TWO LEARNING METHODS FOR A TREE-SEARCH COMBINATORIAL OPTIMIZER

Marek A. Perkowski *, *Pawel Dysko*+, *Bogdan J. Falkowski* *,

* Department of Electrical Engineering,
Portland State University,
P.O. Box 751, Portland, Oregon 97207,

+ Department of Automatic Control,
Technical University of Warsaw,
Nowowiejska 15/19, Warsaw, Poland.

ABSTRACT

Several combinatorial problems of logic synthesis and other CAD problems have been solved in an uniform way using a general purpose tree searching program MULT-II. This paper presents two learning methods that have been implemented to improve program's efficiency. A weighted heuristic function, used to evaluate operators, is applied during searching a solution tree. The optimal vector of coefficients for this function is learned in a simplified perceptron scheme. By using the second learning method, the similarity of shapes among the solution cost improvement curves is used to define the termination moment of the search process. The amplification effect of concurrent action of both these methods has been observed.

1. INTRODUCTION

There exist many problems of logic synthesis and related combinatorial algorithms of VLSI CAD that require tree searching methods to solve them. In particular, especially many of such problems can be reduced to a class of NP-hard combinatorial problems that can be characterized as *constrained logic optimization problems.* They are described using multi-valued Boolean functions, graphs, or arrays of symbols, on which some constraints are formulated and some transformations are executed in order to optimize cost functions. These problems include Boolean satisfiability [22], tautology [58], complementation [57], set covering [23], even/odd set covering [21], clique partitioning [62], maximum clique [62], graph coloring, maximum independent set, set partitioning, matching, linear and quadratic assignment, covering/closure, edge covering, and others. Considering the importance of these problems, several different approaches have been introduced to solve them. These approaches include:

- mathematical analysis of the problems is done in order to find algorithms as efficient as possible algorithms (exact or approximate), or algorithms for particular sub-classes of these problems [7, 54-58], in spite of the fact that the problems are NP-hard so that no efficient (polynomial) algorithm exists them.
- special hardware accelerators [22,40,58] are designed to speed-up the most often executed or slowest operations on these types of data.
- general purpose parallel computers, like message-passing hypercubes, SIMD arrays, data flow computers and shared memory computers are used [9].
- the ideas of Artificial Intelligence, computer learning and neural networks are used, also mimicking humans that solve these problems [39,43,44].

This paper follows the last approach. Since the problems of the constrained logic optimization class, as NP-hard, will be always difficult to solve, one tries to find good heuristics that take into account the peculiarities of real life data in order to maximize execution efficiency, even by sacrificing the understanding why this or other technique works well. When a problem of developing a new algorithm is encountered, the logic theorist/program developer has, based on his experience and a large collection of similar problems, to find an appropriate tree-searching algorithm and the corresponding heuristics. A software system has been developed that helps the designer in this task, MULT II, a fast prototyper that permits to check, evaluate and compare design ideas before writing the final code [41-44,47]. MULT II is a general purpose combinatorial problem solving program that has been used to solve many problems related to logic synthesis, operations research, Finite State Machines, physical design, graph theory and other areas. The program permits the user to choose values for several parameters that define searching strategy, (like depth-first, ordered search, branch-and-bound), and other search heuristic routines. Some of the parameters are used to select a combination of *learning methods* for the improvement of the search efficiency. This way the program designer can quickly evaluate usefulness of his various ideas; how much each of them contributes to the success of the tree search.

Two learning methods are presented in this paper. The first method learns criteria of selecting good operators by using a weighted evaluation function and learning its weight coefficients. The same approach is used for selecting nodes as well, where the coefficients of the evaluation function for solution tree nodes are learned. Another variant of this method observes the fact that a search strategy in MULT-II can be described by a set of subroutine flags that are used to connect or disconnect the respective subroutines from the main tree-searching program. The subroutine with a flag taking values $0 \leq p \leq 1$ is connected with a probability of p. The probability coefficients of the flags can be learned in a similar way that the coefficients of the evaluation function are learned. Analogous approaches have been successfully applied in game playing programs and pattern recognizers [2,3,8,19,33,35,37,38,51-53,59,65] but to the best of our knowledge, they have never been used to solve the class of combinatorial design problems considered by us.

The second, completely new learning method, used in Branch-And-Bound strategies of MULT-II, determines such a moment of search in which backtracking can be stopped, because a better solution is very unlikely to be obtained in future. In many problems a strategy can be constructed that quickly leads to a minimal solution, however, it takes a very long time for the computer to prove that the solution is really the minimal one. Therefore, it is important to know the moment when the further backtracking can be stopped. The method uses a normalized shape diagram which for different data of the same problem refers the cost function values to the number of subsequently generated solutions. The predictions of the stopping moment are calculated for some estimated probability of not loosing the optimal solution.

Section 2 presents the multipurpose multistrategical problem solver with learning. In section 3 the evaluation function learning method and its illustration with a *set covering problem* are presented. It is well known, that the PLA minimization problem, microcode optimization, data path allocation, TANT network minimization problem [], factorization, test minimization and many other logic synthesis problems can be reduced to the set covering. Hence, the latter problem can be treated as a *generic logic synthesis subroutine.* Several efficient algorithms for this problem have been created [4-6,12,14,18,22,23,34,49,64]. Section 4 presents the backtracking stopping learning method and illustrates it with a *linear assignment problem* that finds several applications in VLSI layout, logic synthesis, operations research, production scheduling, marriage counseling, economics, communication networks, personnel administration, clustering, psychometrics, and statistical inference [1,9,17,18,24,25,27,29,30,36,48,60,63]. It is closely related to the traveling salesman problem [31]. Interestingly, the same problem finds also applications to create general-purpose efficient heuristic learning schemes. Section 5 illustrates a newly formulated and more comprehensive problem of logic synthesis: design of easily-testable EXOR/AND trees for multiple-valued input incompletely specified logic functions [11,20,21,45]. The search problem here is to find the best sequence of input variables for a Shannon-like expansion, one that minimizes the circuit's complexity.

2. MULT-II - THE MULTIPURPOSE MULTISTRATEGICAL COMBINATORIAL PROBLEM-SOLVER WITH LEARNING

2.1. BASIC IDEAS

Program MULT-II has been written in FORTRAN 77 and implemented on VAX 11/750 as well as on Gould 9080. MULT-II uses a *state-space tree search method* that is realized by a multipurpose, multiapplication subroutine: MULTCOM. It was used for PLA minimization, FSM state minimization [41,47,66], FSM two-dimensional minimization [66], FSM state assignment [32], FSM concurrent state minimization and state assignment [32], TANT network synthesis [46], negative gate network synthesis [42], graph coloring, logic puzzles, board games and robot path planning [43,44].

MULTCOM realizes the design task by seeking to find a set of the solutions that fulfill all problem conditions. It checks a large number of partial results and temporary solutions in the tree search process until it finally proves the optimality of the solutions. The *state-space S for a particular problem* solved by the MULT-II program is a set which includes all the solutions for the problem. New states are created from previous states by application of *operators.*

By assigning values to parameters the user of this method can experiment with variants of problem description and create various search strategies for different tree search methods to optimize the efficiency of the search. For example, in some cases the search can essentially be limited by the use of the problem's symmetry. Then, when the user defines a symmetry parameter, the relation of the equivalency is checked on the operators in all of the nodes. Subsequently, all but one of the operators is removed from each equivalence class. This method expedites, among others, covering problems.

During the realization of the search process in the state-space, a memory structure termed *solution tree, solution space,* is used. The solution tree D = [NO, RS] contains *nodes* of NO (which stands for the stages in the solution process) and *arrows* of RS. The arrows are labeled by the *descriptors of the operators.* Each node contains a description of a state-space state and some other search related information, in particular descriptors of the operators to be applied. Descriptors are some simple data items and can be created, manipulated and stored in nodes or removed from them. Operators traverse from node to node what is equivalent to searching among the states of S. Each of the solution tree's nodes is a *vector of data structures.* This vector's *coordinates* are denoted as follows: NC - node number, SD - node depth, PC - node number of the immediately preceding node, OP - descriptor of the operator applied from PC to NC, F - node cost, AS - description of the hereditary structure, QS - partial solution, GS - set of descriptors of available operators, NAS - actual length of list AS, NQS - actual length of list QS, NGS - actual length of list GS. Additional coordinates can be defined when required.

The operator descriptor is denoted by OP, application of operator O with the descriptor OP to node N of the tree is denoted by O(OP, N). A *macro-operator* is a sequence of operators that can be applied successively without retaining the temporarily created nodes.

Prerequisite to formulation of the design problem for the search model is to specify the necessary coordinates for the specified problem in the *initial node - the root of the tree.* The way in which the coordinates of the subsequent nodes are created from the preceding nodes must be specified as well. This leads to the description of the *generator of the solution space (tree generator).* Solution conditions and/or cost functions should be formulated for most of the problems.

QS is the *partial solution* i.e, the portion of the solution that is incrementally grown along the branch of the tree until the final solution is arrived at. A set of all possible values of QS is a state-space of the problem. It is assumed that some relation $RE \subset S \times S$ of a partial order exists. Therefore, the state $s \in S$ symbolically describes the set of all $s' \in S$ such that s RE s'. The solution tree usually starts with $QS(N_0)$ which is either the *minimal* or the *maximal element* of S. The set GS(N) of descriptors denotes the set of all operators that can be applied to the node N. The *hereditary structure* AS(N) denotes some properties of the node N that it has inherited along the path from the root. The *solution* is a state of space that meets all the *solution conditions.* The *cost function* F is a function that assigns the cost to each solution. The *quality function* QF can be defined as the function of integer or real values pertinent to each node. If QS(N) is the solution, then QF(N) = F(N). D(N) denotes a subtree with node N as a root. Often function QF(N) is defined as a sum of function F(N) and a *heuristic function* $\hat{h}(N)$: $QF(N) = F(N) + \hat{h}(N)$. $\hat{h}(N)$ evaluates the distance h(N) of node N from the best solution in D(N). F(N) in such a case defines a partial cost of QS(N). One attempts to define $\hat{h}$ in such a way that it as close to h as possible (see [38] for general description and [39,43,44] for applications in VLSI CAD problems). An *optimal solution* is a solution $QS(N) = s \in S$ such that it does not exist $s' \in S$ where F(s) > F(s'). The problem can have more than one optimal solution. Additional *quality functions for operators* can also be used.

2.2. THE MULT-II SEARCH STRATEGIES

A number of *search strategies can be* specified for the tree search procedure with the quality functions. Beginning with the initial node, the information needed to produce the solution tree can be divided into *global information,* that relates to the whole tree, and *local information.* Local information in node N refers to subtree D(N). The user-specified search strategies are also divided into a *global search* and a *local search.* The selection of the strategy by the user is based on a set of *Strategy Describing Parameters* that can also be dynamically changed by the program during the search and learning process. They set some values or connect some *Strategy Describing Subroutines.*

The approach of MULT-II offers the following advantages:

- The *quasi-optimal solution* is quickly found and then, by backtracking, successive, better solutions are found until the *optimal solution* is produced. This procedure allows for the trade-off between the quality of the solution and speed of arriving at it.
- The search in the state-space can be limited by including as many *heuristics* as required. In general, a heuristic is any rule that directs the search and is described in the form of a *parameter-connected subroutine.*
- The application of various quality functions, cost functions, and constraints is possible.
- By using macro-operators along with other properties, the strategies require less memory than the comparable, well-known search strategies [38,59].

The search strategy is either selected from the *universal strategies* (Breadth-First, Depth-First, Branch-And-Bound, Ordered-Search, Random) or it is created by the user's assigning of values to *local strategy describing parameters* and writing of the *sections' codes.* In the Branch-and-Bound strategy the temporary cost B is assigned which retains the lowest cost of the solution node already found. Whenever a new node NC is generated, its cost F(NC) is compared to the value of B. All the nodes whose cost exceed the value of B will be cut off from the tree. In the Ordered-Search strategy the quality function Q(OP, NC) is defined to evaluate the cost of all the available descriptors of the node being extended. These descriptors are applied by the subroutine realizing the operators; in the order that corresponds to their, evaluated earlier, costs. This strategy, as well as the Random one, can be combined with the Branch-and-Bound strategy.

The *Strategy Describing Subroutines* are outlined below. There are two types of conditions for each node of the tree: *by-pass* and *cut-off.* If the cut-off condition (i.e., the predicate function defined on node N as an argument) is met in node N, the subtree D(N) is prevented from being generated and backtracking results. The by-pass conditions do not cause backtracking and the tree will continue to extend from node N. The following cut-off conditions exist:

a) *bound condition* - it is known (possibly from information created in node N) that node N_1 exists (not yet constructed) such that $F(N_1) < F(N)$ and $QS(N_1)$ is a solution.

b) *depth limit condition* - SD(N) is equal to the declared depth limit SD_{max}.

c) *dead position* - no operators can be applied to N i.e., GS(N) = ∅.

d) *restricting conditions* - QS(N) does not fulfill certain restriction, i.e., no solution can be found in D(N).

e) *solution conditions of the cut-off type* - if QS(N) is a solution, then for each $M \in D(N)$, F(M) > F(N) (or F(M) ≥ F(N)) and M may not be taken into account.

f) *other types of conditions* formulated for some other type of restrictions special to problems (these are user selected by setting flags for MULT-II communication).

The following relations between the operator descriptors (so called *relations on descriptors)* can be described by the programmer to limit the search process: *relation of domination, relation of global equivalence, relation of local equivalence* [43,44]. When all the solution conditions are met in a certain node N, the QS(N) is a solution to the given problem. Then, the latter is added to the set of solutions and is eventually printed. The value of B := F(N) is retained. If one of the solutions is of the cut-off type, the program backtracks. Otherwise, the branch is expanded.

2.3. THE MULT-II STRUCTURE

The simplified structure of the Problem-Solver of MULT-II is shown in Fig. 2.1. Except for the portion inside the dotted line rectangle, all the segments can be selectively linked and modified to meet the design problem formulation and strategy description.

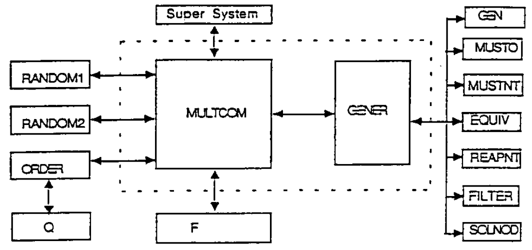

Fig. 2.1. Structure of the Problem-Solver in MULT-II.

MULTCOM is in charge of the global search which includes the selection of strategies, the arrangement of the *open-list* of nodes [38] and the other lists as well as the decision making facilities related to the cut-off branch, and the configuration of the memory structure for the tree. The segments that realize the strategies of Breadth-First, Depth-First, or Branch-And-Bound are built into MULTCOM. RANDOM1 and RANDOM2 are selectively linked for the random selection of the operator or the open node. The role of the main subroutines linked to MULTCOM is as follows: GENER is responsible for the local search that extends each node. GENER cuts off the nodes which will not lead to the solution node when the description for the new node is created. GEN carries out the task of creating nodes. MUSTO serves to find the *indispensable operators,* the operators that must be applied. This set is then substituted as the new value of coordinate GS(N). MUSTNT deletes *subordinate operators* that would lead to worse solutions or no solutions at all. The set MUSTNT(N) is subtracted from set GS(N). EQUIV cancels those nodes that are included in other nodes. FILTER checks whether the newly created node meets the conditions. SOLNOD checks the solution condition. REAPNT is used to avoid the repeated applications of operators when the sequence of operator applications does not influence the solution. ORDER sorts the descriptors, Q calculates the quality function for the descriptors, and F calculates the cost of the nodes. The conditions, relations, sorting and selecting functions and strategy parameters, all together describe some "personalized" solution tree searching method and strategy. The possibility of dynamic modification of flags and coefficients used in them is *the basic principle of learning in MULT-II.* It is presented in Fig. 2.2.

The Problem Solver in the top left corner is one from Fig. 2.1. Its strategy is specified by a Strategy Vector - an ordered vector of numerical coefficients called **Strategy Parameters** (in general real numbers, often zeros and ones or numbers from [0, 1]). Each value of the Vector describes one of the strategies that can be realized by the Problem-Solver. The sub-vectors of the System Vector (subsets of Strategy Parameters) are created by four Learning Schemes:

1. Learning Scheme # 1 - learning the Operators Evaluation Function.
2. Learning Scheme # 2 - learning the Tree Nodes Evaluation Function.
3. Learning Scheme # 3 - learning the moment when to stop the searching.
4. Learning Scheme # 4 - learning the probabilities of calling various Strategy Defining Subroutines.

The user can set the initial values of all Strategy Parameters and some of them cannot be modified by learning. The Strategy Vector Generator checks the consistency of parameters. Certain subroutine flags are *conditionally disjoint*. For instance, there can be three subroutines, SS1, SS2, and SS3, to perform sorting of operators, according to three different methods and comparison criteria to be used in a tree's node. Since only one of them can be used at given time, selecting SS1 (flag for SS1 enabled) will disable flags for SS2 and SS3. Let us observe, that the sum of probabilities for parameters SS1, SS2 and SS3 does not have to be equal one, since in any case, selection of one of the subroutine flags will disable the other ones from the group. In general, the *Exclusion Conditions* can be of the form: *Si & Sj* or *Sk & Sl & Sf -> not Su & not Sv* which means that if flags *Si* and *Sj* or flags *Sk, Sl* and *Sf* have been selected then flags *Su* and *Sv* must be disabled.

The four learning schemes have their local memories that store sets of pairs [*partial vector sample, its cost*] or other similar data. The costs are provided by calculations of some solution parameters, like cost function values, total solution times, solution cost improvement times, sizes of the used memory. All these methods are "orthogonal" and can be applied separately or together. For strategies #1, 2, and 4, improper learning or lack of its convergence cannot result in non-minimal solutions, but the optimal solution can be delayed.

3. THE FIRST METHOD - LEARNING THE EVALUATION FUNCTION.

3.1. THE METHOD

Several studies lay the stress on the importance of selecting an appropriate Evaluation Functions (Quality Functions) while searching a tree [59,38]. This section considers only Branch-And-Bound strategies, or mixed strategies, that combine the Branch-And-Bound and Ordered-Search Strategies. This principle is used in Learning Schemes # 1, 2, and 4.

Let the measure of the intelligence of the system be the number of nodes which have to be generated to find the optimal solution. It is easy to observe that this number

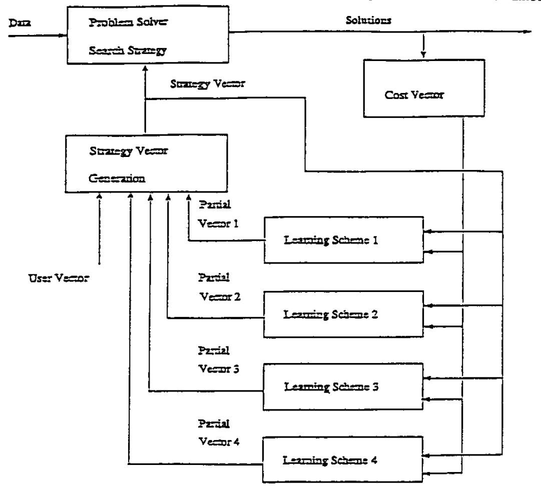

Fig. 2.2. Basic principle of learning in MULT-II.

depends on how quickly the program will encounter the appropriate branch (it means one that terminates with the optimal solution) and this will in turn depend on the good selection of evaluation functions for operators and nodes. For instance, in MULT-II the quality function for operators can be defined in a form

$$\sum_{i=0}^{M} c_i f_i = C^T \cdot F$$

where $F^T = (f_1, f_2, ..., f_n)$ is a vector of *partial quality functions* and $C^T = (c_1, c_2, ..., c_n)$ is a *vector of weight coefficients*. For each new problem, or even particular set of data for this problem, there is a question of how to select the vector C^T. The program can automatically learn C^T while solving the given problem or a set of examples. The learning problem can be formulated as follows. On the basis of the already searched part of the solution tree, and the information gathered from the previously solved examples of the same set, select such C^T that if a subsequent search of the same tree were executed, the program would directly extend the branch leading to the best of the solutions selected until now, opening only those nodes, which are on the branch from the initial state of the tree.

Let us denote by N any node on this path, except the solution, and by NN one of its successors on this path. Let O(N) be the operator transforming N to NN on the best branch from all the previously searched branches, and let $O_i(N)$ stands for all other operators in N.

To solve the formulated above problem we need to select such vector C^T that the following set of inequalities be satisfied:

$$(\forall\ N)\ [C^T \cdot F(O(N)) > C^T \cdot F(O_i(N))] \tag{3.1}$$

or, after transformation

$$(\exists\ N)\ [C^T(F(O(N)) - F(O_i(N))) > 0]\ . \tag{3.2}$$

Such set is usually inconsistent, so the problem of learning is reformulated to the problem of selecting C^T which satisfies the greatest possible number of inequalities from the above set.

For illustration, let us consider a two-dimensional case (Fig. 3.1a).

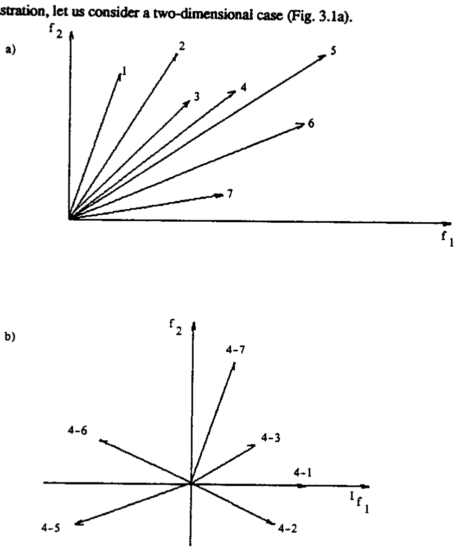

Fig. 3.1. Feature vectors and differences for a two-dimensional space.

Let us assume that in some node there are 7 operators, for which the quality vectors can be represented as in Fig. 3.1a. Let us also assume that after finding the next solution the vector 4 is best. Using (3.2) we create then the difference vectors $(F(O) - F(O_i))$. We can observe that no vector C exists in Fig. 3.1b that would satisfy the set of equations (3.2).

Let us, therefore, reformulate the learning problem in the following manner: Select such a vector C for which as many as possible of the inequalities from the set of equations (3.2) are satisfied. As it can be noticed, this problem is equivalent to the problem of calculating such a *hyperplane* running across the coordinates system center, that as many as possible of the differences $(F(O(N)) - F(O_i(N))$ are located on one side of this hyperplane, and as few as possible on the other side. In the case of n-dimensional space this problem is time consuming to solve, so we have implemented an approximate solution according to [59]. A hyperplane will be considered optimal, when $\sum_{i=1} C^T \cdot D_i = max$, where $|C| = 1$ and

$$D_i = \frac{F(O) - F(O_i)}{|F(O)) - F(O_i)|} \tag{3.3}$$

is a *normalized vector of differences*.
The normalized vector

$$C = \frac{\sum_{i=1}^{n} D_i}{|\sum_{i=1}^{n} D_i|} \tag{3.4}$$

is taken as a solution. Whenever new solution is found or when a backtrack occurs, the new inequalities are added to the learning procedures databases and the Learning Schemes #1, 2, and/or 3 are called in order to update the values of respective vectors C, using the above formulas.

An algorithm using the above principles is very fast and gives satisfactory results. Application of this type of learning increases the time (in the learning phase) by about 20% - 30 %, but reduces the tree size by about 15%- 20 %.

3.2. EXAMPLE OF APPLICATION. THE SET COVERING PROBLEM.

This problem is widely encountered in logic design (among others, in PLA minimization [28], test minimization [28], multilevel design [34]). The covering problem is represented in a form of k × n binary matrix. In addition, a cost is associated with each row. Value 1 at the intersection of row r_i and column c_j means that row r_i covers column c_j. This can be described as: $(r_i, c_j) \in COV \subset R \times C$, or $COV(r_i, c_j)$. A set of rows which cover all the columns and have a minimal total cost should be found.

Problem formulation.

1. **Given:**
   a) the set $R = \{ r_1, r_2, ..., r_k \}$ (each r_i is a row in the table);
   b) the set $C = \{ c_1, c_2, ..., c_n \}$ (each c_j is a column in the table);
   c) the costs of rows $f1(r_j)$, $j = 1, ..., k$;
   d) the relation of covering columns by rows is $COV \subset R \times C$.
2. **Find:**
   Set $SOL \subset R$;
3. **Which fulfills the condition:**
   $$(\forall c_j \in C)(\exists r_i \in SOL)[COV(r_i, c_j)].$$
4. **And minimizes the cost function:**
   $$f2 = \sum_{r_i \in SOL} f1(r_i).$$

It follows from the above formulation that the state-space $S = 2^R$. This means that $SOL \subseteq R$. Hence, all the subsets of a set are being sought. Then the standard generator, called T_1, that generates all the subsets of a set is selected.

The previously mentioned relation RE on the set S × S can be found for this problem and used to reduce searching by applying using the appropriate search method. It can be defined as follows: $s_1 \text{ RE } s_2 \Longleftrightarrow s_2 \subseteq s_1$. Therefore, when a solution is found a cut-off occurs in the respective branch.

For each element $c_j \in C$ there exists an $r_i \in SOL$ such that their relation COV is met, what means that the predicate $COV(r_i, c_j)$ is fulfilled. In such a case, the cost function F is the total sum of f1 costs of rows from the set SOL.

A tree search method and strategy for this problem will be defined below.

1) Initial node N_0:

$(QS, GS, AS, F) := (\varnothing, \{r_k \in R \mid COV(r_k, c_1)\}, C, 0)$
The first element of C is denoted by c_1 above.

2) Operator:

```
O(N, $r_i$) = [
    QS(NN) := QS(N) $\cup$ {$r_i$},
    AS(NN) := AS(N) − {$c_j \in C \mid COV(r_i, c_j)$},
    $c_j$ := the first element of AS(NN),
    GS(NN) := {$r_k \in R \mid COV(r_k, c_j)$},
    F(NN) := F(N) + $f_1(r_i)$ ].
```

3) Solution condition (cut-off type):

$p_1(NN) = (AS(NN) = \varnothing)$.

4) The following code is declared in a Strategy Describing Subroutine called "Actions on the Selected Node."

If $AS(N) \not\subseteq \{ c_j \in C \mid (\exists\ r_i \in GS(N)) [COV(r_i, c_j)] \}$ **then** $GS(N) := \varnothing$;

that means that the cut-off is done by clearing set GS(N) when the set of all the columns covered by the available descriptors from GS(N) does not include the set AS(N) of columns to be covered.

Quality Functions.

Serious advantages result from the introduction of heuristic functions that direct the order in which the tree is extended. The introduction of such functions will not only find the optimal solution quicker but also speed up the proof of its optimality. This is due to more extensive application of the cutting-off property; the search is less extensive when the optimal solution is found sooner.

5) Quality function for nodes:

$$QF(NN) = F(NN) + \hat{h}(NN), \tag{3.5}$$

where

$$\hat{h}(NN) = CARD(AS(NN)) \cdot CARD(GS(NN)) \cdot K \tag{3.6}$$

and

$$K = \frac{\sum_{r_i \in GS(NN)} f_1(r_i) \cdot CARD\{c_j \in AS(NN) \mid COV(r_i,c_j)\}}{(\sum_{r_i \in GS(NN)} CARD\{c_j \in AS(NN) \mid COV(r_i,c_j)\})^2} \tag{3.7}$$

Function $\hat{h}$ defined in this way is relatively easy to calculate. As proven in the experiments, it yields an accurate evaluation of the real distance h of NN from the best solution. It is calculated as an additional coordinate of the node's vector.

6) The quality function for operators is defined by the formula:

$$q^{NN}(r_i) = c_1\ f_1(r_i) + c_2\ f_2(r_i) + c_3\ f_3(r_i), \tag{3.8}$$

where f_1 has been defined previously as the cost function of rows,

$$f_2(r_i) = CARD\ c_j \in AS(NN) \mid COV(r_i, c_j), \tag{3.9}$$

$$f_3(r_i) = \tag{3.10}$$

$$\frac{1}{f_2(r_i)} \sum_{j=1}^{n} CARD\{r_e \mid c_j \in AS(NN)\ \&\ COV(r_i,c_j)\ \&\ r_e \in GS(NN)\ \&\ COV(r_e, c_j)\}$$

where n is a number of columns and c_1, c_2, c_3 are the arbitrarily selected weight coefficients. Function $f_3(r_i)$ defines the "resultant usefulness factor of the row" r_i in node NN. Let us assume that there exist k rows covering some column in the set GS(NN). The usefulness factor of each of these rows with respect to this column is k. When k = 1, the descriptor is *indispensable* (or with respect to Boolean minimization, the corresponding prime implicant is *essential*). The *resultant usefulness factor of the row* is the arithmetical average of the *column usefulness factors* of all the columns covered by it. Then, an instruction is added in the Operator subroutine to sort the descriptors in GS(NN) according to the nonincreasing values of the quality function for descriptors q^{NN}.

7) Relations. Sections of code are also defined that check the ***global*** and ***local equivalence*** and ***domination relations*** in descriptors, as well as indispensability of descriptors [44,65].

Experimental Results.

It has been found that the application of each of the equivalence conditions, dominance conditions, or indispensable conditions in the covering problem reduces the search space by about 2 to 3 times. The joint application of all the conditions brings about a reduction of approximately 50 to 200 times of the generated space. The influence of learning method on solution efficiency has been investigated. The computation time was increased in the learning phase by 20% to 30%. The vector of coefficients obtained in this learning was: C = [-0.85, 0.25, -0.35]. This vector brings next 15% - 20% decrease in the generated solution space size, as compared with the initial vector: C = [-1, 0, 0].

4. THE SECOND METHOD. LEARNING THE STOPPING MOMENT.

The stopping process can be explained using an *"improvement curve"* in a diagram where the x-axis is a solution time expressed as a number of expanded nodes and the y-axis is a value of the cost function of the best of the solutions found until then (Fig. 4.1).

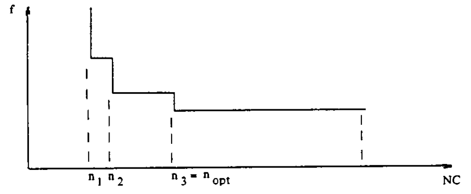

Fig. 4.1. Example of the dependency of the cost function on the number of expanded nodes (The Improvement Curve).

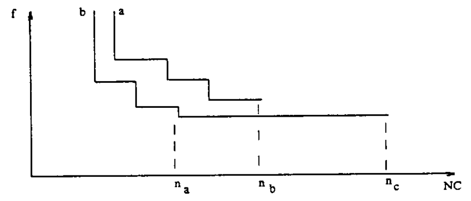

Fig. 4.2. Comparison of improvement curves : a) without learning quality function coefficients, b) with learning quality function coefficients.

The solution process can be in this case terminated after expanding n_3 nodes, while the further expansion will not bring any cost function improvements. A question can be thus raised *"can one construct a system that would learn to predict the effects (or, specifically, lack of effects) of the further solution space expansion ?"*. A positive answer to this question would additionally improve the solution space method efficiency.

A new approach to this problem will be proposed below. It takes the following assumptions:

1. There exist combinatorial problems for which for all examples from some set of data the improvement curve shapes are similar.
2. There exists a *statistical dependency* (proportionality) between the number of nodes which have to be expanded in order to find successive, better solutions, i.e., n_1 $^-n_2$ $^-n_3$,.... (See Fig. 4.1.).

In order to establish this proportionality, a sufficiently large number of examples must be tested. Also, it is necessary to introduce certain "normalization" that would make possible comparison of various improvement curves.

Let us introduce the concept of the *dimension of the problem with the respect to the i-th improvement*, or *i-th dimension* for short as a number of nodes that must be expanded to obtain the i-th cost function value improvement. In Fig. 4.1 the dimension with respect to the first improvement is n_1, with respect to the second is n_2, and so on. For the given search strategy, the dimension sequence determines a unique *improvement curve* for this problem. Various problems can have sequences of dimensions of various lengths.

Solution of each example by a computer creates some number of pairs

$$(n_i, n_{opt}) \tag{4.1}$$

In the discussed example the pairs (n_1, n_3), and (n_2, n_3) are created. Each pair is reduced to a standard dimension N, by multiplying both its coordinates by a factor of N/n_i. The following pairs are obtained

$$(N, \frac{n_{opt} \cdot N}{n_i}), \tag{4.2}$$

where N is a fixed coefficient. The second coordinate of each pair determines the standard number of nodes N_{oi} that results from the i-th dimension and which also has to be expanded in order to find the optimal solution. By disposing several improvement curves for many examples, one can calculate average standard numbers of nodes $\overline{N_{oi}}$ as weighted averages, while the weight of each result N_{oi} grows with the value of the i-th dimension n_i. It is based on the assumption that with an increase of example dimension the results obtained from the example are more reliable i.e., are characterized by a smaller expected standard deviation. Hence,

$$\overline{N_{oi}} = \frac{\sum_{j=1}^{k_i} N_{oij} \cdot n_{ij}}{\sum_{j=1}^{k_i} n_{ij}} = N \cdot \frac{\sum_{j=1}^{k_i} N_{jopt}}{\sum_{j=1}^{k_i} n_{ij}} \tag{4.3}$$

where k_i is a number of already solved examples that have the i-th dimension. The standard deviations of the obtained results are as follows:

$$\sigma_i = \sqrt{\frac{\sum_{j=1}^{k_i} (N_{oij} - \overline{N_{oi}})^2 \cdot n_{ij}}{\sum_{j=1}^{k_i} n_{ij}}} \tag{4.4}$$

In order to assure sufficiently high probability that the space expansion would be not terminated before obtaining the optimal solution, a width of half-interval of confidence with some coefficient is added to the calculated values of $\overline{N_{oi}}$:

$$N_{imax} = \overline{N_{oi}} + m \cdot \sigma_i \tag{4.5}$$

Coefficient m=3 assures the probability of 99.7% assuming the normal probability density of the results. Therefore, N_{imax} determines the maximal standard number of nodes - calculated with respect to the i-th dimension - which should be expanded in order to find the optimum solution.

The calculated values are applied as follows. Having found the first solution by expanding n_1 nodes the maximum number of nodes n_{1max} (already not the standard one) is calculated,

$$n_{1max} = \frac{N_{1max} \cdot n_1}{N} \tag{4.6}$$

If a better solutions is not found after expansion of n_{1max} nodes the system stops. However, if after expansion of $n_2 < n_{1max}$ nodes a better solution is found then the value of n_{2max} is calculated according to the formula:

$$n_{2max} = \frac{(k_1 \cdot n_{1max} + k_2 \cdot \frac{N_{2max} \cdot n_2}{N})}{(k_1 + k_2)} \tag{4.7}$$

where k_1 and k_2 determine number of examples which have been used to calculate the values of N_{1max} and N_{2max}. The system behavior after generating next solutions is based on the same principles.

The method described above bypasses several difficulties related to the normalization of several individual properties of the problems, such as different numbers of improvement curve steps, or various values of quality function decrements. It can be observed that this method collaborates with the one to learn quality functions coefficients from section 3 and gives better results when the other produces better results.
Fig. 4.2. presents schematically the effect that can be obtained by concurrent application of the both learning methods. Without learning, n_c nodes should be expanded. By applying the learning method from this section one would need n_b nodes while applying additionally the method from section 3 one would need only n_a nodes.

4.1. EXAMPLE OF APPLICATION. THE LINEAR ASSIGNMENT PROBLEM.

This problem, similarly to the previous one, has *several* CAD applications including VLSI chip placement and Finite State Machine assignment.

Problem Formulation.

Given is n machines and n workers and the productivity of the worker i on machine j is denoted by w_{ij}. The assignment of machines to workers is sought that maximizes the total productivity of all workers. The assignment matrix $W_{nxn} = [w_{ij}]$ is given from which n elements must be selected in such a way that any two of them are taken from a different row and column and the sum of the elements is maximum. Since MULT-II looks for a minimum, the problem is reformulated as below.

Each object has n properties. The value of property x_i determines the column number from which the element from row i has been selected. In order to simplify the program the elements of the matrix are selected in the following order: first an element from row one is chosen, next an element from row two, and so on. In this case the operator is a number specifying only the second coordinate of the selected element w_{ij}, and the first coordinate is specified by the depth of the node in the tree. Secondly, the list AS(N) of nodes specifying the state of the object in node N becomes in this case unnecessary, while it would contain the set of non-selected rows, and this set is already specified by the depth of the node in the tree. For instance, in such description, the solution <3, 1, 2> for a 3 × 3 problem means that elements w_{13}, w_{21}, w_{32} have been selected from matrix W. The description of the **initial state** is as follows:
QS(0) = ∅,
GS(0) = set of numbers of all columns.

Operation of **operator** $O(N, r_i)$ can be described as follows:
QS(NN) = QS(N) ∪ { r_i },
GS(NN) = GS(N) − { r_i } .
The solution condition is GS(NN) = ∅.

The specification of the **quality function** values for operators is in this case obvious - they are the negated values of the respective elements w_{ij} of the matrix. The value of cost function for states is equal to the sum of negated costs of operators on the path from the initial state to the given state. **A heuristic quality function for nodes,** that specifies the minimal cost of the path from the given node to the solution is also useful. It is defined as follows:

$$\hat{h} = \sum_i w_{i,\min}$$

where summing is extended over the non-selected rows, and $w_{i,\min}$ denotes the least element of the i-th row among the elements from the non-selected columns.

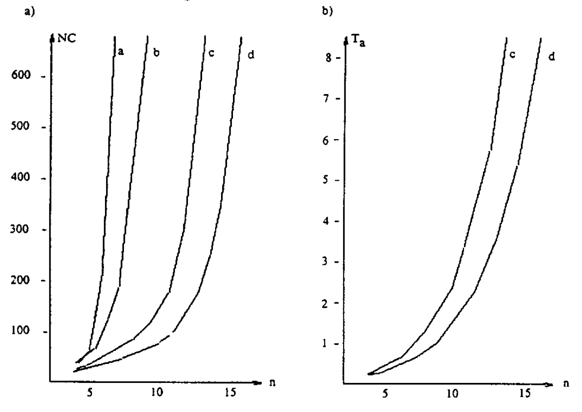

Fig. 4.3. Dependence of a) number of nodes and b) processing time on the problem size for strategies: a - Breadth First, b - Depth First, c - Branch-And-Bound, d- Ordered Search.

Experimental Results.

The influence of the search strategy on the solution efficiency has been investigated. Fig. 4.3. presents the statistical dependency of the solution time and the number of nodes expanded on the problem dimension, for various strategies. The Ordered-Search strategy has been most efficient for this problem. It is due to the usage of a very accurate heuristic function which permitted for cutting branches at small depth of the tree.

This problem is one for which the Stopping Learning Method gives good results, because it is characterized by a improvement curve with sufficiently large number of steps. (For instance about 4-10 steps for Branch-And-Bound strategy and n < 14). Additionally, it was verified that about 30% of the solution tree was expanded "redundantly" after finding the first optimal solution. Using the above method permitted to decrease this part of tree by about 40%. For instance, when the entire space generated by the system included, on average, 600 nodes and the optimal solution was found after 420 nodes, by using this method it was sufficient to expand on 530 nodes.

5. A COMPLETE EXAMPLE: SYNTHESIS OF EASILY TESTABLE MULTI-VALUED INPUT EXOR/AND TREES FOR INCOMPLETELY SPECIFIED LOGIC FUNCTIONS.

In this section we will present the method to learn the best *Quality Function For Nodes* for a tree search algorithm that finds the optimum Multi-Valued Input Generalized Reed-Muller Tree (MGRMT). Such circuits are new generalizations of Generalized Reed-Muller Forms. In the first part of this section the concept of MGRMs will be introduced.

5.1. Multi-Valued Input Generalized Reed Muller Forms.

Reed-Muller forms for binary logic are canonical exclusive sum of products of *positive* (non-complemented) input variables. Two extensions of Reed-Muller (RM) forms have been investigated for completely specified Boolean functions. *Fixed-polarity Generalized Reed-Muller Forms (GRM)* are of similar form but allow the inputs to the products to be both positive and negative (complemented). Each variable has, however, to stand in either positive or negative form, but not in both [13]. Such forms are also canonical what means that only one such form exists for every polarity of variables (there are 2^n such polarites for n binary inputs). When all restrictions on input variables polarities are removed (each variable can be positive and negative in the same exclusive sum of products expression) one gets the mixed polarity *Exclusive Sum of Product (ESOP)* expression that is not a canonical form [20].

A multiple-valued input, two-valued output, incompletely specified switching function f *(multiple-valued function,* for short) is a mapping $f(X_1, X_2, \cdots, X_n)$: $P_1 \times P_2 \times \cdots P_n \rightarrow B$, where X_i is a *multiple-valued variable*, $P_i = \{0, 1, ..., p_i - 1\}$ is a *set of truth values* that this variable may assume, and $B = \{0, 1, -\}$ (- denotes a *don't care value).* This is a generalization of an ordinary n-input switching function f: $B^n \rightarrow B$.

Definition 5.1. For any subset $S_i \subseteq P_i$, $X_i^{S_i}$ is a *literal* of X_i representing the function such that

$$X_i^{S_i} = \begin{cases} 1 & \text{if } X_i \in S_i \\ 0 & \text{if } X_i \notin S_i. \end{cases}$$

S_i will be called a *polarity of literal* $X_i^{S_i}$.

Definition 5.2. A product of literals, $X_1^{S_1} X_2^{S_2} \ldots X_n^{S_n}$, is referred to as a *product term* (also called *term* or *product* for short). A sum of products is denoted as a (multi-valued input) *sum-of-products expression (SOPE).*

Switching functions with multiple-valued inputs, two-valued outputs, find several applications in logic design, pattern recognition, and other areas. In logic design, they are primarily used for the minimization of PLAs that have 2-bit decoders on the inputs. A Programmable Logic Array (PLA) with r-bit decoders directly realizes a SOPE of a 2^r-valued input, two-valued output, function [54-57]. Such decoders can be also used in any other realization of the logic with multi-valued inputs, like ESOPs [45] and their simplified form is used in the "fixed-polarity" *Multi-Valued Input* Generalized Reed-Muller Trees (MGRMTs) that will be presented below.

In the case of binary-input logic, each variable from a GRM form can have one of two possible *polarities*, 0 or 1. Let us observe that if two polarities were available for even a single variable, then the expression would be not canonical, for instance x and $1 \oplus \bar{x}$ would represent two different expressions for the same function f(x) = x. In a logic with p-valued inputs one has p values for which literals with arbitrary $p-1$ polarities $x^{i_1}, x^{i_2}, x^{i_3}, \cdots, x^{i_{p-1}}$ can be taken, for instance by removing x^2 one gets the following *allowed literals:* $x^0, x^1, x^3, x^4, ..., x^{p-1}$. It can be proven that for GRM expansion one can take any p-1 values that form an orthogonal *polarity matrix.* For instance for p=4 one can have the following set of allowed literals: $\{X^{012}, X^{013}, X^{023}\}$, which is described by a polarity matrix

$$\begin{bmatrix} 1110 \\ 1101 \\ 1011 \end{bmatrix}$$

Some examples of such sets of allowed literals for a 4-valued input variable X are: $\{X^{012}, X^{013}, X^{123}\}$, $\{X^{13}, X^{23}, X^{3}\}$, $\{X^{02}, X^{01}, X^{012}\}$, $\{X^{13}, X^{23}, X^{123}\}$. Let us observe that a complete set of *allowed polarities* can be obtained from other ones by complementing and exoring rows of the polarity matrix. All such forms have very good testability properties [50], especially the form corresponding to the last set from the above example leads to realizations that have a very simple decoder (a single OR gate). It is also easy for the test generation (using an adaptation of the method from [50]), and minimizes the total layout area.

Definition 5.3. The set of *allowed polarities* for a p-valued variable X is a set with $p-1$ elements whose corresponding polarity matrix is orthogonal. *Allowed literal* is a literal with allowed polarity.

Definition 5.4. By a *Multiple-Valued Input Generalized Reed-Muller Form (MGRM)* one understands an exclusive sum of products in which all literals are allowed.

This is a generalization of the concept of a GRM form, where each variable can be complemented or not complemented, but it cannot stand in both forms. It results from the above definitions that the MGRM class is properly included in the ESOP class. The concept of the MGRM has not been introduced in the literature on the subject yet. The introduced above concepts and definitions will be now illustrated with an example.

Example 5.1. Assuming 4-valued input variables X and Y, the expression: $f(X,Y) = 1 \oplus X^{012} Y^{012} \oplus X^{013} Y^{23} \oplus X^{123} Y^{23} \oplus X^{023} Y^{123}$ is an ESOP but it is not a MGRM form because there exists the variable X that has four different polarities, while only three polarities are allowed for it. The equivalent MGRM can be obtained by the substitution:
$f(X,Y) = 1 \oplus (1 \oplus X^{013} \oplus X^{123} \oplus X^{023}) Y^{012} \oplus X^{013} Y^{23} \oplus X^{123} Y^{23} \oplus X^{023} Y^{123}$ and in the next stage converting the above expression to exor of products form.

5.2. The Concept of Generalized Shannon Expansion and Multi-Valued Input Generalized Reed Muller Trees.

The well-known Shannon expansion for the case of Exclusive Sum of Product expansion is as follows [13]:

$$f(x_1,...,x_i,...,x_n) = \bar{x}_i \cdot f(x_1,...,x_i{=}0,...,x_n) \oplus x_i \cdot f(x_1,...,x_i{=}1,...,x_n) \qquad (5.1)$$

By applying laws $\bar{a} = 1 \oplus a$ and $a = 1 \oplus \bar{a}$ one gets: $f(x_1,...,x_i,...,x_n) =$

$$f(x_1,...,x_i{=}0,...,x_n) \oplus x_i \cdot [f(x_1,...,x_i = 0,...,x_n) \oplus f(x_1,...,x_i{=}1,...,x_n)] \qquad (5.2)$$

and $f(x_1,...,x_i,...,x_n) =$

$$f(x_1,...,x_i{=}1,...,x_n) \oplus \bar{x}_i \cdot [f(x_1,...,x_i{=}0,...,x_n) \oplus f(x_1,...,x_i{=}1,...,x_n)] \qquad (5.3)$$

In the short form:

$$f = x_i \cdot f_{x_i} \oplus \bar{x}_i \cdot f_{\bar{x}_i} = f_{\bar{x}_i} \oplus x_i \cdot [f_{x_i} \oplus f_{\bar{x}_i}] = f_{x_i} \oplus \bar{x}_i \cdot [f_{x_i} \oplus f_{\bar{x}_i}] \qquad (5.4)$$

Let us observe that these formulas have been applied by several authors for synthesis of GRM forms for completely specified functions [61,11], but they can be used for incompletely specified functions as well.

ON(f) is a set of *true cubes* of function f (cubes for which f = 1). OFF(f) is a set of *false cubes* of function f (cubes for which f = 0). All other input values are don't cares. Let us now denote:

$$ON(f_{\bar{x}}) = ON(f) \cap \bar{x}, \quad ON(f_x) = ON(f) \cap x, \qquad (5.5)$$

$$OFF(f_{\bar{x}}) = OFF(f) \cap \bar{x}, \quad OFF(f_x) = OFF(f) \cap x, \qquad (5.6)$$

In general, for multi-valued input logic:

$$ON(f_{X^S}) = ON(f) \cap X^S, \quad OFF(f_{X^S}) = OFF(f) \cap X^S, \qquad (5.7)$$

Part of equation (5.4) can be now rewritten to the form:

$$f = [ON(f),OFF(f)] = [ON(f_{\bar{x}_i}),OFF(f_{\bar{x}_i})] \oplus x_i \cdot [ON(f_{x_i} \oplus f_{\bar{x}_i}),OFF(f_{x_i} \oplus f_{\bar{x}_i})] \quad (5.8)$$

where expressions $ON(f_{x_i} \oplus f_{\bar{x}_i})$, and $OFF(f_{x_i} \oplus f_{\bar{x}_i})$ are calculated as it is described in section 5.3. Similarly other parts of (5.4) can be rewritten. ⊕ means joining of arrays of cubes (of exor types); there is no cube calculus operation executed by this symbol.

Assuming any allowed polarities, equation (5.4) can be generalized for the multiple-valued input logic. For instance, for the selected by us polarities of 4-valued variables, the expansion is:

$$f = f_{a^{012}} \oplus a^{13} (f_{a^{13}} \oplus f_{a^{012}}) \oplus a^{23} (f_{a^{23}} \oplus f_{a^{012}}) \oplus a^{123} (f_{a^{123}} \oplus f_{a^{012}}) \qquad (5.9)$$

The counterpart of equation (5.8) for equation (5.9) can be also formulated in an analogous way. For a completely specified function and a given sequence of expansion variables there exists exactly one (canonical) factorized expression (tree) created by this expansion. Such a tree will be called a *Multi-Valued Input Generalized Reed Muller Tree (MGRMT).* In the case of incompletely specified functions the tree is not canonical. The MGRMT synthesis task is to select the order of expansion variables that minimizes the complexity of this tree. By *flattening* the MGRMT one gets a MGRM form. Flattening is done by repeated application of the law: a(b ⊕ c) = ab ⊕ ac.

5.3. Search for an Optimum MGRMT.

Below we will present the program GERMANISM-MV (GEneralized Reed Muller And Not Incompletely Specified function Minimizer for Multi-Valued logic) that finds a MGRMT. GERMANISM-MV starts from the arrays ON and OFF of *disjoint cubes of function f.* Such cubes are created using program DISJOINT [15,16]. GERMANISM-MV uses subroutine EXPAND(ON(f) ; OFF(f) ; X_i ; POLARITY) to find expansion with respect to a single multi-valued input variable. Program EXPAND expands f = [ON(f) , OFF(f)] for variable X_i assuming polarity POLARITY of this variable. For simplification we will present here only the binary case. We will denote: $g_{x_i} = f_{x_i} \oplus f_{\bar{x}_i}$. The symbol # means the *non-disjoint sharp operation* [14]. By *type 1/0 minterm* we understand a situation when there is a 1 *(true minterm)* in $f_{\bar{x}}$ and a 0 *(false minterm)* in f_x that are adjacent with respect to variable x. Similarly a 1/- cube is a cube where all respective minterms in $f_{\bar{x}}$ are true and all minterms in f_x are don't cares.

EXPAND(ON(f) ; OFF(f) ; X_i ; POLARITY).

1. From ON(f) and OFF(f) calculate the arrays: ON(f_{x_i}), OFF(f_{x_i}), ON($f_{\bar{x}_i}$), and OFF($f_{\bar{x}_i}$).
2. Calculate the following arrays of cubes: S1 = ON(f_{x_i}) ∪ OFF(f_{x_i}), S2 = ON($f_{\bar{x}_i}$) ∪ ON($f_{\bar{x}_i}$), S3 = ON($f_{\bar{x}_i}$) ∩ OFF(f_{x_i}), S4 = ON($f_{\bar{x}_i}$) ∩ ON(f_{x_i}), S5 = OFF($f_{\bar{x}_i}$) ∩ ON(f_{x_i}), S6 = OFF($f_{\bar{x}_i}$) ∩ OFF(f_{x_i}), S7 = ON($f_{\bar{x}_i}$) # S1, S8 = OFF($f_{\bar{x}_i}$) # S1, S9 = ON(f_{x_i}) # S2, S10 = OFF(f_{x_i}) # S2.
   Comment. The meaning of the sets is: S3 = 1/0, S4 = 1/1, S5 = 0/1, S6 = 0/0, S7 = 1/-, S8 = 0/-, S9 = -/1, S10 = -/0.
3. *Processing of 1/1 type cubes.*
   ON($f_{\bar{x}_i}$) := ON($f_{\bar{x}_i}$) ⊕ $S3_{\bar{x}_i}$, ON(f_{x_i}) := ON(f_{x_i}) ⊕ $S3_{x_i}$, ON(g_{x_i}) := ON(g_{x_i}) ⊕ S3.
4. *Processing of 0/1 type cubes.*
   ON($f_{\bar{x}_i}$) := ON($f_{\bar{x}_i}$) ⊕ $S4_{\bar{x}_i}$, ON(f_{x_i}) := ON(f_{x_i}) ⊕ $S4_{x_i}$, OFF(g_{x_i}) := OFF(g_{x_i}) ⊕ S4.

5. *Processing of 1/0 type cubes.*
   $OFF(f_{\bar{x}_i}) := OFF(f_{\bar{x}_i}) \oplus S5_{\bar{x}_i}$, $OFF(f_{x_i}) := OFF(f_{x_i}) \oplus S5_{x_i}$, $ON(g_{x_i}) := ON(g_{x_i}) \oplus$ S5.
6. *Processing of 0/0 type cubes.*
   $OFF(f_{\bar{x}_i}) := OFF(f_{\bar{x}_i}) \oplus S6_{\bar{x}_i}$, $OFF(f_{x_i}) := OFF(f_{x_i}) \oplus S6_{x_i}$, $OFF(g_{x_i}) := OFF(g_{x_i}) \oplus$ S6.
7. *Processing of 1/- type cubes.*
   $ON(f_{\bar{x}_i}) := ON(f_{\bar{x}_i}) \oplus S7_{\bar{x}_i}$, $ON(f_{x_i}) := ON(f_{x_i}) \oplus S7_{x_i}$, $OFF(g_{x_i}) := OFF(g_{x_i}) \oplus$ S7. (as in 1/1 cubes - S4).
8. *Processing of 0/- type cubes.*
   $OFF(f_{\bar{x}_i}) := OFF(f_{\bar{x}_i}) \oplus S8_{\bar{x}_i}$, $OFF(f_{x_i}) := OFF(f_{x_i}) \oplus S8_{x_i}$, $OFF(g_{x_i}) := OFF(g_{x_i}) \oplus$ S8. (as in 0/- cubes - S6).
9. *Processing of -/1 type cubes.*
   $ON(f_{\bar{x}_i}) := ON(f_{\bar{x}_i}) \oplus S9_{\bar{x}_i}$, $ON(f_{x_i}) := ON(f_{x_i}) \oplus S9_{x_i}$, $OFF(g_{x_i}) := OFF(g_{x_i}) \oplus$ S9. (as in 1/- cubes - S7).
10. *Processing of -/0 type cubes.*
    $OFF(f_{\bar{x}_i}) := OFF(f_{\bar{x}_i}) \oplus S10_{\bar{x}_i}$, $OFF(f_{x_i}) := OFF(f_{x_i}) \oplus S10_{x_i}$, $OFF(g_{x_i}) := OFF(g_{x_i})$ $\oplus$ S10. (as in 0/- cubes - S8).
12. If $ON(f_{\bar{x}_i}) = \phi$, assume $f_{\bar{x}_i} \equiv 0$. If $OFF(f_{\bar{x}_i}) = \phi$, assume $f_{\bar{x}_i} \equiv 1$. If $ON(f_{x_i}) = \phi$, assume $f_{x_i} \equiv 0$. If $OFF(f_{x_i}) = \phi$, assume $f_{x_i} \equiv 1$. If $ON(g_{x_i}) = \phi$, assume $g_{x_i} \equiv 0$. If $OFF(g_{x_i}) = \phi$, assume $g_{x_i} \equiv 1$.
13. If POLARITY = 1 then

```
 begin
 ON(f) := $ON(f_{\bar{x}_i}) \oplus x_i \cdot ON(g_{x_i})$;
 OFF(f) := $OFF(f_{\bar{x}_i}) \oplus x_i \cdot OFF(g_{x_i})$
 end
else
 begin
 ON(f) := $ON(f_{x_i}) \oplus \bar{x}_i \cdot ON(g_{x_i})$;
 OFF(f) := $OFF(f_{x_i}) \oplus \bar{x}_i \cdot OFF(g_{x_i})$
 end;
```

14. If ON(f) = ϕ for any function $f_{x_i}, f_{\bar{x}_i}, g_{x_i}$
    then substitute the logic 0 in the respective place.
15. If OFF(f) = ϕ for any function f, f_{x_i}, g_{x_i}
    then substitute the logic 1 in the respective place.
16. Simplify the expressions for ON(f) and OFF(f) on the top level using basic Boolean laws.
    End

Procedure EXPAND returns an expression of f that is either a pair of cube arrays [ON(f), OFF(f)] or a simplified expression that does not need further processing.

Similarly one can define procedure EXPAND for multi-valued input variable X_i and the list of sets of allowed polarities as a value of POLARITY. In our case the variables are 4-valued and the set of allowed polarities is {13,23,123}, which in this particular case is a single value of the variable POLARITY.

The program GERMANISM is a tree search program which operates on function f represented in the form of a tree (a deeply parenthesised list). Such list is a value of coordinate QS(N). The leaves of this tree are either non-expandable (constant 1, literals and simplified functions f^r) or expandable (pairs of sets [ON(f^r) , OFF(f^r)] for respective functions). An operator consists in applying expansion of QS(N) with respect to X_i. When an expansion with respect to variable X_i is being performed, all the expandable sub-expressions (leaves) are expanded. When there are no more expandable leaves the MGRMT is complete, the cost of the solution F(N) is calculated, the solution (if better) is obtained, and backtrack in the tree search occurs. For each partially expanded MGRMT tree (each N of the search tree) two functions are calculated, cost function F(N) that calculates the cost of the already constructed part of the circuit, and the learning-based Quality Function for Nodes QF(N), that evaluates heuristically nodes of the search tree. Whenever the partial cost F(N) exceeds the value of B, the respective branch of the search tree is cut-off.

The Shannon-expansion based method shown here is very general and also finds applications to MGRMs, ESOPs, and the corresponding trees as well as Directed Acyclic Graphs (DAGs), for incompletely specified functions represented as arrays of ON and OFF cubes. Separate variants for both binary and multi-valued input logic functions have been created. The algorithms' heuristics are the selection of the expansion variable and of its polarities. Either a single variable and its polarities are chosen for the entire function (the MGRMs and respective multi-level networks), or in each "subtree" independently (the ESOPs and respective trees and DAGs).

All the above methods give optimal results for functions that are not *sparse*. For instance, $f(x_1, \underline{x_2}, \underline{x_3}) = x_1 x_2 x_3 \oplus x_1 x_2 x_3$ is an optimal ESOP for function $f(x_1, x_2, x_3) = x_1 x_2 x_3 + x_1 x_2 x_3$ but the "minimal" ESOP tree is: $f(x_1, x_2, x_3) = \underline{x_2}\,\underline{x_3} \oplus x_1 (\underline{x_3} \oplus x_2)$, which after flattening gives: $f(x_1, x_2, x_3) = x_2 x_3 \oplus x_1 x_3 \oplus x_1 x_2$, This function is an example of the sparse function, one which has few minterms having larger Hamming distance between them. Special methods of preprocessing of such functions have to be used [11,21].

5.4. Learning the Quality Function for Nodes.

The QF(N) function is of the form:

$$QF(N) = \tag{5.10}$$

$$\sum_{EXPN} [c_1\, cost(f_{a^{012}}) + + c_2 [cost(f_{a^{13}} \oplus f_{a^{012}}) + cost(f_{a^{23}} \oplus f_{a^{012}})] + c_3\, cost(f_{a^{123}} \oplus f_{a^{012}})]$$

where:

- EXPN is a set of all expandable leaves f^r in QS(N), $f^r = f^r{}_{a^{012}}, f^r{}_{a^{13}}, f^r{}_{a^{23}}, f^r{}_{a^{123}}$.
- c_j are coefficients for learning, and

$$cost(f_{a^t}) = \begin{cases} 0 & \text{if } f_{a^t}=0 \\ 1 & \text{if } f_{a^t}=1 \\ cost1(ON(f_{a^t})) + cost1(OFF(f_{a^t})) & \text{elsewhere} \end{cases} \tag{5.11}$$

where t = 012, 13, 23, 123.

In order to choose variable X_i in such a way that X_i has the least minimal number of sub-trees, the method similar to one from [10] is used that has been modified to make learning possible.

Definition 5.5. For each variable X_i, we define

$$L(X_i) = \begin{cases} 1+|\cup(S_i - I)| & \text{if } X^I{}_i \textit{ is a product}, I \subseteq P_i \\ |\cup S_i| & \text{if } \textit{no literal of } X_i \textit{ is a product} \end{cases} \tag{5.12}$$

where |S| represents the number of elements in S and the union is taken over all products. If X_i does not appear in a product then $S_i = P_i$ for that product.

Definition 5.6. The *delta function* on X_i, $\delta(X_i)$, is given by

$$\delta(X_i) = \begin{cases} 1 & \text{if } L(X_i) \neq p_i \\ 0 & \text{otherwise.} \end{cases} \tag{5.13}$$

Definition 5.7. The *branch function* is defined by $b(X_i) = L(X_i) + \delta(X_i)$.

Definition 5.8. An n-tuple $(a_1, \dots, a_n)$ where $0 \leq a_i < p_i - 1$, is called an *n-term* of function f iff $f(a_1, \dots, a_n) = 1$.

Definition 5.9. For each i, $1 \leq i \leq n$ and each j, $0 \leq j \leq p_i - 1$, *freq(i, j)* is defined to be the number of occurences of n-terms with *i*-th coordinate that are equal to the value of j.

The *cost1* function for both ON and OFF cube arrays is:

$$cost1(array, X_i) = \alpha_1 L(X_i) + \alpha_2 \delta(X_i) + \alpha_3 \gamma(X_i) + \alpha_4 \zeta(X_i) + \alpha_5 \psi(X_i) \tag{5.14}$$

where:

- $\gamma(X_i)$ is 1 if X_i is a product (cube of array) with the fewest number of literals, or 0 otherwise.
- $\zeta(X_i)$ is the number of products in which X_i appears.
- $\psi(X_i)$ is 1 if *freq(i, j)* is the maximum among all *freq(i', j')* for all *i', j'*.
- α_j are learning coefficients.

6. CONCLUSION

Three new variants of the evaluation function coefficient learning method have been formulated and implemented. The method was applied to several new problems and for most of them gave practically useful results. A new Stopping Learning Method has also been proposed and found useful for a class of applications. We plan to use learning methods for more logic synthesis and CAD applications, especially those problems for which there is no human problem solving experience and proven heuristics. Such problems include the design of various generalization of the Generalized Reed Muller Forms for Multi-valued Logic [11,13,20,21,45] and spectral transform based logic synthesis [16], since the rules and heuristics applied in them are often counter-intuitive. It would be then interesting to see whether some good sets of operators and rules can be learned automatically. The presented methods are especially useful when many similar problems are solved in a sequence, for instance, several calls to a Boolean minimizer during the state assignment [32], or to the linear assignment subroutine during VLSI placement [1].

In the future we consider to use various Neural Network routines in MULT-II. In the new variant we work on the development of learning process in a dedicated *Learning Co-processor* - a Motorola DSP board that will work in parallel with the main tree-searching processor. Therefore, disregarding small communication overheads, it will not be any speed degradation in learning phases.

7. LITERATURE

[1] Akers, S.B., On the use of linear assignment algorithm in module placement, *25 Years of Electronic Design Automation*, 1988, pp. 218-223.

[2] Bagchi, A., Mahanti, A., Search Algorithms under Different Kinds of Heuristics - A Comparative Study, *JACM*, Vol. 30., 1983, pp. 1-21.

[3] Berliner, H., On the Construction of Evaluation Functions for Large Domains, *Proceedings IJCAI-79*, Tokyo, Japan, 1979, pp. 53-55.

[4] Bertolazzi, P., Sassano, A., A class of polynomially solvable set-covering problems, *SIAM J. Discrete Math.*, Vol. 1., No. 3., August 1988, pp. 306-316.

[5] Bertolazzi, P., Sassano, A., An O(mn) algorithm for regular set-covering problems, *Theor. Comput. Sci.*, Vol. 54., 2,3, Oct. 1987, pp. 237-247.

[6] Besslich, Ph.W., Heuristic minimization of MVL functions: a direct cover approach, *IEEE Trans. Comput.*, Vol. C-35, No. 2, pp. 134 - 144, 1986.

[7] Brayton, R.K., Hachtel, G.D., McMullen, C.T., Sangiovanni-Vincentelli, A.L., Logic Minimization Algorithms for VLSI Synthesis, *Kluwer Academic Publishers, 1984.*

[8] Buchanan, B.G., Johnson, C.R., Mitchell, T.M., Smith, R.G., Models of Learning Systems, in: Belzer, J. (Ed.), *Encyclopedia of Computer Science and Technology, 11, Marcel Dekker*, New York, 1978, pp. 24-51.

[9] Carpenter, B., Davis, IV, N., Implementation and performance analysis of parallel assignment algorithms on a hypercube computer, *Proc. "Hypercube Concurrent Computers and Applications, Vol. 2.*, Pasadena, CA, Jan. 19-20, 1980, pp. 1231-1235.

[10] Chan, A., Using decision trees to derive the complement of a binary function with multiple-valued inputs, *IEEE Trans. Comp.*, Vol. C-36, No. 2., Febr. 1987, pp. 212-214.

[11] Csanky, L., Perkowski, M., Canonical restricted mixed-polarity exclusive sum of products and the efficient algorithm for their minimization, *Submitted to DAC'90.*

[12] Dagenais, M.R., Agarwal, V.K., Rumin, N.C., McBoole: a new procedure for exact logic minimization, *IEEE Trans. on CAD*, Vol. CAD-5, No.1., Jan. 1986, pp. 229-238.

[13] Davio, P., J.P. Deschamps, and A. Thayse, Discrete and Switching Functions. *George and McGraw-Hill*, New York, 1978.

[14] Dietmayer, D.L., Logic Design of Digital Systems. (2nd ed.) *Allyn and Bacon*, Boston 1978.

[15] Falkowski, B.J., Perkowski, M.A., An Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions, *Submitted to International Journal of Electronics*, 1989.

[16] Falkowski, B.J., Schäfer, I., Perkowski, M.A., Effective Computer Methods for the Calculation of Hadamard-Walsh Spectrum and Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions, *Submitted to IEE Proc, Pt.E*, 1989.

[17] Frenk, J., Van Houweninge, M., Kan, R., Order statistics and the linear assignment problem, *Computing*, N.Y., Vol. 39, April 1, 1987, pp. 165-174.

[18] Garfinkel, R.S. Nemhauser, G.L., Integer Programming, *Wiley*, N.Y., 1972.

[19] Griffith, A.K., A Comparison and Evaluation of Three Machine Learning Procedures as Applied to the Game of Checkers, *Artificial Intelligence*, Vol. 5., 1974, pp. 137-148.

[20] Helliwell, M., Perkowski, M., A fast algorithm to minimize multi-output mixed-polarity generalized Reed-Muller forms, *Proceedings of 25th ACM/IEEE Design Automation Conference*. 1988, Las Vegas, pp. 427 - 432.

[21] Helliwell, M., Perkowski, M., Jeske-Chrzanowska, M., An exact algorithm to minimize mixed polarity exclusive sum of products for incompletely specified Boolean functions, *Submitted to ISCAS'90.*

[22] Ho, P.M., Perkowski, M.A., Systolic Architecture for Solving Combinatorial Problems of Logic Design, *Proceedings of International Symposium on Circuits and Systems - ISCAS*, 1989, pp. 1170-1173.

[23] Hochbaum, D.S., Approximation algorithms for the weighted set covering and node covering problems, *SIAM J. Comput.*, Vol. 11, 1982, pp. 535-556.

[24] Hubert, L.J., Assignment Methods in Combinatorial Data Analysis, *Marcel Dekker Inc.*, New York/Basel, 1987.

[25] Hubert, L.J., Statistical applications of linear assignment, *Psychometrica*, Vol. 49, 1984, pp. 449-473.

[26] Hurst, S.L., The Logical Processing of Digital Signals. *Crane-Russak*, New York and Edward Arnold, London, 1978.

[27] Jonker, R., Volgenant, A shortest augmenting path algorithm for dense and sparse linear assignment problems, *Computing*, N.Y., Vol. 38., No. 4., March 1987, pp. 325-340.

[28] Kohavi Z., Switching and Finite Automata Theory. (2nd edition), *McGraw-Hill*, New York, 1978.

[29] Koopmans, T.C., Beckmann, M., Assignment problems and the location of economic activities, *Econometrica*, 25., pp. 53-76, 1957.

[30] Kuhn, H.W., The hungarian method for the assignment problem, *Naval Res. Logist. Quart.*, 2., pp. 83-87, 1955.

[31] Lawler, E.L., Lenstra, J.K., A.H.G. Rinooy Kan, D.B., Shmoys, The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization, *Wiley*, 1985.

[32] Lee, E.B., Perkowski, M., Concurrent minimization and state assignment of Finite State Machines, *Proc. IEEE Intern. Conf. on Systems, Man, and Cybernetics*, Halifax, Nova Scotia, Canada, October 9-12, 1984.

[33] Lee, K.F., Mahajan, S., A Pattern Classification Approach to Evaluation Function Learning, *Artificial Intelligence*, Vol. 36., pp. 1-25, 1988.

[34] MCNC International Workshops on Logic Synthesis, Research Triangle Park, North Carolina, 1987, 1989.

[35] Mitchell, T., Carbonell, J., Michalski, R. (eds), Machine Learning: A Guide to Current Research. (Knowledge Representation, Learning, and Expert Systems). *Kluwer Academic Publishers*, Nowell, MA, 1986.

[36] Mirchandaney, R., Stankovic, J., Using Stochastic Learning Automata for Job Scheduling in Distributed Processing Systems, *J. Parallel Distrib. Comput.*, Vol. 3., No. 4., Dec. 1987, pp. 527-552.

[37] Nilsson, N.J., Learning Machines, *McGraw-Hill*, New York, 1965.

[38] Nilsson, N.J., Problem-Solving Methods in Artificial Intelligence, *McGraw Hill*, New York 1971.

[39] Perkowski, M., "The State Space Approach to the Design of Multipurpose Problem Solver for Logic Design," *Proc. IFIP WG 5.2. Conference "Artificial Intelligence and Pattern Recognition in Computer Aided Design*, Grenoble 17-19 March, *North Holland Publ. Comp.*, Amsterdam, 1978, pp. 123- 140.

[40] Perkowski, M.A., A Systolic Architecture for the Logic Design Machine, *Proc. of ICCAD*, 1985, Santa Clara.

[41] Perkowski, M., Nguyen, N., Minimization of Finite State Machines in SuperPeg, *Proc. Midwest Symp. on Circuits and Systems*, Luisville, Kentucky, 22-24 August 1985.

[42] Perkowski, M., Minimization of two-level networks from negative gates, *Proc. Midwest Symp. on Circuits and Systems*, Lincoln, Nebraska, 1-12 August 1986.

[43] Perkowski, M., Liu, J., A System for Fast Prototyping of Logic Design Programs, *Proc. 1987 Midwest Symposium on Circuits and Systems*, Syracuse, New York.

[44] Perkowski, M., Liu, J., Brown, J., Quick Software Prototyping: CAD Design of Digital CAD Algorithms, In: G. Zobrist (Ed.) "Progress in Computer Aided VLSI Design", *Ablex Publishing Corp.*, 1989.

[45] Perkowski, M., Helliwell, M., Wu, P., Minimization of Multiple-Valued Input Multi-Output Mixed-Radix Exclusive Sums of Products for Incompletely Specified Boolean Functions, *Proc. IEEE Inter. Symp. Multiple Valued Logic*, Guangzhou, People's Republic of China, May 1989, pp. 256-263.

[46] Perkowski, M., Liu, J., Minimization of TANT networks, *Submitted to ISCAS'90.*

[47] Perkowski, M., Liu, J., Generation and optimization of Finite State Machines from parallel program graphs, *Submitted to ISCAS'90.*

[48] Pertsekas, D., The auction algorithm: a distributed relaxation method for the assignment problem, *Oper. Res.*, Vol. 14., 1-4, June 1988, pp. 105-123.

[49] Pyber, L., Clique covering of graphs, *Combinatorica*, Vol. 6., No. 4., 1986, pp. 393-398.

[50] Reddy, S.M., Easy testable realizations for logic functions, *IEEE Trans. Comput.*, Vol. C-21, pp. 1183 - 1188, 1972.

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

[52] Samuel, A.L., Some Studies in Machine Learning Using the Game of Checkers, *IBM J.*, Vol. 3., 1959, pp. 210-229.

[53] Samuel, A.L., Some Studies in Machine Learning Using the Game of Checkers, II, *IBM J.*, Vol. 11., 1967, pp. 601-617.

[54] Sasao, T., Terada, H., Multiple-valued logic and the design of programmable logic arrays with decoders, *Proc. of 9th International Symposium on Multiple-Valued Logic*, Bath, England, pp. 27 - 37, 1979.

[55] Sasao, T., Multiple-valued decomposition of generalized Boolean functions and the complexity of programmable logic arrays, *IEEE Trans. Comput.*, Vol. C-30, pp. 635 - 643, 1981.

[56] Sasao, T., Input variable assignment and output phase optimization of PLA's, *IEEE Trans. Comput.*, Vol. C-31, pp. 879 - 894, 1984.

[57] Sasao, T., An Algorithm to Derive the Complement of a Binary Function with Multiple-Valued Inputs, *IEEE Trans. Comput.*, Vol. C-34, pp. 131 - 140, 1985.

[58] Sasao, T., HART: A hardware for logic minimization and verification, *Proc. ICCD'85*, Oct. 7-10, 1985.

[59] Slagle, J.R., Artificial Intelligence: the Heuristic Programming Approach, *McGraw Hill*, New York 1970.

[60] Thorndike, R.L. The problem of classification of personnel, *Psymetrika*, 15, 1950, pp. 215-235.

[61] Tran, A., Graphical method for the conversion of minterms to Reed-Muller coefficients and the minimization of exclusive-OR switching functions, *Proc. IEE*, Vol. 134, Pt. E, No. 2, 1987.

[62] Tseng, Ch.J., Siewiorek, D.P., Automated synthesis of data path in digital systems, *IEEE Trans on CAD*, Vol. CAD-5, No. 3, July 1986, pp. 379-395.

[63] Vanderstraeten, G., Bergeron, M., Automatic assignment of aircraft to gates at a terminal, *Comput. Ind. Eng.*, Vol. 14, No. 1, Jan. 1988, pp. 15-25.

[64] Vasko, F., Wolf, F., Solving large set covering problems on a personal computer, *Comput. Oper. Res.*, Vol. 15., No. 2., Febr. 1988, pp. 115-121.

[65] Yu, C., Wah, B., Learning dominance relations in combinatorial search problems, *IEEE Trans. Software Engng.*, Vol. 14., No. 8., August 1988, pp. 1155-1175.

[66] Zhao, W., Two-dimensional minimization of Finite State Machines, *Master Thesis, Dept. Electr. Engn.*, PSU, 1989.