i)$ for transformation, or $\sum_i entity_i$ for guidance).

A *heuristic rules* module in an Attribute Generator represents knowledge in the form of a set of:

if <condition> then <follow procedure to generate attribute(s)>

statements. A simplified example is the following rule from a Combinator Generator:

if [(ATT_A⇒eval.>0) and (ATT_B⇒eval.<0)] then propose:

<procedure for generating proper format for attribute ATT_A/ATT_B>

This rule assumes that each evaluation field has a positive value for a productive heuristic, and a negative value for attributes which are detrimental. For this rule, its condition is dependent on the type of evaluation function employed. These rules differ slightly from the production rules [17] form of knowledge representation in that every condition match results in proposal of an attribute (subject to the Scheduler in the Manager). This is due to the fact that generation of all possible (preferably effective) attributes is desired.

Both procedural and heuristic rule modules may be active in a given Generator. In fact, the procedural representation may be thought of as a heuristic rule in "if <true> then <...>" format.

MANAGER: The Manager is responsible for handling the pool of attributes and scheduling the search using the physical resources available, such as time, and memory. Each attribute is considered to be a potential heuristic. The Manager is responsible for presenting attributes to the evaluator for evaluation, thereby controlling the scheduling of the Evaluator. The Evaluator (see below) is not be able to perform a complete evaluation of a heuristic; evaluation is done by experimentation. An attribute is evaluated for a fixed time quantum. After that quantum the MANAGER determines whether to continue evaluation on that attribute, begin evaluation of a new attribute, or continue evaluation of an attribute which had been set aside previously. Implicit to the task of generating new heuristics on the basis of old ones and handling the search of heuristics is the task of scheduling for the search. The manager has two subcomponents: the Scheduler, and the Record Manager.

The Scheduler: The Scheduler is responsible for determining what areas and techniques should be searched, and prioritizes them. The ATTRIBUTE GENERATORS use many techniques for learning new attributes. Some of these techniques may be more or less effective at learning new attributes, depending on the target numeric optimization problem. The Scheduler determines which techniques should receive the greatest attention. The Scheduler also determines which attributes, in the pool of attributes, have the greatest potential for use in the learning of new heuristics.

In both cases, the Scheduler determines priority in the same way. The basis is empiricism, the technique or attribute which in the past has yielded the best candidates receives higher priority for new learning. If an impasse is reached, i.e. a moderate amount of time has passed without the return of an effective term or attribute, then the learning effectiveness of that term or attribute is decreased. The initial rating of an attribute is made on the basis of the information supplied by the EVALUATOR. Thus, the Scheduler can be thought of as performing a best-first search over the space of possible heuristics.

The Scheduler also determines the resources to be used by the EVALUATOR in order to evaluate an attribute. Attributes are evaluated for fixed time quantums. After the time quantum, the Scheduler determines whether the evaluation should continue, or a different attribute should be evaluated in that quantum. In this way, very bad heuristics will be evaluated quickly (not consuming are a large amount of the computer resources), while stronger attributes will be evaluated more carefully.

The Record Manager: The Record Manager maintains a data structure containing the records of attributes. A sample record was illustrated in Section IV.B. For convenient access, the Record Manager may categorize attributes by the manner in which they relate to good behavior of a heuristic, as determined by the EVALUATOR.

EVALUATOR: The Evaluator is responsible for evaluating and rating a heuristic. The basis for this determination is the empirical relationship the heuristic has with sample problems. The Evaluator is

Table 6. Possible Evaluation Functions.

Target Function	Criteria for Evaluation of		
	Number of Nodes	Communication	Processing Complexity
Bound	CSS*; correlation to real value; mean % error; average depth of prune; probability of prune; etc.	CSS; number of nodes measures; average difference between best and second best bound; etc.	Average at each node.
Guidance	CSS; correlation to real value; mean % error; probability selected node is must be evaluated; etc.	CSS; number of node measures; probability of hopping; probability of hopping from node that is eventually evaluated; etc.	Average at each node.
Transformation	CSS; probability selected variable in solution; mean decrease in bound function; probability children are pruned; etc.	CSS; guidance function measures; etc.	Average at each node; determination if can be done in preprocessing; etc.

*CSS: Complete Search Samples.

composed of two units: a Rating Unit, and a Tree Generator. The Rating Unit assigns values to the evaluation field in an attribute record. The Tree Generator generates sample searches and nodes for use by the Rating Unit.

The Rating Unit: The Rating Unit is responsible for determining the relationship between a potential heuristic and the performance of an "optimal" heuristic. The relationship between an attribute and "optimal" performance is strongly determined by both the objectives of the learning task and the type of heuristic learned. A guidance function, for example, may wish to both minimize the number of nodes expanded *and* minimize backtracking. In both cases, the evaluation process will be much different than when evaluating a transformation function. The form that the Rating Unit takes is dependent on the type of heuristic to be learned.

In Section II.C, three measures were described that reflect the performance of a search using a heuristic: number of nodes, communication behavior, and processing time required for evaluation at each node. It would be very time consuming to use a potential heuristic for performance of a complete solution to many problem instances and averaging their behavior, in order to evaluate the potential heuristic. It would be preferable if other measures could be developed that do not require such a large computation for each individual measurement. Some possibilities for these measure are indicated in Table 6.

The degree to which any of these measures reflects the larger goal of number of nodes expanded, memory behavior, or etc., is dependent on the problem domain. The Rating Unit is responsible for determining the relative importance of the measures, and must inform the Manager of the function used to combine the fields into an overall ranking of the function. The Rating Unit "learns" which measures are appropriate by comparing the measures to real search performance.

The role that these measures plays in the formulation of attributes depends largely on the objectives of the user. If there is no memory hierarchy, then the user may not care about memory behavior, and may only be concerned with the average number of nodes expanded. The relative importance of the measures is dependent on the system on which the search will be implemented. Thus, the user is responsible for supplying available measures, and the Evaluator determines the role each measure plays.

Naturally, evaluation of a given heuristic function is impacted by the specific functions used for the other two types of functions. There are two alternatives. One is to evaluate the attribute on the basis of several combinations of heuristic functions. The other strategy is merely to use the best known functions for the other heuristics and iteratively refine each type of heuristic.

Tree Generator: The Tree Generator generates sample nodes for use by the Rating Unit. This involves generation of problem instances, and solving these problem instances. As there may be no known heuristic for solution of the problem, the Tree Generator must solve problem instances by exhaustive enumeration, with a random transformation function.

Unfortunately, this prevents the use of extremely large examples by the tree generator. This is not necessarily a disadvantage. It has been postulated that most heuristics behave more accurately near the end of a search [18]. Thus, resolution can be performed accurately at this level. If a heuristic is thought to be a very good candidate, it can later be tested on larger trees. Nodes from a large number of sample searches should be used in order to keep from biasing the evaluation of the nodes. In the case where no evaluation function has been discovered, the evaluator will interact directly with the tree generator for performance of searches using the attribute being evaluated. Measurements on these searches will then be used as the criteria for evaluation.

IV.D. Illustrative Example

In this section, a greatly simplified example of the TEACHER 2.0 method for learning heuristics is presented. The intent is to demonstrate how the components of TEACHER 2.0 interact. As transformation functions have the most restricted form, they are natural candidates for the simplified example. The result of the example is counterintuitive. This illustrates the value of developing systems such as TEACHER 2.0.

The problem will be to learn transformation functions for the 0/1KP with the bound function which repeatedly places the remaining item with $max(v_i/w_i)$ into the sack, until the capacity of the sack is exceeded. The guidance function is best-first search on the basis of the bound function. The transformation function will be evaluated on the basis of problems with the value and weight of each item being random and uniformly distributed, 1 through 10. The capacity for sample problems will be $K = (N/2)$*(random uniform 1 through 10). The distribution of K increases with N so that reasonable searches with roughly half the items fitting into the sack are generated. The relative values of potential transformation functions are dependent on the bound and guidance functions, as well as the distributions for sample problems. This example is merely exemplar of how TEACHER 2.0 performs learning.

To avoid unnecessary complexity in the example, only transformation functions which statically order the selection of items are considered. As the ordering is independent of the node, this ordering can be done prior to starting the search, and processing time is not an issue. In this example, the Evaluator evaluates attributes by generating complete sample searches, and taking measurements. For the example, only number of nodes generated will be used as a measure of the performance of the heuristic. The quantum used for evaluation in the simple example is a fixed time unit (time being measured in number of nodes expanded). The number of problems solvable in the time unit is indicative of the quality of the transformation function.

The usual transformation function used is $\max(v_i/w_i)$ [3, 7]. In the example, this transformation function is input by the user through the Advice Taking Generator. The search generated by TEACHER 2.0 is illustrated in Figure 4. In the figure, each box contains the number of the attribute, the name of the Attribute Generator which generated the attribute, the attribute itself, the attribute(s) used as input(s), and the evaluation (x) For the figure the evaluation, x, is just an average over 1000 randomly generated problems of size N = 20. The performance of the search is described by Table 7. For simplicity, all attributes are assumed to be of the form $\max_i:(att)$.

The example is specific to the 0/1KP, but the approach and rules used for attribute generation are applicable to other problems as well. For example the (only) rule used by the Decomposition Generator in the example is:

if <rule in form x*r* y> then <propose attribute x and attribute 1/y>
where *r* is a negative relation

This rule distinguishes between positive relations, such as addition and multiplication, and negative relations such as division and subtraction. The heuristic involved is similar to a rule used by the Combinator:

if <attribute x is "good"> and <attribute y is "good"> then
<propose attribute (x + y) and attribute (x * y)>

It is based on the assumption that if x and y are good, then x added or multiplied to y may be better. In the above rule, the definition of "good" is

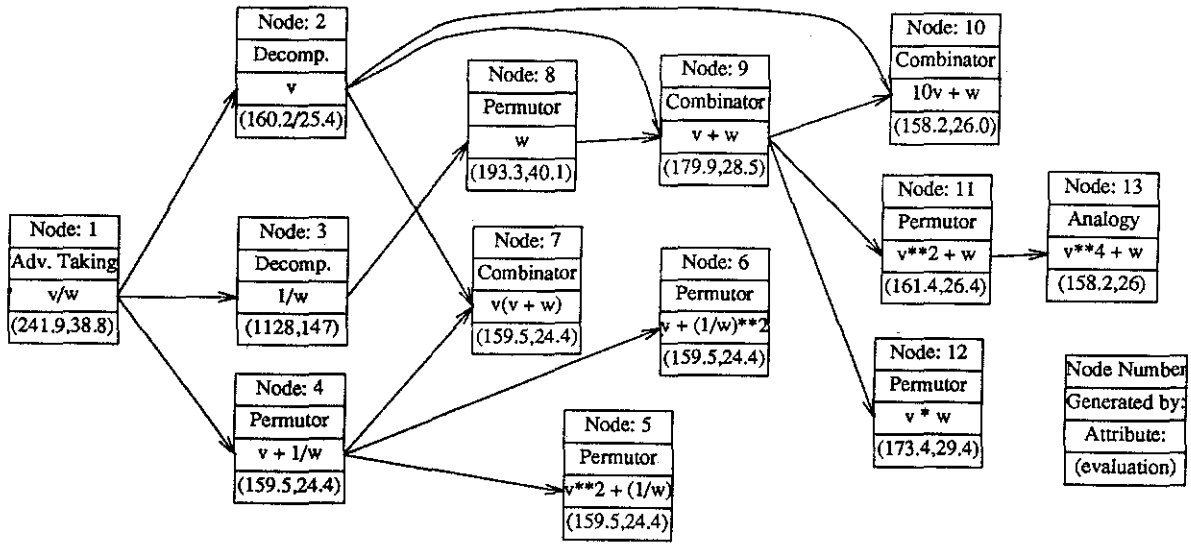

Figure 4. Example Learning of Transformation Functions for the 0/1KP

specific to the evaluation used, and is partially determined by the Manager.

The order of expansion in Figure 4, indicated by the node number, is representative of the manner in which the Manager would control the search. Heuristics which have been evaluated to be the best, and heuristics which show dramatic improvement in their evaluation are likely to be scheduled earlier in the search. The search tree in Figure 4 is representative of where the Manager could stop the search; all expansions of nodes have failed to improve the evaluation measures. In reality, the search would be much larger. Many more rules would be invoked, and many more attributes would be explored. The specific rules invoked for generation and beginning the search from $\max_i(v_i/w_i)$ were done for the purpose of generating an illustrative example.

Table 7. Example Search Nodes.

Node	Comment
Node 1	Input by user through Advice Taking Generator.
Node 2 Node 3	Decomposition of Node 1.
Node 4	v_i/w_i is just $v_i * \frac{1}{w_i}$. Permutor changes the multiplication to addition. This is a reasonable heuristic substitution as addition and multiplication are both positive relations.
Node 5 Node 6	Two permutations of node 4. The permutation is an example if a heuristic rule for changing the relative weight of the two parts.
Node 7	Nodes 5 & 6 failed to improve Node 4. The Manager would try to combine 2 & 4 as they are the most promising heuristics so far. This node is an attempt by the Combinator to combine nodes 2 & 4.
Node 8	Node 3 is a dramatic change from Node 1. This makes it a candidate for negation or inversion. This is done by the Permutor.
Node 9	A Combinator assembly of Nodes 2 and 8. Node 8 is a dramatic improvement from Node 3. This makes the Scheduler increase its priority for formulation of heuristics.
Node 10 Node 11 Node 12	These are all permutations of node 9. Node 9 was an improvement of node 8, and so further modification of this node would be scheduled.
Node 13	This is an example of how the Analogy technique of learning may be used in ways other than analogy from other problem domains. Squaring v_i in Node 11 resulted in an improvement. Analogy is then used to take this a greater extreme by raising v_i to the fourth.

The best heuristic discovered in this search, with respect to average number of nodes was $max(v_i^4+w_i)$ and $max(10v_i+w_i)$. The best with respect to average number of backtracks was $max(v_i+w_i)$. Just $max(v_i)$ performs surprisingly well. It is interesting to note that the $\max(v_i/w_i)$ heuristic was one of the worst heuristics explored. A plot of the performance of some discovered heuristics, and the $\max_i(v_i/w_i)$ heuristic is shown in Figure 5. The figure serves two purposes. First, it indicates that the use of $N = 20$ was adequate for evaluating the heuristics. Second, it highlights the value of the system. This figure illustrates the gains possible by applying TEACHER 2.0 to numeric optimization problems. At $N = 30$, the heuristic is already giving a nearly twofold improvement in number of nodes expanded. As N grows, so does the improvement. Looking at the curves, it can be seen that this small piece of the TEACHER 2.0 search space enables a problem with 20% more variables to be solved using the same resources.

The example illustrates the value of systems like TEACHER 2.0 and the fact that there is a great deal of knowledge which has yet to be gained in this area. This example is illustrative of the functioning of TEACHER 2.0. The actual derivation of interesting new transformation functions, however, was more a function of the actual evaluation of the alternatives as the specific form of TEACHER 2.0.

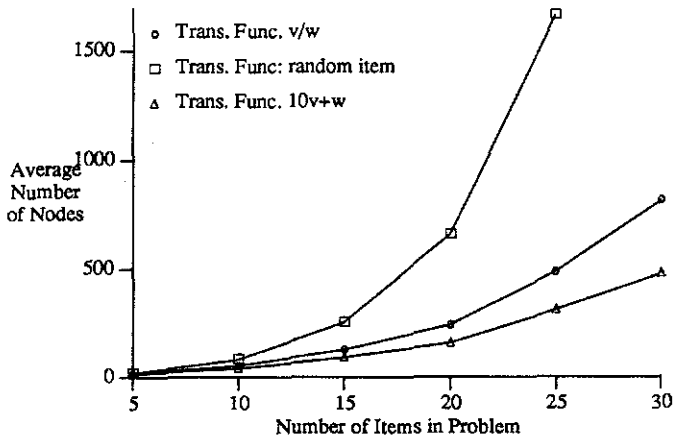

Figure 5. Average Number of Nodes Performance of Transformation Functions.

Table 8. Learning Techniques Used in TEACHER 2.0.

Unit	Primary Learning Techniques	Comments
Attribute Generator	Relaxation	Generates Attributes
	Analogy	Generates Attributes
	Instruction	Generates Attributes
	Systematic Exploration	Generates Attributes
	Heuristic Rules	Generates Attributes
Manager	Heuristic Scheduling	Of Tasks.
Evaluator	Probabilistic Estimation	For rating heuristics
	Statistical Analysis	
	Experimentation	

The superior performance of the "learned" transformation functions is due to the ability of the function to make branches which allow pruning higher in the search tree. This results in fewer average number of nodes expanded. In general, humans are not always good at determining the features of the transformation function which allow earlier pruning. For this reason, TEACHER 2.0 is capable of making a contribution in this area.

V. CONCLUSIONS

In this paper, the problem of learning heuristic functions for increasing the efficiency of search solutions to numeric optimization problems was discussed. A strategy for development of a learning system was developed, tailored to the domain of learning for numeric optimization problems. The strategy was then carefully followed.

Design of the system began with an analysis of the problem domain, and the areas and characteristics of the target of the learning process. Three types of heuristics were found to be of interest: bound functions, guidance functions, and transformation functions. Many learning techniques, both general and specific to heuristic functions were then delineated and evaluated.

Finally, a system for learning heuristics was proposed. The system was designed using as many learning techniques as possible. A learning philosophy was developed which enables coherent incorporation of many learning strategies into one system. Attributes can be generated by many different learning techniques, and allow the system to generate heuristics in one or many steps. This makes a powerful system capable of learning heuristics based on more than one learning strategy.

By allowing more than one learning technique, the system is not restricted to learning only a small class of potential heuristics. By dynamically scheduling the search process, the system not only learns heuristic functions for numeric optimization problems, but is capable of tuning itself to perform the learning task more efficiently. The system is unusual in its use of many different learning strategies; the strategies are enumerated in Table 8 by where in the system they are employed.

VI. REFERENCES

[1] H. J. Berliner, "An Examination of Brute Force Intelligence," *IJCAI*, pp. 581-587, 1981.

[2] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "An Overview of Machine Learning," in *Machine Learning*, ed., R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. Tioga, 1983.

[3] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*. New York, NY: John Wiley & Sons, 1972.

[4] J. Gaschnig, "A Problem Similarity Approach to Devising Heuristics: First Results," *IJCAI*, pp. 301-307, 1979.

[5] A. M. Geoffrion and R. E. Marsten, "Integer Programming Algorithms: A Framework and State-of-the-Art Survey," *Management Science*, pp. 465-491, May 1972.

[6] A. M. Geoffrion, "Lagrangian Relaxation and Its Uses in Integer Programming Problems," *Mathematical Programming Study*, vol. 2, pp. 82-114, 1974.

[7] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. Rockville, Md.: Computer Science Press, 1978.

[8] V. Kumar and L. Kanal, A. Martelli, and U. Montanari, and Judea Pearl, "Knowledge versus Search: A Quantitative Analysis Using A*," *Artificial Intelligence*, vol. 20, pp. 1-13, 1983.

[9] V. Kumar and L. N. Kanal, "A General Branch and Bound Formulation for Understanding and Synthesizing And/Or Tree Search Procedures," *Artificial Intelligence*, vol. 21, pp. 179-198, 1983.

[10] D. B. Lenat, "The Nature of Heuristics," *Artificial Intelligence*, vol. 19, pp. 189-249, 1982.

[11] D.B. Lenat, "EURISKO: A Program That Learns New Heuristics and Domain Concepts; The Nature of Heuristics III: Program Design and Results," *Artificial Intelligence*, vol. 21, pp. 61-98, 1983.

[12] D. B. Lenat, "Theory Formation by Heuristic Search; The Nature of Heuristics II: Background and Examples," *Artificial Intelligence*, vol. 21, pp. 31-59, 1983.

[13] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine Learning: An Artificial Intelligence Approach, Vol. 1, 2*. Los Altos, Ca.: Morgan Kaufman Inc., 1983, 1986.

[14] 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. Los Altos, CA: Morgan Kaufmann, 1986.

[15] S. Minton and J. G. Carbonell, "Strategies for Learning Search Control Rules: An Explanation-based Approach," in *Proc. Tenth Int'l Joint Conf. on Artificial Intelligence*, Milan, Italy, pp. 334-337, Aug. 1987.

[16] L.G. Mitten, "Branch-and-Bound Methods: General Formulation and Properties," *Oper. Res.*, vol. 18, pp. 23-34, 1970, Errata in Oper. Res. 19 (1971) 550.

[17] A. Newell, "Production Systems: Models of Control Structures," in *Visual Information Processing*, ed., W. G. Chase. Academic Press, 1975.

[18] J. Pearl, *Heuristics--Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley, 1984.

[19] L. A. Rendell, "A New Basis for State-Space Learning Systems and a Successful Implementation," *Artificial Intelligence*, vol. 20, pp. 369-392, 1983.

[20] L. A. Rendell, "A Doubly Layered, Genetic Penetrance Learning System," *Proc. of the AAAI*, pp. 343-347, 1983.

[21] H. A. Simon, "Why Should Machines Learn?," in *Machine Learning: An Artificial Intelligence Approach*, ed., R. S. Michalski, J. G. Carbonell, T. M. Mitchell. Los Altos, Cal.: Morgan Kaufmann Publishers Inc., pp. 25-37, 1983.

[22] B. W. Wah and C. F. Yu, "Stochastic Modeling of Branch-and-Bound Algorithms with Best-First Search," *IEEE Trans. on Software Engineering*, vol. SE-11, pp. 922-934, Sept. 1985.

[23] B. W. Wah, G. J. Li, and C. F. Yu, "Multiprocessing of Combinatorial Search Problems," *IEEE Computer*, vol. 18, pp. 93-108, June 1985.

[24] C. F. Yu, *Efficient Combinatorial Search Algorithms*. West Lafayette, IN: Ph.D. Thesis, School of Electrical Engineering, Purdue University, Dec. 1986.

[25] C. F. Yu and B. W. Wah, "Learning Dominance Relations in Combinatorial Searches," *Trans. on Software Engineering*, vol. SE-14, No. 8, Aug. 1988.