Minimal input support problem and algorithms to solve it

Research Report
ISSN 0167-9708
Coden: TEUEDE

Eindhoven
University of Technology
Netherlands

Faculty of Electrical Engineering

Minimal Input Support Problem and Algorithms to Solve It

by
Pawel A. Konieczny
Lech Jóźwiak

EUT Report 95-E-289
ISBN 90-6144-289-3
April 1995

Eindhoven University of Technology Research Reports
EINDHOVEN UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering
Eindhoven, The Netherlands

ISSN 0167-9708 Coden: TEUEDE

Minimal Input Support Problem and Algorithms to Solve It

by

Paweł A. Konieczny
Lech Jóźwiak

EUT Report 95-E-289
ISBN 90-6144-289-3

Eindhoven
April 1995

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Konieczny, Paweł A.

Minimal input support problem and algorithms to solve it /
by Paweł A. Konieczny, Lech Jóźwiak. – Eindhoven :
Eindhoven University of Technology, Faculty of Electrical
Engineering. – Fig., tab. – (EUT report, ISSN 0167-9708 ;
95-E-289)
With ref.
ISBN 90-6144-289-3
NUGI 853
Subject headings: information systems / logic design /
input support minimization.

Minimal Input Support Problem and Algorithms to Solve It

by Paweł A. Konieczny and Lech Jóźwiak

Section of Digital Information Systems, Faculty of Electrical Engineering, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

Input support reduction is one of the most important aspects of the economical representation of information in information processing systems. Solutions to this problem can be directly applied in such areas as logic design (for minimizing the number of input lines of a logic building block which implements a certain (sub-)function) or information system design (for minimizing the number of attributes in decision tables). In the case of logic synthesis, input support reduction is crucial, because the number of input lines of a logic building block influences both the active area taken by the building block and the routing area taken by interconnections.

In relation to logic synthesis, the problem received quite much attention [4, 5, 7, 8, 11, 12, 16, 26, 27, 29, 30, 32]. However, the exact algorithms for input bit minimization (all but not [11]), have too low efficiency for large instances of the problem and the approximate solution proposed by Grinshpon [11] can be quite easily improved. The main aim of the research reported here was to develop more efficient exact and approximate algorithms. Another aim was to check the usefulness of various search paradigms in solving problems of such a kind. We tried to formulate the support reduction problem as general as possible (in order to use its solution method in various areas), to analyze the problem, and to solve the problem using known local search, tabu search and branch-and-bound paradigms, and proposed by us the quick scan and beam search paradigms. In the report, the formal definition of the problem is presented, the problem is proved to be *NP*-complete and the problem structure is studied. Subsequently, the considered solution algorithms are presented and results obtained from their implementations are described.

It turned out that the *QuickScan* algorithm has the best performance for large instances of the problem, the exact algorithm is the best one for small instances (up to 20 input bits), and the local and tabu searches are not suitable. Furthermore, we decided to improve the effectiveness of the approximate *QuickScan* by proposing a beam-search algorithm, and efficiency of the exact algorithm by proposing a new branch-and-bound algorithm.

Index Terms: information system design, logic design, heuristic algorithms, combinational optimization, input support minimization

Contents

Acknowledgment: The Authors are indebted to Prof. ir. M.P.J. Stevens for making it possible to perform this work.

Introduction

Modern microelectronics technology gives opportunities to build digital circuits of huge complexity and provides a wide diversity of logic blocks. A rapidly growing interest in programmable devices has also been observed, as a result of their very attractive characteristic features, such as high speed (comparing to software solutions), low cost (comparing to hard-wired ASIC solutions), high reliability and fast customization without the need for involvement of the manufacturer.

However, programmable devices impose limitations on various circuit parameters due to input, output, functionality, memory, communication and speed constraints. On the other hand, traditional logic design methods are not suitable for very complex circuits or implementations with constrained building blocks for the following main reasons: they are only devoted to some very special cases of possible implementation structures, they often fail to find global optima for large designs, they do not consider hard constraints, they often do not consider correctness aspects in an appropriate way, and they often leave unconsidered some important factors that sufficiently influence the actual design objectives.

Logic synthesis is typically performed without any relation to the target implementation structure. Traditional logic design methods use only a limited set (minimum functionally complete set) of Boolean operators (e.g. AND-OR-NOT) and not the full set of operators implemented by a certain library of hardware building blocks. To implement the minimized expression, a transformation step called technology mapping must be performed in order to transform the expression into a network of actual hardware building blocks. If the repertoire of logic blocks offered by a certain technology library differs substantially from the set of Boolean operators used during synthesis, the work completed during synthesis is almost futile, because the real synthesis must be performed during the technology mapping.

The bad practice of target-independent logic synthesis follows from the lack of appropriate modeling tools and synthesis methods for digital circuit structures. Traditional logic modeling tools model circuits in terms of functionally complete systems composed of a minimal number of some special structural elements (e.g. AND+OR+NOT, NAND, NOR, MUX, or AND+EXOR) instead of modeling them in terms of all structural elements at the designer's disposal or, just generally, in terms of all possible sub-circuits. For example, the commonly used Boolean algebra enables us to express all possible functions but fails to model their implementation structures. Boolean algebra makes it possible to decompose functions exclusively into networks consisting of

AND, OR, and NOT sub-functions, or into the equivalent NAND or NOR networks, while in general they can be decomposed into sub-functions of any kind. Similarly, binary decision diagrams enable us to express the Boolean functions exclusively in the form of two-input multiplexer networks.

Although logic designers have been building circuits for many years, they have realized that advances in microelectronics technology are outstripping their abilities to make use of the created opportunities. It has become extremely important to develop a new generation of methods which will effectively and efficiently deal with design complexity and the characteristic features of modern building blocks, enabling modeling and synthesis of all reasonable circuit structures and providing "correctness by construction," easy correctness verification, and intelligent search algorithms for the effective and efficient exploration of the huge space of correct circuit structures.

In order to solve the problem, a structural decomposition approach may be used. It consists of transforming a system into the structure of two or more cooperating sub-systems in such a way that the original system's behavior is retained and certain constraints and objectives are satisfied.

The theoretical work in this field was started by Ashenhurst [1] and Curtis [6] for combinational circuits and by Hartmanis [13, 14, 15] for sequential circuits in the early 1960s. However, they over-simplified the actual problems. Since then, the decompositional approach has been further developed by many researchers, but only few of them [18, 19, 20, 22 28, 31] noticed the importance of the decomposition of inputs in addition to decomposition of other circuit parameters (sequential machine internal states, outputs, interconnections, memory, and/or functionality).

A recent development of a general full-decomposition theory of digital circuits by Jóźwiak [25] has rendered the task of finding the minimal number of inputs to a component block a crucial one for an effective and efficient application of the theory. Since an algorithm for input support reduction can be invoked a lot of times during one run of a decomposition algorithm, both the quality of the results and the run-time of the reduction algorithm are extremely important. Although the work reported in this report was performed in the context of research aiming at development of methods for effective and efficient application of the general full-decomposition theory [25], the importance of input support reduction is not limited to this field.

Input support reduction is one of the most important aspects of the economical representation of information in information processing systems. Solutions to this problem can be directly applied in such areas as logic design (for minimizing the number of input lines of a logic building block (e.g. PLA, look-up table, gate), which implements a certain (sub-)function or (sub-)machine) or information system design (for minimizing

the number of attributes in decision tables). In the case of logic synthesis, input support reduction is not less important than, for instance, term reduction performed by traditional two-level logic synthesis (see Example in Section *General Formulation of MISP*). The number of input lines of a logic building block influences both the active area taken by the building block and the routing area taken by interconnections between the primary inputs of a whole circuit or outputs of the other building blocks and inputs of the building block. For large circuits, especially when generated by CAD-tools of different sorts, the gain from the input minimization can be very high.

In relation to logic synthesis, the problem of input support reduction received quite much attention [4, 5, 7, 8, 11, 12, 16, 26, 27, 29, 30, 32]. However, the exact algorithms (all but not [11]) have to low efficiency for large instances of the problem and the approximate solution proposed by Grinshpon [11] can be quite easily improved. Furthermore, all the algorithms solve the problem in its specific statement related to combinational logic. In order to use its solutions in various areas, it is important to formulate and solve it as generally as possible.

The main aim of the research reported here was to formulate the support reduction problem as general as possible, to analyze the problem and to develop more efficient exact and approximate algorithms. Another aim was to check the usefulness of various search paradigms in solving problems of such a kind. We tried to attack the problem using known local search, tabu search, and branch-and-bound paradigms and proposed by us the quick scan and beam search paradigms. It turned out that the *QuickScan* algorithm has the best performance for large instances of the problem. Although it cannot guarantee a strictly optimal solution, the experimental results show that it is able to efficiently find an optimal solution or a solution which is very close to optimal. The local search and tabu search approaches are not suitable for such kind of problems. Furthermore, we decided to improve the effectiveness of the approximate *QuickScan* by proposing a beam-search algorithm, and the efficiency of the exact algorithm by proposing a new branch-and-bound algorithm.

In Chapter *Problem Formulation*, the theoretical aspects of the input support minimization problem are presented. After the problem definition, the *NP*-completeness is proved and the problem geometry is studied. Chapter *Algorithms* presents the considered solution algorithms. The proposed heuristic algorithms are based on the analysis of the problem geometry and preliminary results of a simple Tabu Search algorithm. The next chapter, *Results*, presents the results obtained form the test runs of all the algorithms introduced in the previous chapter. The results are compared and the detailed conclusions are drawn. In the last chapter, some conclusions and recommendations are presented.

Problem Formulation

The input minimization problem is the problem of finding the minimal input support, i.e. the minimal set of input bits that still preserves the functionality of a circuit. Informally it can be defined as follows: given a set of input symbols encoded in values of input variables (bits) and a partition on this set (or, in general, a set of conditions which pairs of symbols must be distinguished), find the minimal subset of coding variables (bits) that still allows to distinguish the given pairs of symbols.

Below, the formal definition of the minimal input support problem (MISP) will be given, based on different types of logical circuits and functional requirements.

Combinational Circuits

Definition 1. A fully specified combinational circuit can be defined as a set of Boolean functions y_j, $y_j: B^n \to B$, $j = 1, \ldots, m$, where n – number of input bits, m – number of output bits, $B = \{0, 1\}$ ([17]).

However, from the functional point of view values of certain functions y_j in certain points of the domain can be unimportant, that is, can be 0 or 1 without changing the useful functionality of the circuit. On the other hand, specifying such points is of great importance for circuit optimization algorithms, which can then freely choose what Boolean value will be produced by the optimized circuit. For this reason the domain of values of functions y_j is defined as a set D, $D = \{0, 1, -\}$, where "–" means "don't care". Then the design of a combinational circuit can be specified by defining functions y_j, $y_j: B^n \to D$. A tabular definition of functions y_i, called a *PLA* form or a *truth table*, defines a set of combinational circuits equivalent from the functional point of view.

Further, values of some input bits can have no influence on the value of a certain function. To reduce the size of the combinational circuit definition, don't-care values are used for inputs as well. If in PLA table a certain input pattern contains don't-cares, this means that the output value specified for this pattern is defined for all points obtained by all possible combinations of zeros and ones in the place of don't-cares. The next reduction of the size of a circuit definition comes up from omitting some points (rows), for which a *default value* is assumed (zero, one, or don't care).

Definition 2. PLA specification can be defined as m sets T_j, $T_j \subset D^{n+1}$, $j = 1, \ldots, m$ such that

$$\forall_{t \in T_j} \; \forall_{x \in B^n : x \subset (t_1,\ldots,t_n)} \quad y_j(x) = t_{n+1} \qquad \text{for } j = 1, \ldots, m$$

and

$$\forall_{x \in B^n \setminus \{x' \in B^n \mid \exists_{t \in T_j} x' \subset (t_1,\ldots,t_n)\}} \quad y_j(x) = \text{"}default\ value\text{"} \qquad \text{for } j = 1, \ldots, m$$

where $\subset$ is a relation in the set D^n such that:

$$\forall_{t', t'' \in D^n} \left[t' \subset t'' \Leftrightarrow \forall_{i=1,\ldots,n} (t'_i = t''_i) \vee (t''_i = \text{"}-\text{"}) \right]$$

Definition 3. The PLA specification is said to be *inconsistent* if there are such two points t' and t'' in T_j for a certain j, that for a certain $x \in B^n$ those points define different values of the output function $y_j(x)$. More formally, the PLA specification is inconsistent if and only if

$$\exists_{t', t'' \in T_j} \; \exists_{x \in D^n} \left[x \subset (t'_1,\ldots,t'_n) \wedge x \subset (t''_1,\ldots,t''_n) \wedge t'_{n+1} \neq t''_{n+1} \right]$$

From here, only the consistent PLAs will be considered.

For the above definition of PLA, MISP can be defined as follows:

Definition 4. MISP. Given m sets T_j, $T_j \subset D^{n+1}$, $j = 1, \ldots, m$, find a minimal input support U, $U \subset \{1,\ldots,n\}$ such that

$$\forall_{t', t'' \in T_j} \left[t'_{n+1} \neq t''_{n+1} \Rightarrow \exists_{i : i \in U} t'_i \neq t''_i \right] \qquad \text{for } j = 1, \ldots, m$$

where $\neq$ is a relation in the set D such that

$$\forall_{x', x'' \in D} (x' \neq x'') \Leftrightarrow ((x' = 0) \wedge (x'' = 1) \vee (x' = 1) \wedge (x'' = 0))$$

Sequential Machines

Definition 5. A sequential (Mealy) machine M is an algebraic system defined by: $M = (I, S, O, \delta, \lambda)$ where:

I - a finite non-empty set of inputs,

S - a finite non-empty set of internal states,

O - a finite set of outputs,

δ - the next state function, $\delta: I \times S \rightarrow S$

λ - the output function $\lambda: I \times S \rightarrow O$

A sequential machine with encoded inputs and outputs is a machine M where $I \subset B^n$ and $O \subset D^m$. Functions δ and λ form a combinational circuit which can be specified in a PLA-like format, KISS.

Definition 6. KISS: m sets T_j, $T_j \subset D^n \times S \times D$, $j = 1, \ldots, m$, and the set T_δ, $T_\delta \subset D^n \times S \times S$ such that

$$\bigvee_{t \in T_j} \bigvee_{x \in B^n : x \subset (t_1, \ldots, t_n)} \lambda_j(x, t_{n+1}) = t_{n+2} \qquad \text{for } j = 1, \ldots, m$$

and

$$\bigvee_{x \in B^n \setminus \{x' \in B^n \mid \exists_{t \in T_j} x' \subset (t_1, \ldots, t_n)\}} \bigvee_{s \in S} \lambda(x, s) = \text{"–"} \qquad \text{for } j = 1, \ldots, m$$

and

$$\bigvee_{t \in T_\delta} \bigvee_{x \in B^n : x \subset (t_1, \ldots, t_n)} \delta(x, t_{n+1}) = t_{n+2}$$

and

$$\bigvee_{x \in B^n \setminus \{x' \in B^n \mid \exists_{t \in T_\delta} x' \subset (t_1, \ldots, t_n)\}} \bigvee_{s \in S} \delta(x, s) = \text{"*"}$$

where "*" is a special symbol in S representing any state.

This specification would be exactly in PLA format if $S \subset B^k$. Usually however, the set S contains symbolic states only and the assignment of bit codes to state symbols is done in the very late stage of optimization as it reduces freedom for possible implementations of the sequential machine.

The consistency of a sequential machine specification can be defined similarly as for a combinational circuit (with taking into account the semantics of the "*" state symbol) and from this point only consistent sequential machines will be considered.

Definition 7. For the above definition of KISS, MISP can be defined as follows: given m sets T_j, $T_j \subset D^n \times S \times D$, $j = 1, \ldots, m$, and the set T_δ, $T_\delta \subset D^n \times S \times S$, find a minimal input support U, $U \subset \{1, \ldots, n+1\}$ such that

$$\bigvee_{t', t'' \in T_j} \left[t'_{n+2} \neq t''_{n+2} \Rightarrow \exists_{i : i \in U} t'_i \neq t''_i \right] \qquad \text{for } j = 1, \ldots, m, \delta$$

where the relation "≠" between states is defined as the lexical inequality of symbols.

This problem can be easily converted to PLA notation simply by choosing an arbitrary but unique coding of symbolic states. Note that when KISS is converted to PLA notation, the notion of a minimal input support is more complicated: if one primary input bit in the support has the cost of a unit, then all bits used to encode one symbolic input (state) have also (and only) the cost of a unit. It can complicate algorithms for solving the problem, so the best way to deal with it is to define (and solve) a general MISP for multi-value input variables and then to treat binary input variables (bits) as a special case (see Definition 8).

Partitions and Set Systems

The two previous definitions of MISP cover the common cases where the importance of input minimization is obvious. There are however some other situations where finding the minimal input support is of great importance. During the sequential machine synthesis (decomposition, state assignment), the relation of inequality of states ("$\neq$" $\subset$ S×S) can be weakened to an equivalence relation by introducing abstraction classes or partitions π in the set S. Then

$$\forall_{s',s''\in S} \; s' \not\equiv s'' \Leftrightarrow \pi[s'] \neq \pi[s'']$$

where $\pi[s]$ is the block of the partition π containing s.

Such a synthesis problem for sequential machines with state partitions can still be easily expressed by the KISS format and the related definition of MISP: the set of states is replaced by the set of blocks and every occurrence of a state in the specification is replaced by its block symbol.

In some stages of the synthesis, the algorithm can require even more relaxed relation. In fact it can be any reflexive and symmetric relation on the set S. Such a relation is then called an incompatibility relation ($\not\sim$) and is usually expressed in a form of a list of pairs of states that are distinguished (incompatible in terms of the "$\not\sim$" relation), or, equivalently, a set system on states (a.k.a. a rough set or an r-set). However, the compatibility relation defined as a complement of such an incompatibility relation ($S \times S \setminus$ "$\not\sim$") is not necessarily an equivalence relation because it may be not transitive.

This problem also can be translated to PLA notation but a special coding of states is necessary. Let $K = |"\not\sim"|$ (the number of state demands). Let $(s_A, s_B)_k$ be k-th state pair from the list of incompatible state pairs according to the lexical order (k = 1, ..., K). Every state $s \in S$ is coded as $\bar{s}$ on K bits ($\bar{s} = (s_1, \ldots, s_K)$) in such a way, that

$$s_k = \begin{cases} 0 & \text{if } (s, s_B)_k \quad \text{for any } s_B \\ 1 & \text{if } (s_A, s)_k \quad \text{for any } s_A \\ \text{"–"} & \text{otherwise} \end{cases} \qquad \text{for } k = 1, \ldots, K$$

Then every pair of states $(s_A, s_B) \in$ "$\not\sim$" is distinguishable by this coding, that is

$$\neg\left[\exists_{x\in B^K} \; x \subset \bar{s}_A \wedge x \subset \bar{s}_B\right]$$

and every pair of states $(s_A, s_B) \notin$ "$\not\sim$" is indistinguishable by this coding, that is

$$\exists_{x\in B^K} \; x \subset \bar{s}_A \wedge x \subset \bar{s}_B$$

The first conclusion follows immediately from the construction of coding, because $s_{Ak} \not\sim s_{Bk}$ and not "–" so x_k cannot be equal to both of them at the same time.

The second conclusion is true because if s_{Ak} is not "–" then s_{Bk} is, and inversely. Then x satisfying the required condition can be defined as follows

$$x_k = \begin{cases} s_{Ak} & \text{if } s_{Bk} \text{ is "–"} \\ s_{Bk} & \text{if } s_{Ak} \text{ is "–"} \\ \text{otherwise} & \text{is not possible} \end{cases}$$

Clearly, $x \subset \bar{s}_A$ and $x \subset \bar{s}_B$.

General Formulation of MISP

Although MISP for sequential machines with state demands can be expressed in a PLA form, the cost of it is the exponential growth of the problem size. While the problem instance with state partitions can be encoded proportionally to $\log|\pi|$ bits, this instance coded as a problem with a state set system requires the number of bits proportional to $|\pi|^2$ Further, similarly as with an artificial coding of symbolic states, the bits used for coding should not be a subject of reduction. For this reason we present the new, most general definition of MISP, appropriate for all the cases presented above.

Definition 8. General MISP. Given a set T, $T \subset Z^n$ (Z is the set of all possible values of all multi-valued input variables plus the "don't-care" value: "–"), and an incompatibility relation $\not\sim$, $\not\sim \subset T \times T$, find a minimal input support U, $U \subset \{1,\ldots,n\}$ such that

$$\forall_{t',t'' \in T} \left[t' \not\sim t'' \Rightarrow \exists_{i: i \in U} \left(t'_i \neq t''_i \wedge t'_i \neq "-" \wedge t''_i \neq "-" \right) \right]$$

(the "$\neq$" relation above denotes strict lexical inequality).

Example. Consider the following switching function presented in Figure 1. The minimal realization according to the number of terms contains 5 terms and requires all 5 inputs (see Figure 2). The PLA area calculated by the formula $(2i + o)p$ is 55.

After input support minimization, we obtain the solution $\{x_1, x_3, x_4, x_5\}$, what means that

$x_5 = 0$

x_3x_4 \ x_1x_2	00	01	11	10
00	1		0	
01	0	0		1
11	1		0	0
10	0	0		1

$x_5 = 1$

x_3x_4 \ x_1x_2	00	01	11	10
00		0		1
01				0
11	0			1
10				0

Figure 1 The example function $f(x_1,x_2,x_3,x_4,x_5)$

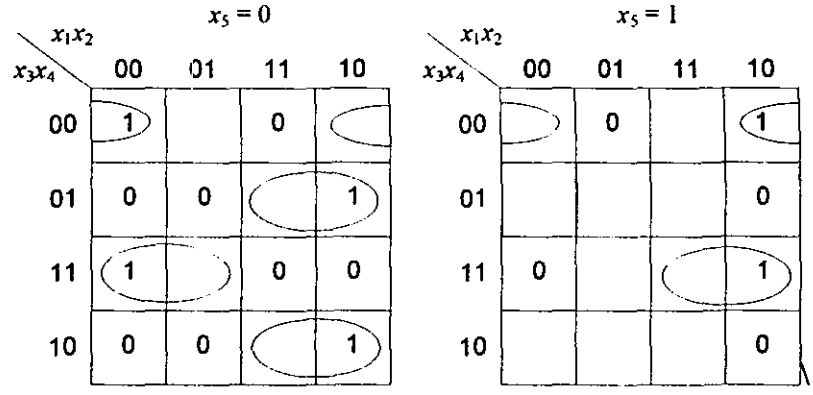

Figure 2 The minimized function $f(x_1,x_2,x_3,x_4,x_5) = \overline{x}_2\overline{x}_3\overline{x}_4 + x_1\overline{x}_3x_4\overline{x}_5 + \overline{x}_1x_3x_4\overline{x}_5 + x_1x_3\overline{x}_4\overline{x}_5 + x_1x_3x_4x_5$

the input variable x_2 can be deleted. The simplified function is presented in Figure 3 (bold values show how don't cases are assigned) and Figure 4 (as a function of four variables). The minimization of the simplified function leads to 6 terms, what makes the PLA area equal to 54.

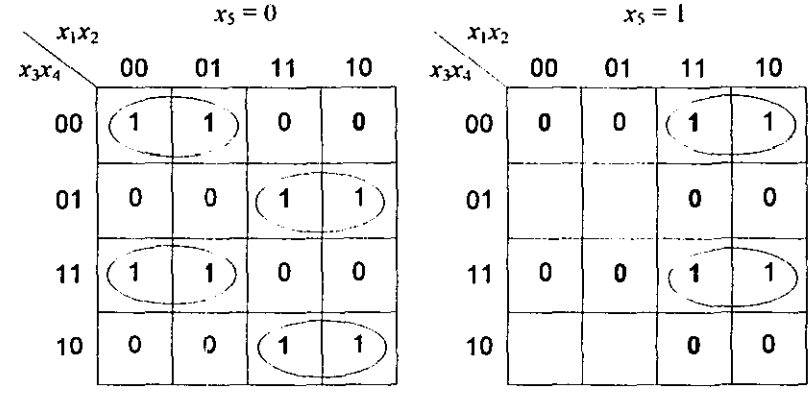

Figure 3 The minimized function with the minimal input support $f(x_1,x_2,x_3,x_4,x_5) = \overline{x}_1\overline{x}_3\overline{x}_4\overline{x}_5 + x_1\overline{x}_3\overline{x}_4x_5 + x_1\overline{x}_3x_4\overline{x}_5 + \overline{x}_1x_3x_4\overline{x}_5 + x_1x_3\overline{x}_4\overline{x}_5 + x_1x_3x_4x_5$

In this case, the input support reduction causes that the circuit is implemented by non-minimal number of product terms, but it still gives the smaller PLA active area than for the minimal term case. In addition to this, it allows for savings in routing area, as one input line is removed. We can also observe the trade-off between the number of inputs to the circuit and the minimal number of product terms (although both factors are correlated in the same direction). In general, the more complex the circuit is, the greater gain is expected by input support minimization. Of course, the gain depends heavily on the precise form of the function.

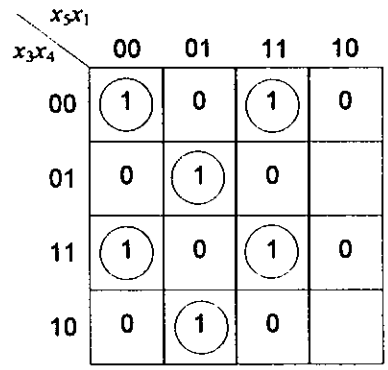

Figure 4 The minimized function with the minimal support as a function of four variables $f(x_1,x_3,x_4,x_5)$

ILP Formulation and the Complexity of the Problem

The general MISP can also be expressed in terms of Integer Linear Programming (ILP). The ILP formulation of MISP involves m binary variables.

Definition 9. The ILP formulation of MISP.
Variables:

$$x_j = \begin{cases} 1 & \text{the input variable } j \text{ is in the support} \\ 0 & \text{otherwise} \end{cases}$$

$$= \min \sum_{j=1,\ldots,m} x_j$$

subject to constraints

$$\forall_{t',t''\in T:\, t'\neq t''} \quad \sum_{i\{t'_i\neq t''_i \wedge t'_i\neq\text{"-"}\wedge t''_i\neq\text{"-"}\}} x_i \geq 1$$

$$\forall_{j=1,\ldots,m} \quad x_j \in B$$

The inequality constraints specify, for each pair of input symbols required to be distinguishable that at least one input variable that can distinguish the pair, must belong to U.

To determine the complexity of the problem, MISP should be expressed as a decision problem. Here, we restrict ourselves to the binary case; however, the proof for the multi-valued case is strictly analogous.

Definition 10. MISP as a decision problem. Given a set T, $T \subseteq D^n$, an incompatibility relation $\neq, \neq \subset D^n \times D^n$, a positive integer K, is there a support $U \subseteq \{1,\ldots,n\}$ of size K or less, that is a set such that

$$\forall_{t',t''\in T}\left[t' \neq t'' \Rightarrow \exists_{i:i\in U} t'_i \neq t''_i \right]$$

To prove that MISP is *NP*-complete, the Hitting Set Problem will be polynomially transformed to MISP.

Definition 11. Hitting Set Problem. Given a set V and a collection C of subsets of V, $C \subseteq 2^V$, a positive integer K, does V contain a hitting set for C of size K or less, that is, a subset $V' \subseteq V$ with $|V| \leq K$, and

$$\forall_{c\in C} c \cap V' \neq \varnothing$$

(V' contains at least one element for each subset in C)?

Hitting Set Problem is *NP*-complete [9] (it can be reduced to Node Cover Problem by the requirement $|c| = 2$ for every $c \in C$).

Theorem. MISP is *NP*-complete.

Proof. MISP belongs to *NP* because checking if a certain U, $|U| \leq K$ is a valid support can be done in polynomial time (exactly $O(n|T|^2)$).

Let V, $C \subseteq 2^V$ bet any instance of the Hitting Set Problem. Assume any order in V, that is, $V = \{v_1, v_2, \ldots, v_n\}$, $n = |V|$. For any $c_j \in C$ we construct the following two elements t^j and $f^j \in T$

$$\forall_{i\in\{1,\ldots,n\}} \quad t_i^j = \begin{cases} 1 & if \quad v_i \in c_j \\ - & if \quad v_i \notin c_j \end{cases}$$

$$f_i^j = \begin{cases} 0 & if \quad v_i \in c_j \\ - & if \quad v_i \notin c_j \end{cases}$$

then

$$T = \bigcup_{j=\{1,\ldots,|C|\}} \{t^j, f^j\}$$

and

$$\not\vdash = \bigcup_{j=\{1,\ldots,|C|\}} \{\{t^j, f^j\}\}$$

It is easy to see how the construction can be accomplished in polynomial time (note that $|T| = 2|C|$ and $|\not\vdash| = |C|$). All that remains to be shown is that there is a hitting set for V and C of size K or less if and only if there is a support U, $|U| \leq K$, for T and $\not\vdash$.

First, suppose that $U \subseteq \{1,\ldots,n\}$ is a support for T and $\not\vdash$ with $|U| \leq K$. Because $\not\vdash$ contains all (and only those) pairs $\{t^j, f^j\}$, U must distinguish all those pairs, that is

$$\forall_{j=\{1,\ldots,|C|\}} \exists_{i:i\in U} t_i^j \neq f_i^j$$

but since t^j and f^j contain different bits only for those i where $v_i \in c_j$ (the set c_j is hit), U must contain i s.t. $v_i \in c_j$.

Because it holds for every c_j, the set $V' = \{v_i : i \in U\}$ is a hitting set for C and $|V'| = |S| \leq K$. Conversely, if V' is a hitting set for C, $|V'| \leq K$, the set U is constructed as

$U = \{i: v_i \in V'\}$, $(|U|=|V'|)$.

Then for every pair $\{t^j, f^j\}$ in $\mathcal{A}$, there is such $i \in U$ that $t_i^j = 1$ and $f_i^j = 0$ (so $t_i^j \neq f_i^j$), simply by choosing such i that v_i ($v_i \in V'$) belongs to c_j. □

Problem Geometry

The solution space of MISP contains all subsets of the input set, that is 2^n points. To represent one input support it is convenient to use its characteristic function expressed in a form of a bit pattern. The meaning of each bit in the pattern is the same as the meaning of the variables x_j in Definition 9. Then the natural notion of a neighborhood, necessary to be defined for any heuristic search, comes up from a Hamming distance: a neighborhood $N(s)$ of a solution s is the set of all points s', for which the Hamming distance to s is 1. Further, because the quality of a solution is equal to the number of zeros in its bit pattern, all solutions can be partitioned into n levels, according to their quality. Figure 5 shows the solution space for several instances of the problem. Each

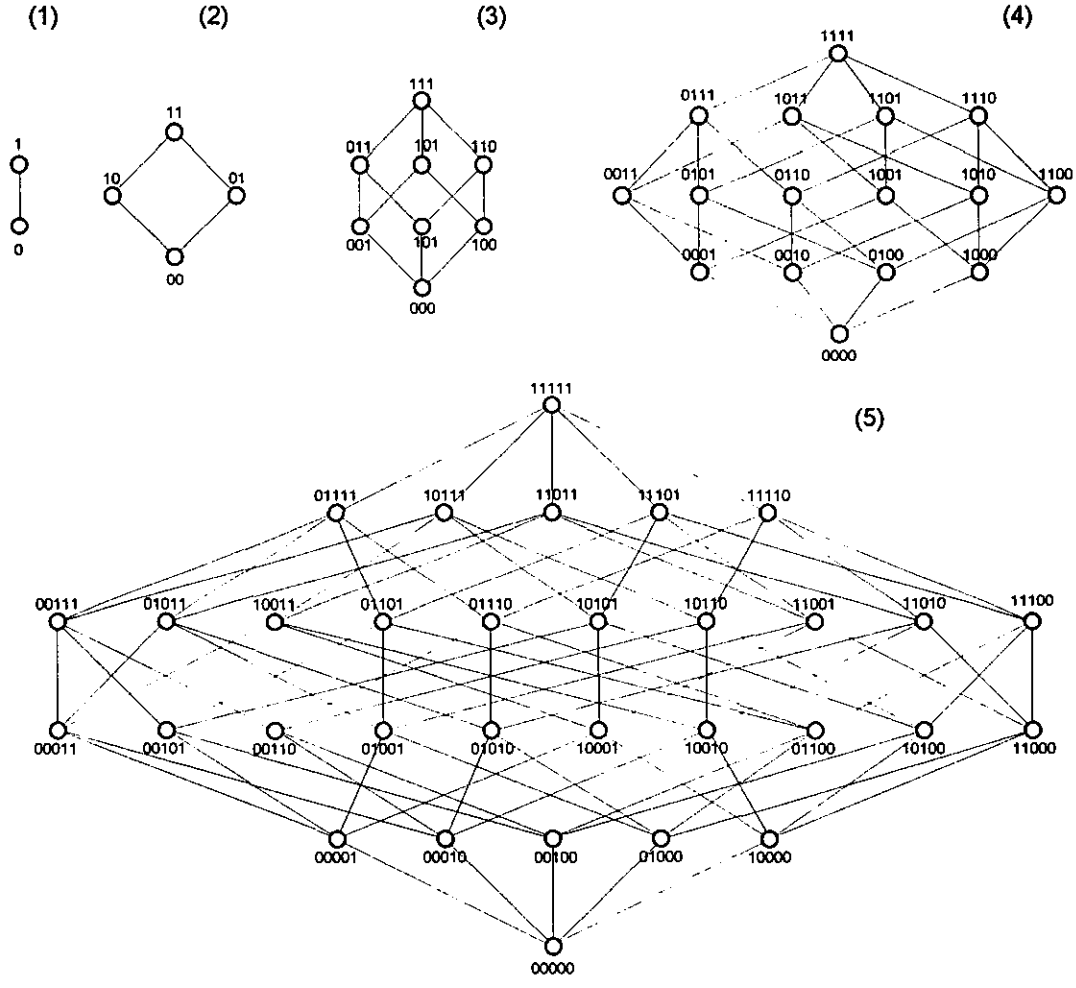

Figure 5 The leveled search space of the problem instances with n = 1, 2, 3, 4, and 5

node represents one solution and each edge joins two nodes in the Hamming distance 1, defines thus valid moves of the search.

In general the number of solutions at a quality level k is equal to $\binom{n}{k}$, so the general search space (for large n) has the form shown in Figure 6. This picture shows exactly the dimensions of the search space, but it is not accurate in expressing the search path since (as it can be observed in Figure 5) two nearly placed points need not be neighbors and the opposite.

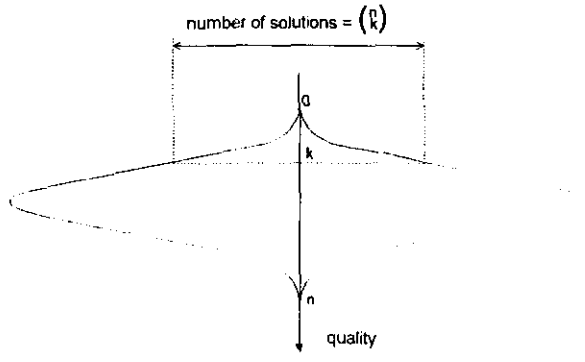

Figure 6 Search space dimensions of MISP

On the other hand it can be observed that the search space is in fact n-dimensional hypercube, and for a given vertex, all adjacent vertices form its neighborhood. Furthermore, the feasible area is easier to imagine. If s is a local minimum then all points containing at least the inputs contained in s are feasible. The set of such points forms another hypercube of the dimension $n-k$, where k is the quality of s, immersed in the solution-space hypercube. The total feasible area is the union of all sub-hypercubes induced by all local minima (Figure 7). In the sequel, this symbolic representation of the search space will be used to present various aspects of algorithms tested.

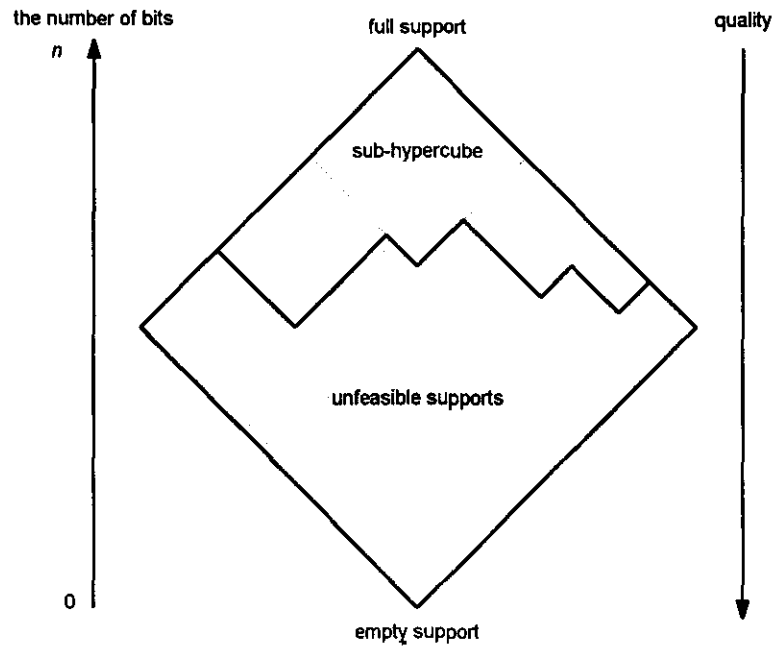

Figure 7 The solution space hypercube and a sub-hupercube of the feasible area

Algorithms

As a reference, we used the algorithm for MISP proposed and implemented by T. Łuba and J. Rybnik [29, 30]. It is an implicit exhaustive search algorithm but it has reasonably good performance for small and medium problem instances. For larger instances, however, it is not satisfactory (see Results).

To find a better quality/performance ratio and to check the usefulness of various search paradigms, a number of other techniques are developed and tested. These are: the *QuickScan* algorithm, various Tabu Search algorithms, the Local Search algorithm, and solving the problem by converting it to an equivalent two-level optimization problem and using Espresso to solve the latter.

LRed

The algorithm used in the program *LRed* is an exhaustive search trying all possible supports and selecting all minimal [29, 30]. First, it constructs a list (table) of all incompatibilities between the output symbols and for each such pair it determines what input bits can distinguish it. Then it exhaustively traverses an implicit binary decision tree (see Figure 8) cutting off the branches that lead to an unfeasible solution and stopping considering a branch when all incompatibilities are distinguished. In Figure 8 the order of input selection to be considered is simple ascending for simplicity of presentation. Actually, the algorithm is an implicit exhaustive search which uses some heuristics when choosing what input to consider next: it chooses among all bits that can distinguish the most restrictive pairs (i.e., the pairs that have the least number of distinguishing inputs), a bit that can distinguish the largest number of pairs.

In other words the algorithm starts from the empty support and adds to it all possible combinations of inputs until a support becomes feasible. In that way the program generates all possible unredundant supports (all local minima) but it collects only the minimal ones. The complexity of the algorithm is exponential.

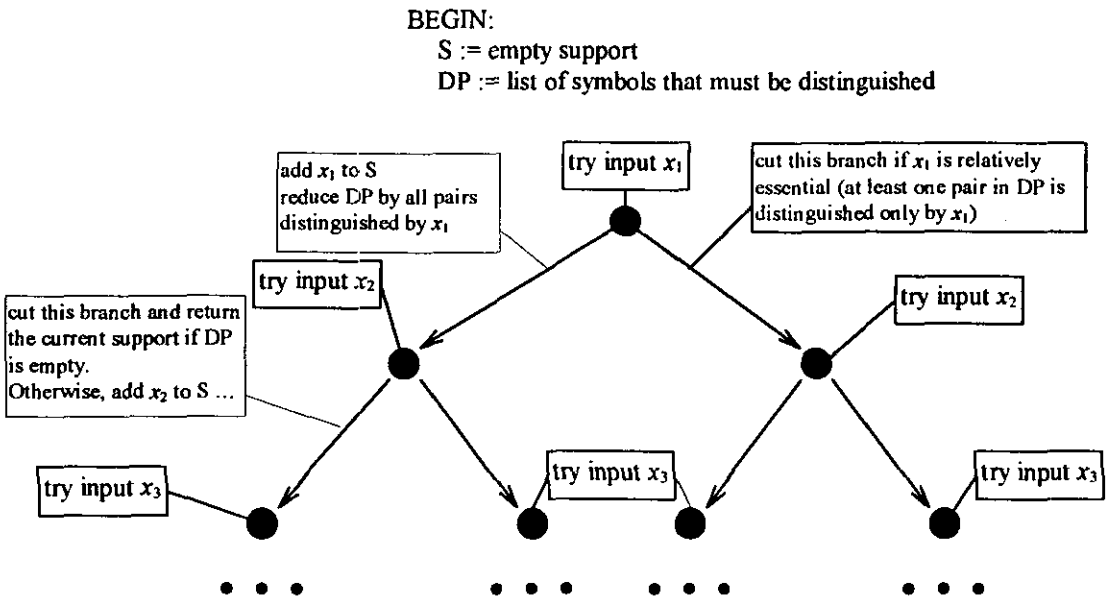

Figure 8 The algorithm of *LRed* program

Espresso

Espresso is the program for two-level Boolean function minimization [3]. If MISP is formulated as a two-level minimization problem, it can be solved directly by *Espresso*. This solution has such a potential advantage over the previous (*LRed*) that *Espresso* runs in an approximated mode using some efficient heuristics and thus allowing for obtaining (sub)optimal solutions even for large instances.

Let x_i be a Boolean variable defined as follows:

$$x_i = \begin{cases} 1 & \text{the input bit } i \text{ is in the support} \\ 0 & \text{otherwise} \end{cases}$$

Then the constraints of MISP can be defined as a Boolean function F in the form of "product of sums" (POS), in such a way that every sum corresponds to one pair of symbols that are to be distinguished, and every variable in a sum corresponds to an input that distinguishes that pair. Formally,

$$F = \bigwedge_{t',t'' \in T:\, t' \Delta t''} \left(\bigvee_{i:\, t'_i \neq t''_i} x_i \right)$$

(compare with Definition 9).

Clearly, there is equivalence between prime implicants of F and unredundant supports (local minima) of MISP (finding all such prime implicants is exactly what *LRed* does). The global minima of MISP corresponds to the largest prime implicants of F.

However, the primary goal of the *Espresso* optimization is to find an equivalent representation of F containing the minimal number of product implicants rather than the largest implicants. Therefore, the problem has to be reformulated before applicable for two-level minimization. First, notice that as F is a positive function (all variables are uncomplemented), so every prime implicant of F covers the positive vertex (corresponding to the full support). Second, as F defines the constraints or the solution space for MISP, the minterms of F represent all possible but not necessary supports. In other words they define don't-care conditions. Hence the problem of minimal support is to find a maximal product term that covers the minterm $x_1x_2...x_n$, may cover the other minterms of F and does not cover any minterms from $\overline{F}$. This can be considered as a two-level optimization problem.

Espresso requires an input written in the PLA format, that is the list of product terms for *ON*-set, *OFF*-set, and *DC*-set (two of them suffice). In our case the *ON*-set (the set of minterms for which the function has the value 1) consists of a single minterm being a positive vertex. The *OFF*-set can be obtained from the POS specifications of F: applying DeMorgan's laws, $\overline{F}$ in SOP can immediately be written by negating all variables, changing sums to products and products to sums. PLA prepared in such a way can be optimized by Espresso and the result contains one (sub-maximal) term representing a solution.

The method described above is very general and is not bound to the *Espresso* program. In fact any two-level logic optimizer could be used. We have chosen *Espresso* for its good general performance.

QuickScan

This algorithm is intended to perform a quick traverse trough several local minima and it was purposed mainly for fast recognition of the "depth" of a particular instance of the problem. Nevertheless, it seems to be a quite useful method for finding reasonably good solutions for very big cases, when other techniques cannot be applied.

Every input support is represented by a bit pattern. A bit in a position i of the pattern is a Boolean variable stating if a certain input bit x_i is or is not included in the support. The quality of the support is equal to the number of zeros in its bit pattern.

The scan is carried out in two modes: the down-mode, when the algorithm seeks a leftmost (according to the lexicographic order) local optimum (it is the initial mode when the search begins), and the up-mode, when the algorithm escapes from the local optimum in the rightmost direction (see Figure 9). During the search in the up-mode, the algorithm

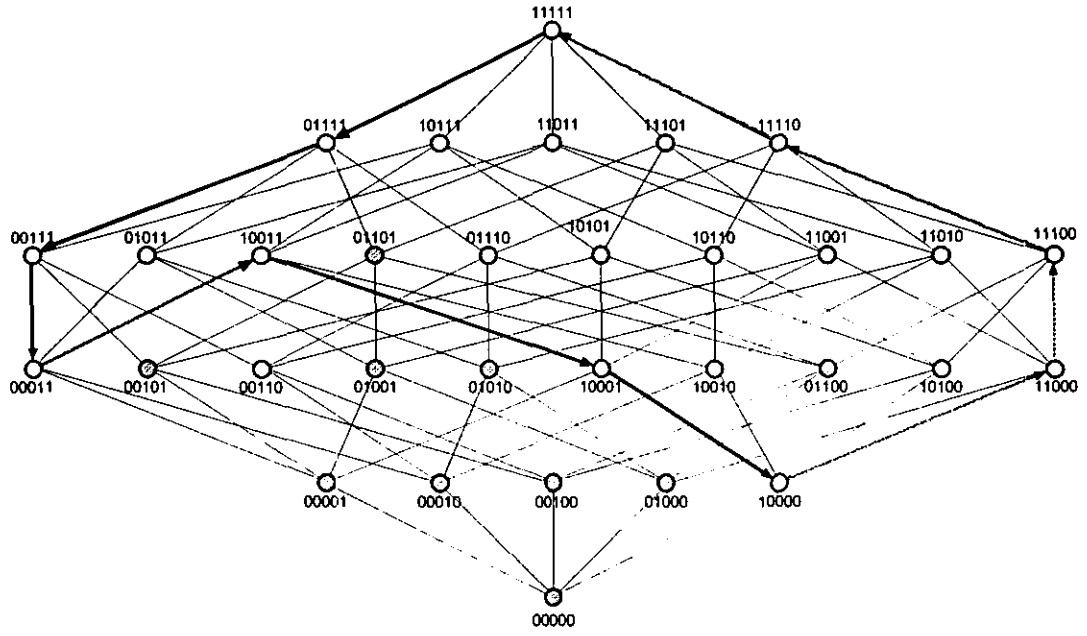

Figure 9 The scan path of the QuickScan algorithm (gray nodes represent unfeasible solutions, black arrows – the leftmost (down) scan, gray arrows – the rightmost (up) scan)

constantly tries to discover a new local minimum to enter, before it makes a next step up. In order to prevent revisiting a previous local minimum, the algorithm considers only those supports down, which are on the right from the previously visited support. The best local minimum visited is stored and makes the result of the algorithm. The complete pseudocode of the algorithm is given in Figure 10.

The search defined in this way is deterministic and cycle-free. Although the global minimum is not guaranteed, the deeper a local minimum is, the more possible ways lead to it. Therefore, the chance is little that the algorithm will skip a solution which is much better than the best visited so far.

```
procedure QuickScan
{
      current support = full support;
      best support = none;
      while (current support exists)
      {
            if (last move was down or first move)
            {
                  next support = leftmost down feasible from current support;
                  if (next support exists)
                  {
                        current support = next support;
                  }
                  else
                  {  // local minimum
                        if (current support < best support)
                              best support = current support;
                        current support = rightmost up from current support;
                  }
            }
            else  // last move was up
            {
                  next support = leftmost down feasible but to the right from previous support
                  if (next support exists)
                  {
                        current support = next support;
                  }
                  else
                  {
                        current support = rightmost up from current support;
                  }
            }
      }
      return best support;
}
```

Figure 10 Pseudocode of *QuickScan*

Tabu Search

The Tabu Search (TS) method can be viewed as an iterative technique which explores a set of problem solutions, denoted by X, by repeatedly making moves from one solution s to another solution s' located in the neighborhood $N(s)$ of s, using flexible (adaptive) memory. The moves are performed with the aim of efficiently reaching a solution that qualifies as "good" (optimal or near-optimal) by the evaluation of some objective function $f(s)$ to be minimized [10].

The notion of using memory to guide the search can be formalized by saying that the solution neighborhood depends on the time stream, hence on the iteration number k. That is, instead of $N(s)$, a neighborhood denoted $N(s, k)$ may be referred to.

Using pseudocode, the general TS algorithm can be described as below:

(a) Choose an initial solution s in X,
$s^* := s$, and $k := 1$

(b) While the stopping condition is not met do
$k := k + 1$
Generate $V^* \subseteq N(s, k)$

Choose the best $s' \in V^*$

$s' := s$

if $f(s') < f(s^*)$ then $s^* := s'$

End while

TS is a variation of a simple descent method, which always seeks the better solution in the sample of the neighborhood of the current solution. Such a descent approach will at best lead to a local minimum of f where it will be trapped. Allowing to accept moves from s to s' even if $f(s') > f(s)$, circumvents the outcome but rises the danger that cycling may occur. TS attempts to avoid the cycling by incorporating a memory structure that forbids or penalizes certain moves that would return to a recently visited solution. The simplest (and the most accurate) form of memory is embodied in a tabu list $\overline{T}$ that records the $|\overline{T}|$ solutions most recently visited, yielding $N(s,k) = N(s) - \overline{T}$. Such memory will prevent cycles of length less than or equal $|\overline{T}|$ from occurring in the trajectory. For many problems this type of memory is practically impossible to realize, due to extreme space consuming, and many other memory models are developed to make a better usage of the memory space, but they are also less precise in estimation of visited solutions. The memory containing complete solutions has such an advantage over the other models, that extending the tabu list will never deteriorate the solution quality caused by forbidding too many not yet visited solutions.

In the case of MISP a solution has a form of a set of numbers limited practically to hundreds. It can be coded effectively in tens of bytes. Hence it is possible to implement a tabu list containing the solutions visited, with the size of thousands complete solutions. The starting solution in MISP is the set containing all input bits, and the neighborhood $N(s)$ is defined as the set of all input supports containing one bit less or one bit more than s. The quality function $f(s)$ is the number of bits in the support s.

All elements except finding the neighborhood, are very easy to compute. Finding $N(s)$ is not so easy because not all from possible n neighbor solutions are feasible. For the part of them the feasibility can be easily found out: if the current solution is feasible, then all the solutions obtained by adding one bit to the current one are feasible; if the current solution is not feasible, then all solutions obtained by removing one bit from the current one are not feasible. However for the second part of solutions, the constraints of MISP have to be checked, and the test of one neighbor solution can require up to m^2 constraint evaluations (note that the upper bound for the number of constraints grows exponentially with n).

Preprocessing and Probing

Because of the exponential growth of the search space as a function of a number of inputs, the preprocessing of the initial instance of the problem is important. The preprocessing implemented and tested in this research involves finding *essential* inputs and *dominated* inputs.

Definition 12. Given T be an instance of MISP, an input i is essential if and only if

$$\exists_{t',t''\in T:\, t'\Delta t''} \; \forall_{j:\, j\in S\setminus\{i\}} \neg(t'_j \neq t''_j)$$

In other words, there is at least one pair of input symbols to be distinguished that the symbols can be distinguished by the input i only.

Definition 13. Given T be an instance of MISP, an input i is dominated if and only if

$$\exists_{j:\, j\in S\setminus\{i\}} \; \forall_{t',t''\in T:\, t'\Delta t''} \left[t'_i \neq t''_i \Rightarrow t'_j \neq t''_j \right]$$

In other words, all symbols that must be distinguished and can be distinguished by a dominated input, may be distinguished by another (single) input as well.

The preprocessing algorithm finds all essential and dominated inputs and fixes ("freezes") them in the support in such a way that all essential inputs always belong to the support (a search algorithm will not try to remove them from the support) and all dominated inputs are removed form the support (and a search algorithm will never try to add them again). Furthermore, the incompatibility relation is simplified by removing all conditions satisfied by the essential inputs.

Of course, after finding and removing some dominated inputs, some other inputs can become essential (they are called then secondary essential). The opposite is also true: after some essential inputs are found and the incompatibility relation is reduced, some other inputs may become dominated. So, the procedure of finding essential and dominated inputs is repeated until no further bits are found.

Łuba in his algorithm finds all essential inputs too, but does not try to detect dominated bits. Instead of this, he introduces a concept of semi-essential inputs [18]. A semi-essential input is an input bit that can be removed from the support, but removing it leads immediately to a local minimum. Thus all semi-essential bits must be contained in the supports having more than one bit removed.

Stated formally, the property of a semi-essential input can be expressed as follows:

$$\forall_{i:\, i \text{ is semi-essential}} \left[i \notin S \Rightarrow \forall_{j:\, j\neq i} \; j \in S \right]$$

This is a special case of a probing technique called the identification of logical implications. The general logical implications can be written as

$$\forall_{i\in\{1,\ldots,n\}}\left[i\notin S\Rightarrow\left[\forall_{j\in C_i} j\in S\wedge\forall_{k\in D_i} k\notin S\right]\right]$$

and

$$\forall_{i\in\{1,\ldots,n\}}\left[i\in S\Rightarrow\left[\forall_{j\in E_i} j\in S\wedge\forall_{k\in G_i} k\notin S\right]\right]$$

where C_i, D_i, E_i, and G_i are subsets of $\{1, \ldots, n\}$.

If the sets C_i and D_i are computed for each bit i during the probing phase as the sets of semi-essential (secondary semi-essential etc.) inputs (C_i) and "semi-dominated" (secondary "semi-dominated") inputs (D_i), the first type of the logical implication states that if the input i is not contained in a given support then all bits from C_i **have to** be included in the support and all bits from D_i **may not** be included in the support. This information may speed up the search very much since (1) the search procedure need not to consider the moves that involve deletion of bits in C_i, and (2) because it is always worthwhile to delete dominated bits, deletion of the bit i should immediately be followed by deletion of bits from D_i, making a sort of composed move. Then only such composed moves should be considered by the search procedure.

The second type of the logical implication is not very useful for MISP since after the preprocessing phase all sets E_i and G_i do not contain any non-fixed inputs.

Preliminary Results

A simple TS algorithm for MISP, as described above, was implemented in a program and tested on several benchmarks. From those runs (not presented here), several important conclusions could be drawn out and they were used to improve the first primitive algorithm.

First, if the search reached the local minimum, it would usually not be possible to escape from it by one or few moves accepting worse solutions. On the other hand, there are many ways leading to the current local optimum. Although the search space for $n = 5$ is somewhat too small, the essence of the problem can be explained using this instance (see Figure 11). Assume that the search algorithm has performed three moves: 1, 2, and 3 and found the local optimum 00101. To escape from the fathomed pit the algorithm executed the moves 4 and 5. It was still not enough and the next improving move 6 leads back in the direction of the just visited local minimum. In this example the move 7 leads out of

the local pit and allows to reach the global minimum, but in large instances it is not so easy.

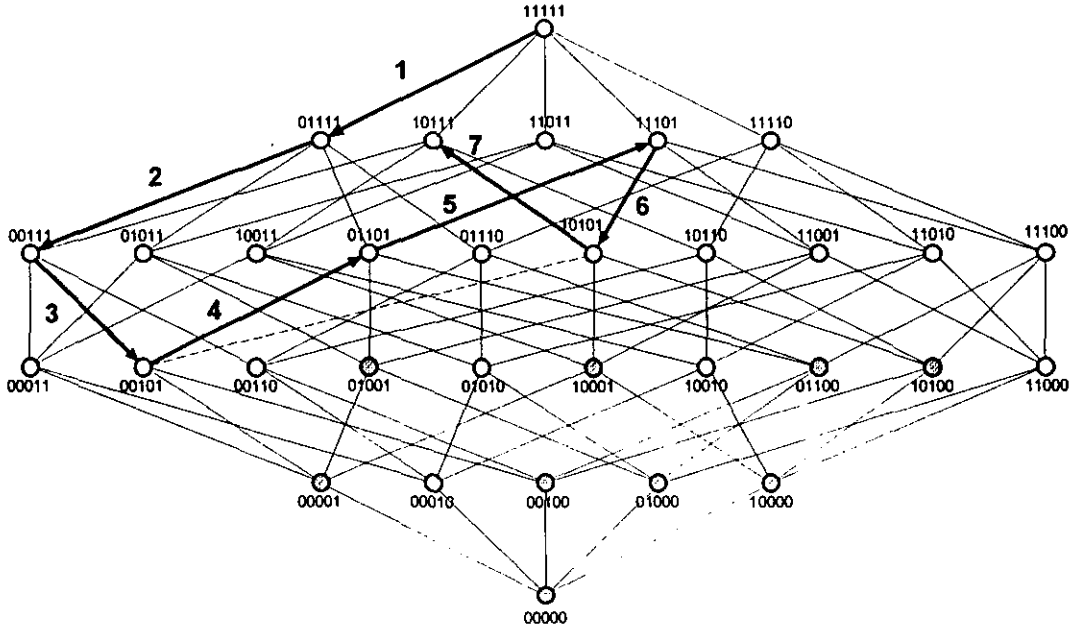

Figure 11 The difficulty of esxaping the local minimum (gray nodes represent unfeasible solutions)

Figure 12 shows the same example but using the (pseudo) cube representation. It can be seen there that the algorithm, after reaching a local optimum, tries to examine all solutions in the local hypercube (that is, this part of the sub-hypercube which is not included by any other sub-hypercube) before entering the sub-hypercube of another local

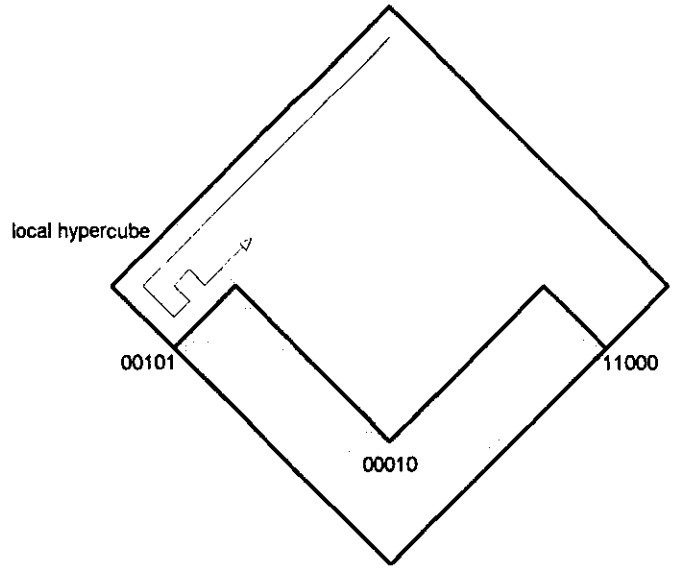

Figure 12 The same problem as in Figure 11 represented in a form of a cube diagram

optimum. This behavior makes the search very inefficient and it is very important to improve it (the improvement possibilities are discussed later).

The next problem is related to the fact that the quality function is strongly bound with the neighborhood definition. In fact all possible next solutions are always a unity better or worse than the current one. This yields many ties in choosing the next move, what was solved by fixed choosing the "left-most" point. Such an algorithm has the drawback that if a previously visited solution is revisited (because it has already fallen out from the tabu list), the whole search falls in a loop. The test runs showed that it happens very often (mostly because, as it was already mentioned, the algorithm tries to visit all the solutions in the local hypercube first so it is close to the old previously visited points) and even extending the tabu list to thousands cannot help for all but the very limited instances. The straightforward cure is to solve ties randomly and it was implemented in this way in all the improved implementations of the search.

Finally, it has happened sometimes to trap the search in a dead point. An example of such trap is depicted in Figure 13 and 14. After finding the local optimum in the third move, the algorithm executes the moves 4 and 5. The sixth move however leads to the situation when the neighborhood is empty.

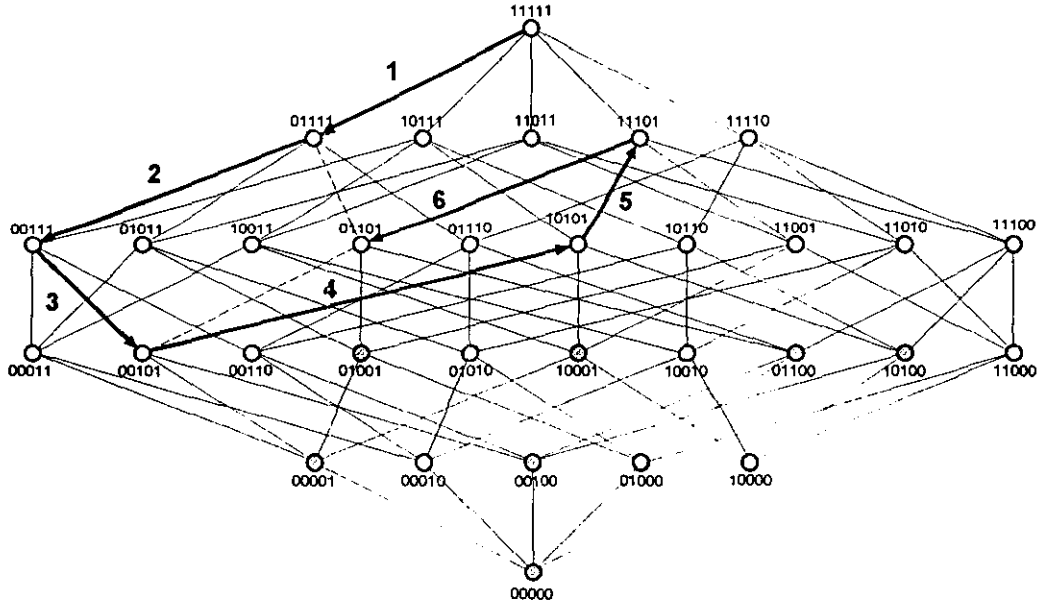

Figure 13 The danger of trapping the search

There are three possible solutions to this problem: The first is to allow the search to violate the tabu attributes of some solutions and break out of the trap. Clearly the best direction to break through is upward but the question is how many moves may violate the tabu. The simplest answer is: until the neighborhood is not empty. In such a case the search may still be involved in a trap but a larger one or it can trap again. The best

answer would be: as log as the local hypercube is not left but it recalls the first problem – how to escape from a local hypercube.

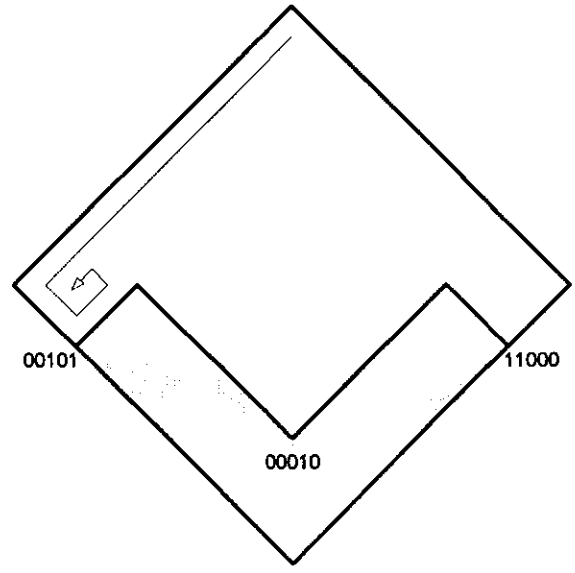

Figure 14 The trap situation in the cube diagram

The next possible solution to the trap problem is to allow the search to traverse temporally the unfeasible solutions. To prevent entering the unfeasibility region when not necessary, the unfeasible solutions must have the quality less than zero. Additionally, the quality function should drive the search fast to the nearest feasible region but since the unfeasibility border is not known, the only heuristic is to press the search up as only then we can be sure it will finally reach the feasible region. Hence the quality of the unfeasible solutions is defined as a negation of the quality they would have if they were feasible.

The third solution to this problem pursues the idea how to speed up the breaking through the tabu solutions or the unfeasible area. It may be observed that the next good point for continuing the search cannot be found in one step so why not to jump several points in order to find more free space. Only the jumps upward can be assured they will lead to a feasible solution so they still keep the search in the same sub-hypercube. However the more up the search is the greater chance that it enters the region partially overlapped by another sub-hypercube, not yet fathomed. At extreme, jumping to the top (starting) solution allows to enter any other local minimum, as the starting point is shared by all sub-hypercubes. But then we have an algorithm that restarts the search from the beginning after a local minimum is found, what is almost the simple steepest descent method and the usefulness of the tabu list is questionable.

Escaping from a Local Hypercube

All the considerations above show, that the escaping from the local hypercube after reaching a local minimum is at an absolute premium. It cannot be done in a deterministic way since the size and the shape of the local cube is not known – choosing one direction of escaping may require more moves up than another direction (see Figure 15).

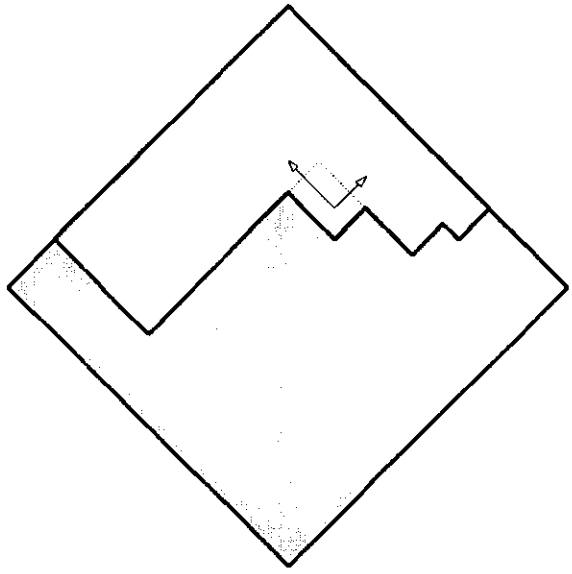

Figure 15 Depending on the direction of the search, escaping from the local hypercube can differ in the number of steps

We propose the following two methods for leaving the local hypercube. The first follows the situation showed in Figure 15: after reaching the local minimum, an arbitrary (random) direction of escape is chosen and no improvement move will be accepted if it does not lead out of the current sub-hypercube. Expressing it in the tabu terms, all operators of removing bits not included in the last visited local minimum are forbidden until another remove takes place. This method has such a drawback that the randomly chosen direction of the escape may cause that no neighbor sup-hypercube is found. This situation is presented in Figure 16.

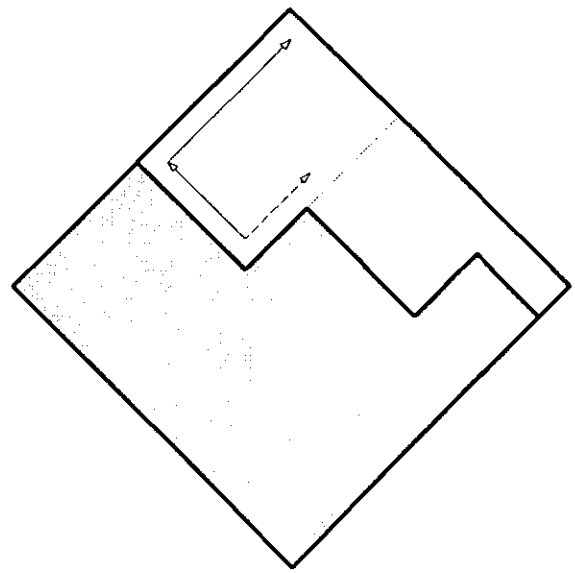

Figure 16 Choosing the wrong direction of escape can lead to the initial solution instead to the next local minimum

Instead of rendering certain moves a temporary tabu, this method can be viewed as adding (during the escape) all solutions at the same quality level from the sub-hypercube to the tabu list. This "extended tabu rendering" is stopped when another local hypercube is entered (Figure 17).

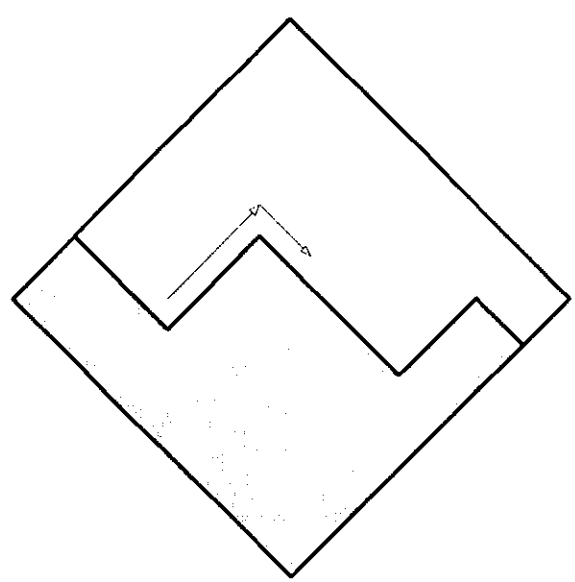

Figure 17 When leaving a local hypercube, all solutions in it at the same level of quality are rendered tabu

This observation suggests the idea that instead keeping the tabu list in a form of all the (last) previously visited solutions, the whole sub-hypercube should be rendered tabu, when the local minimum is found. Of course, in such a case a tabu status of a solution cannot be prohibition of visiting it but the entering should be penalized only.

Furthermore, to drive the search out of the tabu cube, such penalty should be diversified to prevent revisiting the last local minimum before the sub-hypercube is left. Because the direction to the next local optimum is not known, such penalty pressure can only drive the search upward – the same situation as it was with the quality for unfeasible solutions. It will overestimate the upper parts of the solution space but it should be compensated by the fact that the upper space belongs to several sub-hypercubes and would be penalized several times when some more local minima were discovered.

Improved Algorithms

It is difficult to predict which one from the improvements proposed will be the best. The intuitive order of accuracy is as used in the introducing the improvements in the previous section since it reflects the considerations going deeper and deeper in the problem nature. On the other hand all the methods suffer from some drawbacks and therefore all of them are implemented in the program (*Misp*) and compared with several non-TS techniques. Below, a concise review of the algorithms implemented is presented.

1. **Violate the tabu area.** The solution to the trap problem. The search can enter the tabu area, which quality function is the negation of the ordinary quality function. Parameters: the stop criterion, the tabu list length.
2. **Violate the unfeasibility area.** The solution to the trap problem. The search can enter the unfeasibility area, which quality function is the negation of the ordinary quality function. Parameters: the stop criterion, the tabu list length.
3. **Restart from the beginning.** After a local minimum is found, the next step jumps back to the complete support. The actual tabu list is retained. Parameters: the stop criterion, the tabu list length.
4. **Tabu for all remove-bit moves** involving one of the bits not included in the last local minimum found. This tabu is imposed when a minimum is found, and retracted when any non-tabu remove-bit move takes place (the search leaves the sub-hyperspace) or the search returned to the starting point. Parameters: the stop criterion.
5. **Tabu for sub-hypercubes.** This tabu is imposed on all solutions contained in the sub-hypercube. The tabu solutions may be traversed but their quality is lowered by the number of tabu sub-hypercubes they belong to. The tabu status is not recalled (unless a sub-hypercube falls out of the tabu list as the length of the list exceeds a given value). Parameters: the stop criterion, the tabu list length.

The stop criterion can be a maximal number of steps without improvement, a maximal number of steps executed, a solution of a given quality found, or given time elapsed. For comparing the results of TS algorithms with others, the last two conditions were used for testing (although all of them are implemented).

The algorithms used to compare the efficiency of TS algorithms are:

1. **The program *LRed*** thanks to the courtesy of Prof. T. Łuba from Warsaw University of Technology. This program was specially developed for the input minimization of binary and multi-value combinational circuits expressed in the PLA form. It implements several formulas from logic synthesis theory and, based on this, it performs an implicit exhaustive tree-search controlled by some heuristics for choosing the branch to process. Hence, it finds always the set of all best solutions and in the worst case requires the exponential time of execution. Parameters: none.
2. **The exhaustive search.** To perform an exhaustive search, it is not necessary to check all possible solutions in the solution space, even not the feasible ones. Since the local minima are always placed at the border between the feasible and unfeasible area, it is enough to perform the search along that border and this is implemented in the exhaustive algorithm. The only essential difference between this algorithm and *LRed* is that this algorithm starts with the full support and tries to remove as many bits as possible maintaining the feasibility of the support, and *LRed* begins with an empty solution and tries to add as few bits as possible in order to achieve feasibility. Parameters: none.
3. **The simple local search.** This algorithm is implemented mainly to compare with the TS Algorithm 3. The only difference between them is that the local search does not use the tabu list. Parameters: the stop criterion.
4. **The QuickScan algorithm.** This algorithm deterministically visits a number of local optima and returns the best ones. It was implemented mainly to perform a quick estimation of possible results for a given instance of the problem, but it may be a reasonable alternative for instances that are too difficult for more sophisticated algorithms. Parameters: none.

The last three algorithms are embedded in the program *Misp*.

Results

This section presents the test results of the algorithms described in the previous section on page 30 and next. Almost all test files were specially generated for these tests because the standard benchmarks are already multiple-output minimized so the only valid input support is that containing all inputs. Single-output support minimization is also quite easy for these examples so that any reasonable algorithm performs well. It is thus almost impossible to compare various algorithms when using examples from the standard benchmark set. Our main aim is to apply the input minimization algorithms in the context of circuit decomposition where they have to find a minimal support for usually small subfunctions of a given large function occurring during automatic logic synthesis. So, they are expected to work reasonably for difficult instances of the problem where many input variables can be reduced. The generated instances of the problem give a good test base for the time being. A more reliable set of benchmarks can be gathered during operation of a CALD tool which uses the input minimization procedure. Now the benchmarks artificially generated should suffice, although it is possible that they are too uniform to discover all properties of the algorithms.

Two programs were written to generate the benchmarks. The first one (*misp_gen*) creates a list of minterms for *ON*-set and *OFF*-set. The usual size of the minimal support is 10 and the instances do not contain either essential or dominated inputs. It all makes those benchmarks "difficult" (as the preprocessing does not reduce the problem size) especially for those algorithms which start with the full support and try to delete as much inputs as possible (actually, only *LRed* begins with the empty support and adds inputs to it to find a feasible support). The second program (*gen_pla*) generates randomly a set of terms of different size (not necessarily minterms). However, it defines PLAs specifying only ON-sets and DC-sets (PLA type *fd*), and all algorithms for MISP require ON-set and OFF-set to be given (PLA type *fr*). The conversion between both formats of PLA can be performed using *Espresso*, and the programs tested are able to invoke *Espresso* by themselves. But to avoid influence of this process that might have on execution time of the programs tested, the test files were converted to *fr* format manually. Those benchmarks are characterized by a large number of product terms and the fact that all input bits are either essential or dominated (they have only one local minimum). In other words they constitute "easy" examples, but imposing high requirements on memory resources.

The algorithms were tested on three groups of benchmarks. The first group of benchmarks (Table 1) was used to compare an overall performance of all algorithms. It is

composed of two sets of files: "difficult" (generated but *misp_gen*) and "easy" ones (generated by *gen_pla*). Table 1 describes the benchmarks specifying the file name, the number of inputs, the number of product terms, the number of inputs in the minimal support, the number of minimal supports, and the number of essential and dominated inputs. All PLAs in this group have only one output bit.

Table 1 Benchmarks I

File	*#inputs*	*#terms*	*Q(min)*	*#min*	*#essential*	*#dominated*
i20s04a.pla	20	20	4	31	0	0
i20s05a.pla	20	30	5	66	0	0
i20s06a.pla	20	40	6	72	0	0
i30s08a.pla	30	120	8	7	0	0
i30s08b.pla	30	120	8	17	0	0
i30s08c.pla	30	120	8	13	0	0
i30s10a.pla	30	240	10	3	0	0
i40s09a.pla	40	160	9	1632	0	0
i40s09b.pla	40	160	9	1632	0	0
i40s10a.pla	40	240	10	716	0	0
i20s10a.pla	20	40	10	1	10	10
i20s13a.pla	20	40	13	1	15	5
i20s15a.pla	20	217	15	1	15	5
i20s16a.pla	20	450	16	2	16[1)]	4[1)]
i20x20a.pla	20	181	12	1	12	8
i20x20b.pla	20	150	19	1	19	1
i20x20c.pla	20	37	12	1	12	8
i30s25a.pla	30	1573	25	1	25	5
i30x30a.pla	30	9785	26	1	26	4
i30x30b.pla	30	17747	23	1	23	7
i30x30c.pla	30	9850	13	1	13	17
i40x40a.pla	40	90	10	1	10	30
i40x40b.pla	40	1402	23	1	23	17
i40x40c.pla	40	2110	29	1	29	11

[1)] two inputs are interchangeable so one is rendered dominated and second becomes essential

QuickScan and Preprocessing

Table 2 presents the results of testing the preprocessing algorithm and the *QuickScan* algorithm. The preprocessing can only reduce the instance size, so in order to notice the influence, a regular search has to be performed after that. For this purpose, the *QuickScan* algorithm was selected as it is deterministic and consumes little computer time. Table 2 gives for each test file from the Benchmark Set I the time used by pure *QuickScan*, time consumed by *QuickScan* preceded by the preprocessing, the difference (gain) between them, and additionally the quality of the best solution found.

Table 2 Results of *QuickScan* runs on PC 386/20 MHz

File	*T(qs)*	*T(qs+prep)*	*Δ*	*Q(qs), Q(qs+prep)*
i20s04a.pla	0.4	0.7	-0.3	4
i20s05a.pla	1.5	2	-0.5	5
i20s06a.pla	3	4	-1	6
i30s08a.pla	28	43	-15	9
i30s08b.pla	43	58	-15	9
i30s08c.pla	46	1:01	-15	9
i30s10a.pla	2:43	3:45	-62	11
i40s09a.pla	2:07	2:52	-45	10
i40s09b.pla	2:07	2:52	-45	10
i40s10a.pla	2:51	4:32	-101	11
i20s10a.pla	1	0.5	0.5	10
i20s13a.pla	1.5	0.4	1.1	13
i20s15a.pla	0.5	0.2	0.3	15
i20s16a.pla	1	0.3	0.7	16
i20x20a.pla	7	2	5	12
i20x20b.pla	4	6	-2	19
i20x20c.pla	1	0.4	0.6	12
i30s25a.pla	9:06	m)		25
i30x30a.pla	m)	m)		
i30x30b.pla	m)	m)		
i30x30c.pla	m)	m)		
i40x40a.pla	2	1	1	10
i40x40b.pla	2:37	1:26	71	23
i40x40c.pla	4:25	3:12	73	29

[m] out of memory

For the first group of benchmarks, the value of Δ represents the pure cost of preprocessing (worst case examples), as the preprocessing is not able to reduce the instance sizes. It can be noticed that this cost is not very high and it is worthwhile to spend this additional time on preprocessing as it may speed the search for some examples.

The second group shows that the time used by preprocessing is essentially shorter than for the search algorithm (notice that the search has to discover only one local minimum).

The last column in Table 2 reveals the accuracy of the *QuickScan* algorithm itself. Comparing it with the data in Table 1, we can notice that *QuickScan* reached in all cases the global optimum (in reasonable time).

LRed

The next table (Table 3) presents the results of *LRed* tests. Because this program consumes enormous time for large instances from the "difficult" group, it was tested additionally on a Silicon Graphics' supercomputer. The times measured in this way were used to calculate the average performance ratio, which in turn was used to estimate the computation time for other benchmarks, if the time consumed by *LRed* made it impractical to perform tests on PC.

Table 3 Times of *LRed* runs

File	*PC 386/20 MHz*	*SGi PowerChallenge*	*performance ratio*[a]
i20s04a.pla	1	<1	
i20s05a.pla	7	<1	
i20s06a.pla	15	<1	
i30s08a.pla	2:47:07	2:33	65.536
i30s08b.pla	2:02:35	1:39[0]	74.293
i30s08c.pla	1:54:06	1:39[0]	69.152
i30s10a.pla	22:02:10	20:32	64.391
i40s09a.pla	> 3.5 days	2:50:49	
i40s09b.pla	> 3.5 days	3:05:55	
i40s10a.pla		15:20:32	
i20s10a.pla	2	<1	
i20s13a.pla	2	<1	
i20s15a.pla	1	<1	

File	PC 386/20 MHz	SGi PowerChallenge	performance ratio[a)]
i20s16a.pla	1	<1	
i20x20a.pla	2	<1	
i20x20b.pla	4	<1	
i20x20c.pla	1	<1	
i30s25a.pla	31	<1	
i30x30a.pla	[w)]	1	
i30x30b.pla	[m)]	2	
i30x30c.pla	[w)]	<1	
i40x40a.pla	2	<1	
i40x40b.pla	14	<1	
i40x40c.pla	24	<1	

[f)] memory fault but results still correct

[w)] wrong results

[m)] out of memory

[a)] average performance ratio is 68.3

From this table, the exponential complexity of *LRed* can be noticed for benchmarks from the first set. The results on the second set are not a surprise since *LRed* has a build-in preprocessing. What can be noticed is that *LRed* is apparently more efficiently coded than *Misp*. However, some run-time errors appeared during execution of *LRed*.

Another remark emerges from comparing Table 2 and Table 3: *QuickScan* is able to perform fast search even for very difficult examples. This is investigated further in Table 9.

Espresso

Table 4 presents the results of the minimization algorithm that uses *Espresso* (version 2.3) internally for finding the input support. It can immediately be noticed, that this option is totally outperformed by the algorithms discussed so far, since it was not able to compute solutions for all but very small cases. It often miserably fails because of an internal error in *Espresso* program. Other tests (not presented here), performed on the Benchmark Set III (Table 8), showed that even if the algorithm was able to finish successfully, the solutions found were often not optimal. It is still possible, however, that another two-level logic minimization program can give better results.

Table 4 Results of Espresso runs (misp -logmin) on PC 386/20 MHz

File	T(logmin)	Q(logmin)
i20s04a.pla	1:26	4
i20s05a.pla	m)	
i20s06a.pla	m)	
i30s08a.pla	m)	
i30s08b.pla	m)	
i30s08c.pla	r)	
i30s10a.pla	m)	
i40s09a.pla	m)	
i40s09b.pla	m)	
i40s10a.pla	m)	
i20s10a.pla	5	10
i20s13a.pla	5	13
i20s15a.pla	3	15
i20s16a.pla	4	16
i20x20a.pla	10	12
i20x20b.pla	22	19
i20x20c.pla	4	12
i30s25a.pla	m)	
i30x30a.pla	m)	
i30x30b.pla	m)	
i30x30c.pla	m)	
i40x40a.pla	5	10
i40x40b.pla	m)	
i40x40c.pla	m)	

m) out of memory

r) espresso runtime error: abs_select_restricted: should not have best_var == -1

Tabu Searches and Local Search

The tests of the Tabu Searches and the Local Search on the Benchmark Set I in comparison to *QuickScan* are presented in Table 5. The same time used by *QuickScan* was given to each of the searches and the best solution found in this time makes the

quality measure. Only those files were used for which the execution time of *QuickScan* was more than 10 seconds.

Table 5 Comparing QuickScan, Tabu Searches and Local Search (PC 386/20 MHz)

File	*time*	*Q(qs)*	*Q(t1)*	*Q(t2)*	*Q(t3)*	*Q(t4)*	*Q(t5)*	*Q(ls)*
i30s08a.pla	28	9	27	27	27	27	27	27
i30s08b.pla	43	9	26	26	26	26	26	26
i30s08c.pla	46	9	26	26	26	26	26	26
i30s10a.pla	2:43	11	26	26	26	26	26	26
i40s09a.pla	2:07	10	35	35	35	35	35	35
i40s09b.pla	2:07	10	35	35	35	35	35	35
i40s10a.pla	2:51	11	37	37	37	37	37	37
i30s25a.pla	9:06	25	25	25	25	25	25	25
i40x40b.pla	2:37	23	35	35	35	35	35	35
i40x40c.pla	4:25	29	35	35	35	35	35	35

As it can be seen, the Tabu Searches were not able even to reach a local minimum in the time they were given, what implies the corollary that this type of heuristic search is not very suitable for MISP. This corollary is also supported by other results too (Table 7).

Table 6 compares the performance of *LRed*, the Tabu Searches and the Local Search. The test files are the same as in Table 5, but the "easy" examples are removed. The columns report the time necessary for the program to finish (*T(LRed)*) or time necessary for a search to find the global optimum (Tabu Searches and Local Search). When a search was not able to find the optimum in three hours, the quality of the best solution and the number of the best solutions found is reported in parentheses.

Table 6 Comparing LRed, Tabu Searches and Local Search (PC 386/20 MHz)

File	*T(LRed)*	*T(t1)*	*T(t2)*	*T(t3)*	*T(t4)*	*T(t5)*	*T(ls)*
i30s08a.pla	2:47:07	1:27:40	(9/219)	(9/9)	(9/296)	2:00:00	(9/6)
i30s08b.pla	2:02:35	1:17:50	20:50	(9/11)	39:20	13:18	(9/5)
i30s08c.pla	1:54:06	1:09:40	1:18:50	(9/6)	2:04:00	(9/196)	(9/5)
i30s10a.pla	22:02:10	(11/43)	(11/63)	(11/1)	(11/51)	42:50	(11/2)
i40s09a.pla	8:02:34:00[e)]	23:10	39:10	(10/2)	22:10	(10/176)	(10/3)
i40s09b.pla	8:19:46:00[e)]	23:20	47:10	(10/7)	1:22:10	1:35:40	(10/4)
i40s10a.pla	43:16:32:00[e)]	(11/77)	(11/63)	(11/1)	32:10	(11/51)	(11/1)

[e)] time estimated by multiplying the execution time on SGi Challenge by performance ratio 68.3

Those results show that Tabu Searches can find the optimum much faster than *LRed* in the cases where the number of global optima is large. For example for *i40s09a.pla* the global optimum can be found in half an hour by a Tabu Search, while *LRed* will take more than eight days to finish. It is still possible, however, that *LRed* would find the optimum in comparable time if it incorporated a stop condition (maximal time of execution or the best quality to find) instead of performing the exhaustive search to the very end.

Because the Tabu Searches are nondeterministic and only one run per example was performed, the results in Table 6 do not say much about relative performance of the Tabu Searches. We can conclude only that the Tabu Search 3 and the Local Search are much worse than other searches (they were not able to find a global optimum in any case).

Table 7 presents the results of investigations over the relative performance of Tabu Searches. The test runs were performed on a separate set of benchmarks; the parameters of the files are provided directly in the table. For comparison, the table contains the execution time of LRed for those examples. Tabu Searches and Local Search were run three times on the same benchmark (column *run*). In the column labeled *min*, the first number is the minimal support length, the second – number of minimal supports. For columns labeled *t1-t5*, *ls*, *exh*, the first number is the time necessary for finding one minimal support (">" means that no minimal support was found in 120 min; in that case the number in parentheses gives the length of the best support found), the number after the slash represents the number of iterations executed.

Table 7. Time necessary for finding the minimal support. PLAs: output bits = 1, 1-minterms = 0-minterms = inputs.

inputs	*run*	*min*	*t1*	*t2*	*t3*	*t4*	*t5*	*ls*	*exh*	*LRed*
10	1	4/1	1/12	1/8	1/11	1/10	4/40	1/16	1/41	1
	2		1/10	1/14	1/11	2/22	1/10	3/33		
	3		2/22	2/24	4/40	1/16	2/24	1/12		
20	1	6/57	10/28	17/46	61/121	11/30	11/28	54/122	2/37	7
	2		12/32	16/44	69/138	8/20	21/60	6/14		
	3		11/30	11/30	168 /297	14/40	10/28	37/84		
30	1	6/1	>(8) /3532	>(7) /2958	>(8) /2146	>(8) /11757	>(7) /3541	>(7) /5624	>(8) /276391	16:52
	2		>(7) /3522	78:00 /2066	>(7) /3019	105:0 /4456	87:20 /2662	>(7) /5619		
	3		>(7) /3525	>(7) /2962	>(8) /3014	113:5 /5352	>(7) /3548	>(8) /5607		

inputs	*run*	*min*	*t1*	*t2*	*t3*	*t4*	*t5*	*ls*	*exh*	*LRed*
40	1	$7_{/126}$	31:00 /827	6:33 /219	>(9) /1742	12:05 /569	13:17 /429	>(9) /2515	>(9) /48925	3:48:31
	2		1:32 /33	>(8) /2084	>(9) /1736	47:10 /2317	11:40 /395	>(8) /2516		
	3		7:53 /273	8:16 /269	>(8) /1733	27:20 /1343	15:27 /487	>(9) /2501		
50	1	$7_{/1}$	>(9) /1714	>(9) /1569	>(9) /1100	>(9) /3588	>(8) /1688	>(9) /1351	>(10) /32668	82:03:33
	2		>(9) /1699	>(9) /1572	>(9) /1099	>(9) /3626	>(9) /1707	>(9) /967		
	3		>(9) /1718	>(9) /1570	>(9) /1104	>(9) /3587	>(9) /1707	>(9) /1356		

From these results, several conclusions can be drawn out:

- PLAs with 10 and 20 inputs are too small to be representative. For those instances, the simplest algorithm (exhaustive search) is the best.
- The number of global minima has a great influence on the effectiveness of the Tabu Searches. The more global minima, the greater chance that the searches find one of them what is in accordance with intuition (PLA with 40 inputs and 126 global minima is easier than PLA with 30 inputs and only one global minimum).
- *LRed* is often faster than the Tabu Searches in finding the global minimum.
- The exponential run time of *LRed* can be observed (however those were very difficult PLAs for *LRed*; it works much faster for the real PLAs where only few bits have to be removed to reach the global minimum).
- *LRed* is worse for PLA with 40 inputs, but it finds all global minima. The version of *LRed* finding only one solution is supposed to be much faster.
- t3 is much worse than the other Tabu Searches. It is comparable with the local search. No influence of the tabu list can be observed, maybe it improves a little, but the algorithm executes then fewer steps.
- Apart from this, all Tabu Searches are better than the local search.
- The measurements presented are not precise enough, to determine subtle differences in efficiency among t1, t2, t4, and t5; it seems however, that these algorithms work with comparable speed (the less precise algorithms make up for efficiency with the greater number of steps executed in the same time). To estimate it, more runs for one PLA are necessary (to reduce the influence of random) and the execution time should not be so strictly limited. Of course it costs much time and the general efficiency level is already noticeable.

- The apparent superiority of *LRed* over the exhaustive search incorporated in *Misp* can be explained by the fact that in virtually all cases the minimal support was closer to the empty support than to the full one, so *LRed* has substantially smaller search space to traverse.

Additional Tests for *QuickScan* and *LRed*

The results presented above reveal good performance/quality ratio for *QuickScan*. In other words, this algorithm produces quite good solutions in very short time. To pursue this property, several extra tests were performed. Table 8 describes the third set of benchmarks used in this research.

Table 8 Benchmarks III

File	*#inputs*	*outputs*	*#terms*	*Q(min)*	*#min*	*#essential*	*#dominated*
i5.pla	5	5	40	3	1	3	2
i10.pla	10	10	180	7	1	4	0
i11.pla	11	11	220	7	1	2	0
i12.pla	12	12	264	9	3	7	1
i13.pla	13	13	312	9	27	2	0
i14.pla	14	14	364	9	1	4	0
i15.pla	15	15	450	9	5	1	0
kaz.pla	21	1	31	5	35	0	0
i40.pla	40	1	240	10	716	0	0
i50.pla	50	1	100	7	1	0	0
i100.pla	100	1	200	?	?	0	0

The results of those test runs are presented in Table 9. It can be noticed again that *LRed* is the best algorithm for examples with less than 20 inputs. For larger examples however, its use becomes impractical. *QuickScan*, on the other hand, is able to process even extremely large examples in reasonable time. It does not find the best solution in all cases, but the solution found is one or two bits worse than the optimal (values bolded in Table 9). It can also be observed that *QuickScan* still finds the optimum for small instances and the most difficult case for it was *i50.pla*, where is only one global optimum.

Table 9 Comparing *LRed* and *QuickScan*

File	T(LRed)	Q(min)	T(qs)	Q(qs)
i5.pla	1	3	1	3
i10.pla	1	7	2	7
i11.pla	2	7	6	7
i12.pla	2	9	7	9
i13.pla	5	9	16	9
i14.pla	5[s]	9	13	9
i15.pla	9	9	19	**10**
kaz.pla	6	5	2	6
i40.pla	43:07:50:00[e]	10	2:52	**11**
i50.pla	4:05:40:00[e]	7	25	9
i100.pla	> 7 months[e]	?	3:38	11

[s] program finished successfully but the computer got stuck after that

[e] time estimated by multiplying the execution time on SGi Challenge by performance ratio 68.3

Conclusions and Recommendations

The input support minimization is one of the most important aspects of the economical representation of information in information processing systems. In particular, it is of crucial importance for logic synthesis. However, the exact algorithms for solving the problem [4, 5, 7, 8, 12, 16, 26, 27, 29, 30, 32] have too low efficiency for large instances. Therefore, developing a more efficient exact algorithm as well as an effective and efficient approximate algorithm was really important.

In order to use the algorithms in various areas, it was important to formulate and solve the problem as generally as possible. It was also important to check the usefulness of various search paradigms in solving problems of such a kind.

In this report, the formal definition of the problem is presented, the problem is proved to be *NP*-complete, and the structure of the solution space is analyzed. Based on this analysis, several heuristic algorithms (tabu searches, local search, and *QuickScan* search) were developed, implemented, and tested on a set of benchmarks.

The test results for various algorithms were analyzed and mutually compared. They were also compared with the results of the exact (implicit exhaustive) algorithm *LRed* and with the results obtained by transforming the problem to the two-level optimization problem and using Espresso to solve the latter.

The main conclusions resulted from those tests are the following:

- For small instances of the problem (up to 20 input bits), the exact algorithm implemented in *LRed* is the best one. It guarantees the optimal solutions and it is efficient enough. For larger instances, it is not efficient enough,
- For large instances, *QuickScan* is the best choice of the tested algorithms, because it is able to compute an optimal or almost optimal solution in extremely short time.
- Tabu Searches are not very suitable for MISP due to difficulty in finding a feasible neighborhood of a solution, what results in low quality of the search process.
- For the same reason, other algorithms based on an iterative traverse of the solution space (e.g., Simulated Annealing) are supposed to be not suitable.
- Local Search is substantially worse than Tabu Searches
- One may use Tabu Search on very large instances (many input bits) with a short list of incompatibilities (neighborhoods easy to compute), i.e. for large weakly specified functions, if *QuickScan* gives insufficiently good solutions and there is plenty of time.

- In the case when more accurate heuristic search than QuickScan is required, the best way seems to be the branch-and-bound strategy, driven by some heuristics based on the problem structure (especially on sophisticated preprocessing and probing of each node), and to stop computations at a given level of accuracy, if the computations are going to explode exponentially
- Translating the problem to the two-level optimization problem and using *Espresso* to solve the latter is not a good way; it results in inefficient computations which often cannot find an optimal solution.

Detailed remarks about the particular algorithms tested are summarized in the sections below.

LRed

The program seems to be quite well implemented and it is the best choice for small instances of the problem. However, to make the program fully usable in a CAD tool, several improvements have to be done:

- The discussion in Chapter *Problem Formulation* shows that very often MISP has a form more relaxed than one equivalent to PLA. To make the full advantage of *LRed*, the program should accept input specifying a compatibility set system (a rough set) on inputs.
- Since the program has exponential complexity, a stop condition would be very useful. The possible stop conditions are described on Page 31, but the absolute minimum is: (1) a given computation time elapsed, and (2) a given quality of a solution reached.
- The algorithm embedded in the program requires *ON*-set and *OFF*-set of PLA to be given, so the program calls internally *Espresso* to compute *OFF*-set for a given PLA. But if the program is given a PLA with the *OFF*-set explicitly defined (type *fr* of PLA), the call to *Espresso* is not necessary. However, *LRed* calls *Espresso* unconditionally. It can be worked around by supplying a dummy shell script, named "Espresso," that copies to the output the contents of the file given by the fourth parameter (input file name), but it is better to implement in *LRed* a sort of detection of PLA type or to supply an additional parameter which controls the call to *Espresso* (as it is done in *Misp*).
- It would be very useful if the program reported the quality of the best support found and the number of supports found. Now, it is worked around by a separate shell script that reads the output of the program and calculates the both values.

- Very often the program failed at the very end of computation (after the results were printed out) with the message "memory fault." It happened only in Unix systems, which are more restrictive for memory errors. The cause of the error is probably a corrupted dynamic data structure released at the end of computations or an error in the releasing procedure. This error is not very serious if the program makes a separate part of a CALD system, but as soon as it is embedded in a larger program, the error will crash the whole system.
- For a few examples, when the memory was low (in DOS) the program produced incorrect results instead of reporting the error and stopping the computations.
- For a class of very simple cases, the program crashes without any explanatory message.

Espresso

This option is totally outperformed by the other algorithms. It runs slower, can process smaller examples, requires more memory, gives worse solutions, the translated PLAs have a huge number of product terms, and the *Espresso* program often crashes due to internal errors (we have used Version 2.3, Release 01/31/88). Some of the problems are caused by the method itself (a large number of product terms in converted PLAs), but many of them result from the algorithms implemented in *Espresso*. Consequently, another two-level minimizer may exhibit better performance. In our opinion, however, it is not worthwhile to develop a specific two-level minimizer targeted to MISP; it is better to concentrate on solving MISP directly. Besides, it seems that the group which created *Espresso* has drawn similar conclusions, and developed more recently a branch-and-bound algorithm for Unate Row Cover Problem (which is very close to MISP) [33].

QuickScan

This algorithm has surprisingly good performance. It is very fast and gives quite good solutions. Without doubts, it is the best choice of the tested algorithms in the situations when time is at a premium and a solution is allowed to be near-optimal. It suffers, however, from the impossibility of fine-tuning the search: even if more time is given, it still produces the same "coarse" solutions. Further development of this algorithm should parametrize the search so that if more time is given, the search tries to be more precise. The guideline may be: first to define what potential global optima are skipped between two local minima of the scan, and then to perform a sub-search on the reduced search space.

Tabu Search

The results of the tests show that the Tabu Search approach is not so good as the existing exact algorithm, except for the case where the minimal support is much smaller than the original set of input bits and there are a lot of minimal solutions. The same result is expected for other heuristic search algorithms based on an iterative traverse of the solution space (e.g. Simulated Annealing). The reason why such heuristic searches do not perform the task well enough is the search space structure. There is no simple way to determine if a solution is feasible so there is also no simple way to create the proper neighborhood of a certain solution. In fact the procedure creating a neighborhood is the most time consuming in all the algorithms implemented. Additionally, there is a strong correlation between move operators and the quality of a solution; any move operator increases or decreases the solution quality by a unit. In this way, the whole search problem is not "where to find the best solutions that are feasible" but "where to find the feasible solutions that are good enough" and it seems that the heuristic searches of the considered type are not so efficient in the latter case than in the former.

Preprocessing and Probing

The test results showed that the cost of preprocessing (in the terms of computation time) is not high even for large examples. On the other hand, the gain in reduction of the problem size can be enormous, especially for exact algorithms. For fast heuristics, as *QuickScan*, this gain is less visible but still possible.

Proposal of new exact and approximate algorithms

Reasonable efficiency of the exact algorithm *LRed* for small and medium instances and the extremely high efficiency of the approximate algorithm *QuickScan*, also for large instances, motivated us to develop two new algorithms:

- an approximate **beam search algorithm** which will be more effective than *QuickScan* for larger instances.
- an exact **branch-and-bound algorithm** which will be more efficient than *LRed*, especially for larger instances,

Beam Search Algorithm

We convert the problem to the Column Cover Problem. The cover matrix has as many columns as the number of input symbols, and each row corresponds (1-1) to a unique incompatibility pair of symbols. The element $m_{i,j}$ is equal to 1 if the input symbol j allows

for distinguishing the incompatibility pair related to the row i, otherwise it is equal to 0. The information contained in the cover matrix may be viewed twofold:

- which columns cover which rows
- which rows are covered by which columns

The algorithm uses both types of information. At each step for each partial solution it performs the following operations:

- choose which column is to be included in the partial solution of the next step
- choose which column is not to be included in the partial solution of the next step
- choose a row to be covered by the partial solution of the next step

Because all rows have to be covered, the rows may only be selected; the columns may be selected or rejected.

The meta-heuristics controlling the application of operators are as follows:

- maximize the probability of obtaining the optimal solution from the actual partial solution, with regards to the available resources (beams)
- under the condition above, maximize the information for decision making in the next steps of the algorithm

Here we discuss the details of the applied operators. The selection of a row is a compound move, comprising selection of certain columns covering the considered row. The Selection of columns is determined by the probability of the quality of the columns, i.e. these columns are selected about which it cannot be stated with high probability, that those columns are worse than the others. In each step only MAXBEAMS columns are selected. For each selected column a new partial solution is created, having that column accepted as a cover column. The selection of one column (another operator) is in principle done in the same manner as above. The rejection of a column is done if it can be stated with high probability that the column is worse than the others. The detailed heuristics for application of the operators are as follows:

- accept those rows which have the least number of columns to be selected
- accept the subsets of columns which are small but much better than other subsets
- reject the subsets of columns which are small and much worse than other subsets
- penalize dominated or nearly-dominated columns
- prefer the columns which cover more rows and cover the rows that are more difficult to cover. The importance of a column for covering a given row decreases nonlinearly with the total number of column covering the considered row. The function

describing that importance can be modeled by a sort of hyperbolic function (equal to 0 if all columns cover the row, equal to plus infinity if only one column covers the row)

The first two heuristics should be combined together, the third is an alternative.

The quality of partial solutions may be calculated in the following manner:

- cumulative estimation of the quality of the decisions made so far plus the decisions predicted to be made in the future
- direct calculation of the potential quality of a partial solution by means of a quick search from the considered partial solution to a complete solution (*QuickScan*, best-first search, or another narrow search).

It is expected that the second way of calculating the quality will be faster and will give more accurate results.

We have already developed, implemented, and tested similar beam search algorithms for various decomposition [21, 23], state assignment [24], and test pattern generation [2] problems. In all the cases the algorithms are very effective and efficient. Therefore, we have omitted the implementation of the proposed algorithm as a part of the reported research.

Branch-and-Bound Algorithm

The paradigm of a branch-and-bound method is well known (see for example [33]). One has only to define ways for upper and lower estimation of the quality of a given branch. Here we propose *QuickScan* (or simple best-first search) as an estimation for the upper bound. The lower bound may be calculated as follows: let P_i be a set of columns which cover a given row i. The maximal number of mutually disjoint P_i makes the lower estimation. When choosing branches to consider, the heuristics developed for the beam search (described above) can be used.

Future Works

Future works should focus on implementing and testing the proposed exact branch-and-bound and approximate beam search algorithms. The best algorithms should be applied inside various algorithms for decomposition of sequential machines and switching functions.

References

[1] Ashenhurst R.L.
The Decomposition of Switching Functions.
In: Proceedings of an International Symposium on Theory of Switching (Part 2), April 2-5, 1957.
Cambridge, Mass.: Harvard University Press, 1959. P. 74-116.
(Annals of the Computation Laboratory of Harvard University, vol. 30).

[2] Bosch W.F.A.
Implementation of an OR-BDD Based TPG Algorithm for Combinational Circuits.
Section of Digital Information Systems, Faculty of Electrical Engineering, Eindhoven University of Technology, The Netherlands, 1994.
M. Sc. Graduation Report EB496.

[3] Brayton, R.K. and G.D. Hachtel, C.T. Mullen, A.L. Sangiovanni-Vincentelli.
Logic Minimization Algorithms for VLSI Synthesis.
Dordrecht: Kluwer, 1984.

[4] Brown F.M.
Boolean Reasoning.
Dordrecht: Kluwer, 1990.

[5] Chang S-C. and M. Marek-Sadowska.
BDD Representation of Incompletely Specified Functions.
In: International Workshop on Logic Synthesis, Granlibakken Conference Center, Iahoe City, CA, USA, May 23-26, 1993. P. P6c-1 – P6c-5.

[6] Curtis H.A.,
A New Approach to the Design of Switching Circuits.
Princeton: Van Nostrand, 1962.

[7] Dietmeyer D.L.
Logic Design of Digital Systems. Second Edition.
Boston: Allyn & Bacon, 1978.

[8] Fujita M. and Y. Matsunaga.
Multi-level Logic Minimization based on Minimal Support and its Application to the Minimization of Look-up Table Type FPGAs.
In: IEEE International Conference on Computer-Aided Design, Santa Clara, CA, USA, Nov. 11-14, 1992. Digest of Technical Papers.
Brussels: IEEE, 1991. P. 560-563.

[9] Garey, M.R. and D.S. Johnson.
Computers and Intractability. A Guide to the Theory of NP-Completeness.
New York: W.H. Freeman, 1979.

[10] Glover, F. and E. Taillard, D. de Werra.
A User's Guide to Tabu Search.
Annals of Operations Research, vol. 41 (1993), p. 3-28.

[11] Grinshpon M.S.
Selection Criterion for a Potentially Inessential Argument to Be Eliminated form an Incompletely-Specified Logical Function.
Automatic Control and Computer Sciences, vol. 9 (1975), no. 5, p. 16-18.

[12] Halatsis C. and N. Gaitanis.
Unredundant Normal Forms and Minimal Dependence Sets of a Boolean Function.
IEEE Transactions on Computers, vol. C-27 (1978), no. 11, p. 1064-1068.

[13] Hartmanis J.
Symbolic Analysis of a Decomposition of Information Processing.
Information and Control, vol. 3 (1960), p. 154-178.

[14] Hartmanis J.
Loop-free Structure of Sequential Machines.
Information and Control, vol. 5 (1962), p. 25-43.

[15] Hartmanis J. and R.E. Stearns.
Algebraic Structure Theory of Sequential Machines.
Englewood Cliffs, N.J.: Prentice-Hall, 1966.

[16] Hight, S.L.
Minimal Input Solutions.
IEEE Transactions on Computers, vol. C-20 (1971), no. 8, p. 923-925.

[17] Hill, F.J. and G.R. Peterson.
Computer Aided Logical Design. 4th ed.
New York: Wiley, 1993.

[18] Jasiński, K. and T. Łuba, J. Kalinowski.
Parallel Decomposition in Logic Synthesis.
In: Proceedings of the 15th European Solid State Circuits Conference, Vienna, Sep. 20-22, 1989.
Ed. by H. Grünbacker and G. Sandner.
München: Oldenbourgh, 1989. P. 113-116.

[19] Jóźwiak L.
The Bit Full-Decomposition of Sequential Machines.
Eindhoven: Faculty of Electrical Engineering, Eindhoven University of Technology, The Netherlands, 1989.
EUT Report 89-E-223.

[20] Jóźwiak L.
Simultaneous Decompositions of Sequential Machines.
Microprocessing and Microprogramming, vol. 30 (1990), p. 305-312.

[21] Jóźwiak L. and J.C. Kolsteren.
An Efficient Method for the Seuqential General Decomposition of Sequential Machines.
Microprocessing and Microprogramming, vol. 32 (1991), p. 657-664.

[22] Jóźwiak L. and A. P. H. van Dijk.
A Method for General Simultaneous Full-Decomposition of Sequential Machines: Algorithms and Implementation.
Eindhoven: Faculty of Electrical Engineering, Eindhoven University of Technology, The Netherlands, 1992.
EUT-Report 92-E-267.

[23] Jóźwiak L. and F. Volf.
An Efficient Method for Decomposition of Multiple Output Boolean Functions and Assigned Sequential Machines.
In: EDAC – The European Conference on Design Automation, Brussels, March 16-19, 1992.
Brussels: IEEE, 1992. P. 114-122.

[24] Jóźwiak L.
An Efficient Heuristic Method for State Assignment of Large Sequential Machines.
Journal of Circuits, Systems, and Computers, vol. 2 (1992), no. 1, p. 1-26.

[25] Jóźwiak L.
General Decomposition and Its Use in Digital Circuit Synthesis.
Accepted in 1994 for publication in: VLSI DESIGN: An International Journal of Custom-Chip Design, Simulation, and Testing.

[26] Kambayashi Y.
Logic Design of Programmable Logic Arrays.
IEEE Transactions on Computers, vol. C-28 (1979), no. 9, p. 609-617.

[27] Lin B. and A.R. Newton.
Exact Redundant State Registers Removal Based on Binary Decision Diagrams.
IFIP Transactions A (Computer Science and Technology) (1992), p. 277-286.

[28] Łuba T. and J. Kalinowski, K. Jasiński, H. Krasniewski.
Combining Serial Decomposition with Topological Partitioning for Effective Multi-level PLA Implementations.
In: Proceedings IFIP Working Conference on Logic and Architecture Synthesis, Paris, May 30-June 1, 1990.
Paris: IFIP, 1990. P. 77-86.

[29] Łuba, T. and J. Rybnik.
Algorithm of Elimination of Attributes and Arguments Based on Unate Complement Concept.
Bulletin of the Polish Academy of Sciences, vol. 40 (1992), no. 3, p. 313-322.

[30] Łuba T. and J. Rybnik.
Rough Sets and Some Aspects of Logic Synthesis.
In: Intelligent Decision Support. Edited by Roman Słowiński.
Dordrecht: Kluwer, 1992.

[31] Łuba T. and M. Markowski, B. Zbierzychowski.
Logic Decomposition for Programmable Gate Arrays.
In: Proceedings of Euro-ASIC '92, Paris, June 1-5, 1992.
Brussels: IEEE, 1992. P. 19-24.

[32] Matsunaga Y. and M. Fujita.
Multi-Level Logic Optimization Using Binary Decision Diagrams.
In: IEEE International Conference on Computer-Aided Design, Santa Clara, CA, USA, Nov. 5-9, 1989. Digest of Technical Papers.
Brussels: IEEE, 1989. P. 556-559.

[33] Rudell, R.
Logic Synthesis for VLSI Design.
Ph.D. dissertation, University of California, Berkeley, USA, April 1989.
Tech. Rep. UCB/ERL M89/49.

Eindhoven University of Technology Research Reports ISSN 0167-9708
Coden: TEUEDE

Faculty of Electrical Engineering

(263) Smolders, A.B.
RIGOROUS ANALYSIS OF THICK MICROSTRIP ANTENNAS AND WIRE ANTENNAS EMBEDDED IN A SUBSTRATE.
EUT Report 92-E-263. 1992. ISBN 90-6144-263-X

(264) Freriks, L.W. and P.J.M. Cluitmans, M.J. van Gils
THE ADAPTIVE RESONANCE THEORY NETWORK: (Clustering-) behaviour in relation with brainstem auditory evoked potential patterns.
EUT Report 92-E-264. 1992. ISBN 90-6144-264-8

(265) Wellen, J.S. and F. Karouta, M.F.C. Schemmann, E. Smalbrugge, L.M.F. Kaufmann
MANUFACTURING AND CHARACTERIZATION OF GAAS/ALGAAS MULTIPLE QUANTUMWELL RIDGE WAVEGUIDE LASERS.
EUT Report 92-E-265. 1992. ISBN 90-6144-265-6

(266) Cluitmans, L.J.M.
USING GENETIC ALGORITHMS FOR SCHEDULING DATA FLOW GRAPHS.
EUT Report 92-E-266. 1992. ISBN 90-6144-266-4

(267) Józwiak, L. and A.P.H. van Dijk
A METHOD FOR GENERAL SIMULTANEOUS FULL DECOMPOSITION OF SEQUENTIAL MACHINES: Algorithms and implementation.
EUT Report 92-E-267. 1992. ISBN 90-6144-267-2

(268) Boom, H. van den and W. van Etten, W.H.C. de Krom, P. van Bennekom, F. Huijskens, L. Niessen, F. de Leijer
AN OPTICAL ASK AND FSK PHASE DIVERSITY TRANSMISSION SYSTEM.
EUT Report 92-E-268. 1992. ISBN 90-6144-268-0

(269) Putten, P.H.A. van der
MULTIDISCIPLINAIR SPECIFICEREN EN ONTWERPEN VAN MICROELEKTRONICA IN PRODUKTEN (in Dutch).
EUT Report 93-E-269. 1993. ISBN 90-6144-269-9

(270) Bloks, R.H.J.
PROGRIL: A language for the definition of protocol grammars.
EUT Report 93-E-270. 1993. ISBN 90-6144-270-2

(271) Bloks, R.H.J.
CODE GENERATION FOR THE ATTRIBUTE EVALUATOR OF THE PROTOCOL ENGINE GRAMMAR PROCESSOR UNIT.
EUT Report 93-E-271. 1993. ISBN 90-6144-271-0

(272) Yan, Keping and E.M. van Veldhuizen
FLUE GAS CLEANING BY PULSE CORONA STREAMER.
EUT Report 93-E-272. 1993. ISBN 90-6144-272-9

(273) Smolders, A.B.
FINITE STACKED MICROSTRIP ARRAYS WITH THICK SUBSTRATES.
EUT Report 93-E-273. 1993. ISBN 90-6144-273-7

(274) Bollen, M.H.J. and M.A. van Houten
ON INSULAR POWER SYSTEMS: Drawing up an inventory of phenomena and research possibilities.
EUT Report 93-E-274. 1993. ISBN 90-6144-274-5

(275) Deursen, A.P.J. van
ELECTROMAGNETIC COMPATIBILITY: Part 5, installation and mitigation guidelines, section 3, cabling and wiring.
EUT Report 93-E-275. 1993. ISBN 90-6144-275-3

Eindhoven University of Technology Research Reports ISSN 0167-9708
Coden: TEUEDE

Faculty of Electrical Engineering

(276) Bollen, M.H.J.
LITERATURE SEARCH FOR RELIABILITY DATA OF COMPONENTS IN ELECTRIC DISTRIBUTION NETWORKS.
EUT Report 93-E-276. 1993. ISBN 90-6144-276-1

(277) Weiland, Siep
A BEHAVIORAL APPROACH TO BALANCED REPRESENTATIONS OF DYNAMICAL SYSTEMS.
EUT Report 93-E-277. 1993. ISBN 90-6144-277-X

(278) Gorshkov, Yu.A. and V.I. Vladimirov
LINE REVERSAL GAS FLOW TEMPERATURE MEASUREMENTS: Evaluations of the optical arrangements for the instrument.
EUT Report 93-E-278. 1993. ISBN 90-6144-278-8

(279) Creyghton, Y.L.M. and W.R. Rutgers, E.M. van Veldhuizen
IN-SITU INVESTIGATION OF PULSED CORONA DISCHARGE.
EUT Report 93-E-279. 1993. ISBN 90-6144-279-6

(280) Li, H.Q. and R.P.P. Smeets
GAP-LENGTH DEPENDENT PHENOMENA OF HIGH-FREQUENCY VACUUM ARCS.
EUT Report 93-E-280. 1993. ISBN 90-6144-280-X

(281) Di, Chennian and Jochen A.G. Jess
ON THE DEVELOPMENT OF A FAST AND ACCURATE BRIDGING FAULT SIMULATOR.
EUT Report 94-E-281. 1994. ISBN 90-6144-281-8

(282) Palkus, H.M. and A.A.H. Damen
MULTIVARIABLE H-INFINITY CONTROL DESIGN TOOLBOX: User manual.
EUT Report 94-E-282. 1994. ISBN 90-6144-282-6

(283) Meng, X.Z. and J.G.J. Sloot
THERMAL BUCKLING BEHAVIOUR OF FUSE WIRES.
EUT Report 94-E-283. 1994. ISBN 90-6144-283-4

(284) Rangelrooij, A. van and J.P.M. Voeten
CCSTOOL2: An expansion, minimization, and verification tool for finite state CCS descriptions.
EUT Report 94-E-284. 1994. ISBN 90-6144-284-2

(285) Roer, Th.G. van de
MODELING OF DOUBLE BARRIER RESONANT TUNNELING DIODES: D.C. and noise model.
EUT Report 95-E-285. 1995. ISBN 90-6144-285-0

(286) Dolmans, G.
ELECTROMAGNETIC FIELDS INSIDE A LARGE ROOM WITH PERFECTLY CONDUCTING WALLS.
EUT Report 95-E-286. 1995. ISBN 90-6144-286-9

(287) Liao, Boshu and P. Massee
RELIABILITY ANALYSIS OF AUXILIARY ELECTRICAL SYSTEMS AND GENERATING UNITS.
EUT Report 95-E-287. 1995. ISBN 90-6144-287-7

(288) Weiland, Siep and Anton A. Stoorvogel
OPTIMAL HANKEL NORM IDENTIFICATION OF DYNAMICAL SYSTEMS.
EUT Report 95-E-288. 1995. ISBN 90-6144-288-5

(289) Konieczny, Pawel A. and Lech Józwiak
MINIMAL INPUT SUPPORT PROBLEM AND ALGORITHMS TO SOLVE IT.
EUT Report 95-E-289. 1995. ISBN 90-6144-289-3