Coupling Logical Analysis of Data and Shadow Clustering for Partially Defined Positive Boolean Function Reconstruction

Marco Muselli, Member, IEEE, and Enrico Ferrari

Abstract—The problem of reconstructing the AND-OR expression of a partially defined positive Boolean function (pdpBF) is solved by adopting a novel algorithm, denoted by LSC, which combines the advantages of two efficient techniques, Logical Analysis of Data (LAD) and Shadow Clustering (SC). The kernel of the approach followed by LAD consists in a breadth-first enumeration of all the prime implicants whose degree is not greater than a fixed maximum $d$. In contrast, SC adopts an effective heuristic procedure for retrieving the most promising logical products to be included in the resulting AND-OR expression. Since the computational cost required by LAD prevents its application even for relatively small dimensions of the input domain, LSC employs a depth-first approach, with asymptotically linear memory occupation, to analyze the prime implicants having degree not greater than $d$. In addition, the theoretical analysis proves that LSC presents almost the same asymptotic time complexity as LAD. Extensive simulations on artificial benchmarks validate the good behavior of the computational cost exhibited by LSC, in agreement with the theoretical analysis. Furthermore, the pdpBF retrieved by LSC always shows a better performance, in terms of complexity and accuracy, with respect to those obtained by LAD.

Index Terms—Positive Boolean function, logic synthesis, Logical Analysis of Data, Shadow Clustering.

1 INTRODUCTION

The reconstruction of the AND-OR expression of a Boolean function starting from (a portion of) its truth table is a basic task in the realization of digital circuits. In general, the complexity of the resulting logical network is required to be as low as possible, which leads one to prefer techniques possessing the ability of retrieving functions with minimal or near-minimal expressions, according to some complexity measure.

A wide variety of algorithms has been proposed in the literature for this aim: from methods that perform an exhaustive search in the space of consistent Boolean functions, such as the Quine-McCluskey procedure [1] or the consensus method [2], to heuristic techniques that look for near-optimal solutions while reducing the computational cost. The algorithms ESPRESSO [3] and MINI [4] belong to this second class.

More recently, Boolean function reconstruction has also been considered for the solution of pattern recognition problems with binary inputs. In these cases, only a limited portion of the truth table, called training set, is available and the goal amounts to finding the Boolean function that minimizes some proper criterion while behaving correctly in the examples at hand. Possible criteria depend on simplicity, according to the Occam’s razor principle, or on some prior information about the problem to be solved.

Even machine learning problems with nonbinary inputs can be solved through the reconstruction of a Boolean function. It is sufficient to adopt suitable discretization techniques [5], [6], [7] and binary codings for mapping ordered and nominal variables (assuming values inside a finite, nonordered set, e.g., colors, geometrical shapes) into Boolean strings. For example, thermometer and only-one codes have been proved to preserve ordering and distance when used to transform ordered and nominal attributes, respectively, [8].

However, the dimension $n$ of the input space turns out to be very high ($n > 100$ or $n > 1000$), thus, preventing the adoption of exhaustive methods that retrieve the whole collection of AND-OR expressions consistent with the training set. On the other hand, heuristic techniques for logic synthesis generate Boolean functions with poor accuracy, since they do not take into account the generalization ability of the resulting solution.

Two different approaches have been suggested to deal with this situation. Following the principle of the Occam’s razor, the Logical Analysis of Data (LAD) [7] procedure generates all the logical products (AND operations) consistent with the training set, which include a number of operands not greater than a maximum $d$. Then, if the disjunction of these logical products fails to give the correct output for some examples of the training set, a simple heuristic is adopted to find other AND operations (with more than $d$ factors) that make the resulting Boolean function consistent with the whole training set.

On the other hand, the Hamming Clustering (HC) algorithm [9], [10] employs more refined heuristics for retrieving one at a time the logical products that contribute to form the resulting Boolean function. Different criteria can
be adopted to pursue different targets for the synthesis. For example, if the number of operands is to be minimized, the Lowest Cube criterion must be used, whereas the Maximum covering Cube criterion has the aim of increasing the number of examples in the training set correctly classified by each AND operation.

Now it can be seen that LAD cannot treat problems with high \( n \), since in its exhaustive part, LAD could have to examine \( O(n^d) \) different terms, which can require a huge computation time even for \( d = 4 \) when \( n \geq 1000 \). In addition, since LAD adopts a breadth-first enumerative technique to examine the logical products, even the memory requirement can become excessive. Finally, if the value of \( d \) is too small (as is the case when \( n \) increases), many examples of the training set remain misclassified at the end of the exhaustive part, leaving to the simple heuristic method, the heavy work of retrieving most of the needed AND operations.

The execution of extensive trials has shown that HC is able to achieve good performance, in terms of accuracy, for different values of \( n \), not suffering from computational limitations, since its execution time increases at most quadratically with \( n \) [9]. Nevertheless, whenever possible, the addition of an exhaustive part, as in LAD, can improve significantly the quality of the resulting Boolean function.

Thus, in this paper, we propose a new algorithm for reconstructing a partially defined Boolean function (pdpBf), which combines the exhaustive part of LAD with the heuristic from HC. To avoid an excessive memory allocation requirement, a depth-first approach is adopted in the exhaustive search; however, a particular housekeeping method controls the increase in the computational cost.

It has been verified [8] that a class of machine learning problems can be solved through partially defined positive Boolean function (pdpBf) reconstruction. Therefore, in this paper, we describe the version of the new algorithm dealing with pdpBf synthesis. Since the heuristic procedure (similar to HC) for deriving pdpBf is called Shadow Clustering (SC) [11], [12], the proposed technique is named LSC. Also standard pdBf reconstruction can be treated by LSC through the adoption of a proper mapping.

It is worth noting that positive Boolean functions, whose Disjunctive Normal Form (DNF) does not include negated variables, have recently attracted the attention of many researchers because of possible applications in the solution of several practical problems, ranging from the implementation of stack fillers [13], [14] to the choice of strategies in game theory [15]. In particular, the learnability of positive Boolean functions has been the subject of several papers [16], [17], [18], [19], [20], [21], [22]. In most of these works, the whole truth table is supposed to be available and can be queried through an oracle; in this case, the monotone dualization algorithm can be shown to reconstruct the desired function in quasipolynomial time [23]. Under this hypothesis, an efficient method for positive Boolean function synthesis has also been proposed in [24].

The remainder of the paper is structured as follows: basic definitions and notations introduced in Section 2 are employed to analyze the problem of reconstructing a pdpBf (Section 3). The general framework of the LAD approach is then presented in Section 4, together with implementation details regarding its application to pdpBf synthesis. Section 5 describes the proposed LSC technique, which combines an exhaustive part derived from a depth-first version of LAD with a heuristic part using SC.

The computational cost of LAD and LSC is theoretically examined in Section 6 through a worst-case analysis that derives the asymptotic behavior of the execution time and of the memory occupation. An experimental comparison of the performance exhibited by LAD and LSC on pdpBf and pdBf reconstruction is reported in Section 7. Finally, some discussions and conclusions are presented in Section 8.

2 BASIC DEFINITIONS AND NOTATIONS

Consider the set \( \{0, 1\}^n \) of the binary vectors with length \( n \); to improve readability, its elements will be denoted, henceforth, by strings of bits. The usual operations AND (logical product), OR (logical sum), and NOT (complementation) can be defined on it, making \( \{0, 1\}^n \) a Boolean algebra.

On the other hand, a standard partial ordering can be imposed on \( \{0, 1\}^n \) by writing \( x \leq y \) if and only if \( x + y = y \); with this definition, \( \{0, 1\}^n \) becomes a partially ordered set (or a poset). If \( x \) is the \( i \)th component of the binary vector \( x \), it can be easily seen that we have \( x < y \) if and only if \( x_j < y_j \) for some index \( j \) and \( x_i \leq y_i \) for every \( i \neq j \). Alternative orderings can be considered, such as the well-known lexicographic ordering, which sets \( x < y \) if and only if there exists an index \( j \) such that \( x_j < y_j \) and \( x_i = y_i \) for every \( i = 1, \ldots, j - 1 \).

The following definitions will be used henceforth:

**Definition 1.** Let \((X, \leq)\) be a poset. Then, an antichain is a subset \( A \subset X \) such that for any \( x, y \in A \) with \( x \neq y \), neither \( x < y \) nor \( y < x \) holds.

**Definition 2.** The lower shadow of \( a \subset X \) is the poset \((L(a), \leq)\), being \( L(a) = \{x \in X : x \leq a\} \). Likewise, the upper shadow of \( a \subset X \) is the poset \((U(a), \leq)\), where \( U(a) = \{x \in X : x \geq a\} \).

It directly follows from Definition 2 that \( b \in L(a) \) if and only if \( a \in U(b) \).

The diagram of the poset \( \{0, 1\}^n \) can be obtained by drawing a node for each of its elements, placing \( x \) lower than \( y \) whenever \( x < y \). Then, an edge is added to connect each pair \( x, y \in \{0, 1\}^n \) for which \( x < y \) and no \( a \in \{0, 1\}^n \) exists such that \( x < a < y \). When the standard ordering is used, the diagram of \( \{0, 1\}^3 \) is shown in Fig. 1a. An example of the lower and upper shadows for \( a = 101 \) is then represented in Figs. 1b and 1c, respectively.
Now consider the class $B_n$ of Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$. Each of them induces a separation of $\{0,1\}^n$ into two subsets $D_0(f)$ and $D_1(f)$, called offset and onset, respectively:

\[D_0(f) = \{ x \in \{0,1\}^n : f(x) = 0 \},\]
\[D_1(f) = \{ x \in \{0,1\}^n : f(x) = 1 \}.

A function of $B_n$ can always be written as a Boolean expression, involving literals, i.e., variables $x_i$ or their complement $\bar{x}_i$, constant terms 0, 1, and the operators AND, OR, NOT. Denote by $I_n$ the set $\{1,2,\ldots,n\}$ of the first $n$ positive integers; if $J$ and $K$ are two subsets of $I_n$, a logical product $t(x) = \bigwedge_{j \in J} x_j \bigwedge_{k \in K} \bar{x}_k$ is called term. It covers a point $x \in \{0,1\}^n$ if $t(x) = 1$; the degree of the term $t$ is given by the number $|J| + |K|$ of literals in it, having denoted by $|J|$ the cardinality of the set $J$.

If $f$ and $g$ are two Boolean functions in $B_n$, we write $f \leq g$ if and only if $f(x) \leq g(x)$ for every $x \in \{0,1\}^n$. An implicant of $f$ is a term $t \leq f$ if this relation is no more true when any literal is removed from $t$, the term $t$ is called a prime implicant (or a minterm) of $f$.

It is known that every Boolean function can be written in DNF, i.e., as a logical sum of implicants

\[f(x_1, \ldots, x_n) = \bigvee_{i=1}^h \bigwedge_{j \in J_i} x_j \bigwedge_{k \in K_i} \bar{x}_k, \tag{1}\]

where $J_i \cap K_i = \emptyset$ for $i = 1, \ldots, h$, having set by definition $\bigwedge_{j \in J_i} x_j = 1$ and $\bigwedge_{j \in J_i} x_j = 0$ if $J = \emptyset$. In general, the DNF of a Boolean function $f$ is not unique, even if only prime implicants are included in it.

If the complement operator cannot be employed in Boolean expressions, only the class $L_n$ of (monotone) positive Boolean functions can be realized. If $f \in L_n$, we have that $x \leq y$ implies $f(x) \leq f(y)$, for every $x, y \in \{0,1\}^n$.

Consider the mapping $P : \{0,1\}^n \rightarrow 2^I$ defined as $P(a) = \{ i \in I_n : a_i = 1 \}$; it produces the subset of indices of the components $a_i$ assuming value 1. The inverse of $P$ will be denoted by $p_i$; for any subset $J \subset I_n$, it gives the element $p_i(J) \in \{0,1\}^n$ whose $i$th component $p_i(J)$ has value 1 if and only if $i \in J$. In a similar way, we can define the complementary mapping $S(a) = I_n \setminus P(a)$, which gives the subset of indices of the components $a_i$ assuming value 0, and its inverse $s(J) = p(I_n \setminus J)$.

It can be shown that for every positive Boolean function $f$, there exists a unique antichain $A$ of the poset $\{0,1\}^n$ such that $f$ can be written through the following expression:

\[f(x_1, \ldots, x_n) = \bigvee_{a \in A} \bigwedge_{j \in P(a)} x_j, \tag{2}\]

which is called irredundant Positive Disjunctive Normal Form (PDNF) of the function $f$. Every element $a \in A$ is called a minimum true point of $f$ and is associated with the prime implicant $\bigwedge_{j \in P(a)} x_j$ through the mapping $P$. Note that the onset of this prime implicant is given by the points $x \geq a$, which belong to the upper shadow $U(a)$. They can be obtained by changing some of the components of $a$ from 0 to 1.

### 3 Partially Defined Positive Boolean Function Reconstruction

The problem of reconstructing a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ starting from its truth table (logic synthesis) has received great attention in the science literature, since it is the fundamental task in the design of digital circuits. Several techniques have been proposed to deal with this problem [1], [2], [3], [4], [24], [25]; in general, they aim to find the prime implicants to be included in the DNF expression for the unknown function $f$.

However, since the DNF is not unique, the choice of the prime implicants to be considered is performed according to some specific goal, such as the minimization of the complexity of the Boolean expression. Possible measures of complexity are the number of implicants or the total number of literals in the DNF.

This optimization process can require an excessive computational burden, at least when the number $n$ of inputs increases; hence, suboptimal algorithms [3], [4] have been proposed, which find a good DNF expression within a reasonable execution time. One of the most used methods of this kind is ESPRESSO [3], currently employed in the synthesis of custom digital circuits.

The same approach can be adopted when only a reduced portion of the truth table for the unknown Boolean function $f$ is available. In this case, we speak of pdBF and some of the above-cited optimal algorithms, e.g., spectral techniques [25], can still be used only if a preliminary extension of the available information to the whole input space is performed. However, it should be observed that the minimization of the DNF expression heavily depends on the method adopted for the preliminary extension.

A good compromise between optimal and suboptimal algorithms is offered by the LAD approach [7]: through an exhaustive search, it generates all the prime implicants having degree not greater than a maximum $d$. Then, LAD adopts a simple heuristic to complete the collection of minterms for the reconstruction of the desired DNF. By properly choosing the value of $d$, a good trade-off between solution optimality and computational cost can be achieved.

Nevertheless, when the dimension $n$ of the input domain is large ($n > 100$ or $n > 1000$), even a small $d$ can increase excessively the execution time and the memory occupation required by LAD. To overcome this problem, the approach offered by HC [9] can be employed. It has no exhaustive part, but the prime implicants are generated one at a time, clustering points around the known examples of the truth table through the adoption of clever heuristics.

Here, a new technique for pdBF synthesis is proposed; it is obtained by combining the approaches of LAD and HC. In particular, the version devoted to the reconstruction of pdBFs is described. It is called LSC, since SC is the algorithm similar to HC for the synthesis of positive Boolean functions. The extension to pdBF reconstruction can be readily obtained, although it is shown that a simple input mapping allows LSC to be used even for the general Boolean function synthesis without any loss in computational performance.

The pdBF reconstruction problem can be formalized as follows:

**pdBF reconstruction problem.** Given two subsets $T, F$ of points in $\{0,1\}^n$, with $T \cap F = \emptyset$, we want to find (if it
poset for pdpBf reconstruction is to find the antichain into account the number $P$ bottom point from $T$ sure in machine learning is given by the number of points in important also when a pattern recognition problem is to be binary strings of pdpBf reconstruction problem by searching for promising implicants to be included in the antichain accepting a small amount of error when constructing prime syntheses are needed to produce a satisfactory solution for the problem at hand.

**Definition 3.** Let $A, B$ be two subsets of $\{0, 1\}^n$. $A$ covers $B$ iff, for every $b \in B$, there is an $a \in A$ such that $a \leq b$.

A is lower (resp., upper) separated from $B$ iff, for every $x \in A$, there is no $y \in B$ such that $x \leq y$ (resp., $x \geq y$). $A$ and $B$ are separated from each other iff $A$ is both lower and upper separated from $B$.

According to these definitions, the goal of an algorithm for pdpBf reconstruction is to find the antichain $A$ of the poset $\{0, 1\}^n$, which minimizes a selected complexity measure while covers $T$ and is separated from $F$. Starting from $A$, the irredundant PDNF of the desired Boolean function $f$ is readily written by using (2).

Note that if there are $x \in T$ and $y \in F$ such that $x \leq y$, a positive Boolean function $f$ satisfying $T \subseteq D_1(f)$ and $F \subseteq D_0(f)$ cannot exist. Thus, the prerequisite for pdpBf synthesis is that $T$ must be lower separated from $F$. However, in a real-world problem, the presence of noise can lead to a violation of this requirement. In such cases, the approaches described in this paper can still be applied by accepting a small amount of error when constructing prime implicants to be included in the antichain $A$.

The complexity of a PDNF is usually evaluated by taking into account the number $|A|$ of minterms and the number $\sum_{a \in A} |P(a)|$ of literals included in it. These measures are important also when a pattern recognition problem is to be solved, due to the Occam’s razor principle, which asserts that a simpler function usually presents a better generalization ability. However, another important quality measure in machine learning is given by the number of points in $T$ covered by each prime implicant; in fact, each minterm can be viewed as an intelligible rule, whose relevance depends on how many cases it explains.

When $T \cup F = \{0, 1\}^n$, the positive Boolean function $f$ to be retrieved is completely defined and the uniqueness of the irredundant PDNF ensures that there is only one solution to the problem at hand. This solution can be found by determining all the minimum true points of $f$ to be included in the antichain $A$ of expression (2).

In the opposite case, when $T \cup F$ is a proper subset of $\{0, 1\}^n$ and $T$ is lower separated from $F$, we can solve the pdpBf reconstruction problem by searching for promising binary strings of $\{0, 1\}^n$, which will become the minimum true points of the resulting positive Boolean function $f$. The following definition provides the characterization of these elements:

**Definition 4.** A binary string $a \in \{0, 1\}^n$ will be called a bottom point for the pair $(T, F)$ iff the following three conditions hold:

1. at least a point of $T$ is in the upper shadow of $a$,
2. $a$ is not in the lower shadow of any point in $F$; and
3. for every $b \leq a$, there is $y \in F$ such that $b \leq y$.

With a slight abuse of notation, the expression degree of a bottom point $a$ will indicate the degree of the term $\bigwedge_{j \in P(a)} x_j$, which is equal to the cardinality of $P(a)$.

**LOGICAL ANALYSIS OF DATA (general procedure)**

1. $A = \text{Exhaustive}(T, F, d)$.
2. Let $S$ be the set of patterns $x \in T$ such that there is no $a \in A$ with $a \leq x$.
3. If $S$ is empty then return $A$.
4. Pick $x \in S$.
5. $a = \text{Heuristic}(x, T, F, S, A)$.
6. Add the prime implicant $a$ to $A$.
7. Remove from $S$ all the patterns $z$ for which $a \leq z$ and go to Step 3.

![Fig. 2. General procedure followed by LAD for reconstructing positive Boolean functions.](image)

It can be easily seen that the collection of the bottom points for the pair $(T, F)$ forms an antichain of the poset $\{0, 1\}^n$. Hence, it follows that the prime implicants of $f$ could be obtained by examining all the bottom points $a$ for the pair $(T, F)$. However, if $n > 20$, a thorough analysis can require an excessive computational cost and approximations are needed to produce a satisfactory solution for the problem at hand.

**4 THE LAD APPROACH**

A good compromise between solution optimality and computational cost is offered by the LAD approach, whose general procedure is shown in Fig. 2. At the end of Step 1, the function Exhaustive returns the set $A$ including all the bottom points with degree not greater than a chosen maximum $d$. Then, the collection $S$ of patterns in $T$, not covered by any prime implicant in $A$, is considered at Step 2. Starting from each $x \in S$ (Step 4), a new bottom point $a$, lying in the lower shadow of $x$, is generated by following a proper heuristic, performed by the function Heuristic at Step 5.

The points covered by the new prime implicant $a$ are subsequently removed from $S$ (Step 7) and the procedure Heuristic is repeated until $S$ becomes empty. Note that it is always possible to find a prime implicant $a$ lying in the lower shadow of any $x \in T$ provided that $T$ is lower separated from $F$.

This general scheme allows us to select different implementations for the exhaustive and the heuristic parts. In particular, the original LAD procedure proposes to adopt for the function Exhaustive a breadth-first enumerative technique, which can be easily modified for the case of pdpBf reconstruction, as shown in Fig. 3.

The kernel of the function is the loop at Step 2 that analyzes during each iteration all the bottom points having degree $l$. It employs two sets of patterns $B$ and $C$: they include all the binary strings, which cover some points both in $T$ and $F$ and have value 1 in $l − 1$ and $l$ bits, respectively. Consequently, the set $C$ for the current iteration will become the set $B$ for the next one, as performed at Step 2c. The bottom points found during each iteration are subsequently added to the set $A$ (Step 2bbcaaa).
Initially, $B$ contains only the pattern $p(\emptyset)$ having all the bits with value 0 (Step 1). Then, during each iteration, the binary strings of $B$ are examined, and for each of them, the last run of 0s is considered (Step 2ba). The bits of this run, whose index lies between $k+1$ and $n$, are set to 1 one at a time, thus, obtaining new patterns having $l$ bits with value 1 (Step 2bbb).

However, if $l>1$, each of these patterns can be a valid bottom point only if the set $B$ includes all the binary strings obtained by it after changing a bit from 1 to 0. This is verified through the loop at Steps 2bbaa-2bbac.

The if-then statements at Steps 2bbc and 2bbca have the aim of checking if each new valid pattern covers points in $T$ and $F$, respectively. According to the results of these control statements, the pattern is inserted into $A$ (if it covers some points in $T$ and no elements of $F$), into $C$ (if it covers points both in $T$ and $F$), or discarded (if it does not cover any point of $T$).

The loop at Step 2bb ensures that each pattern with $l$ bits to 1 is considered only once. As a matter of fact, since all the binary strings in $B$ are different among them, the addition of value 1 in the last run of 0s cannot give rise to the same pattern.

Initially, $B$ contains only the pattern $p(\emptyset)$ having all the bits with value 0 (Step 1). Then, during each iteration, the binary strings of $B$ are examined, and for each of them, the last run of 0s is considered (Step 2ba). The bits of this run, whose index lies between $k+1$ and $n$, are set to 1 one at a time, thus, obtaining new patterns having $l$ bits with value 1 (Step 2bbb).

However, if $l>1$, each of these patterns can be a valid bottom point only if the set $B$ includes all the binary strings obtained by it after changing a bit from 1 to 0. This is verified through the loop at Steps 2bbaa-2bbac.

The if-then statements at Steps 2bbc and 2bbca have the aim of checking if each new valid pattern covers points in $T$ and $F$, respectively. According to the results of these control statements, the pattern is inserted into $A$ (if it covers some points in $T$ and no elements of $F$), into $C$ (if it covers points both in $T$ and $F$), or discarded (if it does not cover any point of $T$).

The loop at Step 2bb ensures that each pattern with $l$ bits to 1 is considered only once. As a matter of fact, since all the binary strings in $B$ are different among them, the addition of value 1 in the last run of 0s cannot give rise to the same pattern.

When the exhaustive part is completed, the original implementation of LAD uses a simple heuristic procedure to retrieve the remaining bottom points to be inserted in $A$ (if needed). Again, it can be directly modified to deal with pdpBf reconstruction as reported in Fig. 4.

The bits of $x$ having value 1 are changed one at a time (Step 2a), verifying that the resulting pattern does not cover any point of $F$. The set $J$ contains the indices of the components that cannot be switched. It should be noted that the final bottom point $p(J)$ depends on the order followed when selecting the bits of $x$ to be changed. A random choice is normally performed, since the exhaustive part is supposed to have found all the relevant prime implicants. However, this is not true when the dimension $n$ of the input domain is high, so that only small values of $d$ can be considered without increasing excessively the computational burden.

5 THE PROPOSED LSC TECHNIQUE

As previously noted, the methods adopted by the original version of LAD for implementing the functions Exhaustive and Heuristic present some drawbacks that can make the performance of the approach worse, especially when the length $n$ of the binary strings is high. In particular, the breadth-first technique adopted in Fig. 3 may lead to an excessive increase in memory occupation, since the number of patterns having $l$ bits with value 1 is $O(n^l)$.

To recover this problem, the proposed LSC technique carries out the exhaustive part of Fig. 2 (Step 1) by adopting a depth-first approach, which significantly reduces the memory request. Then, the SC procedure is employed in the heuristic part of LSC to retrieve the near-optimal prime implicants starting from a pattern $x \in T$ not covered by the bottom points included in the current antichain $A$.

5.1 LSC: Exhaustive Part

The exhaustive part of LSC simply consists in a call to the procedure Depth shown in Fig. 5 with parameters $I = I_n$, $J = \emptyset$, and $A = \emptyset$.

All the parameters in the argument to the call are passed by reference (or can be considered as global variables). In particular, the two sets $I$ and $J$ contain the indices of the bits with value 0 in the currently examined pattern $s(I \cup J)$. By construction (Step 2a), the elements of $J$ are always greater than those of $I$; it should be noted that Steps 1 and 3...
Fig. 5. The recursive depth-first enumerative technique followed by LSC for generating the consistent prime implicants for a pdpBf.

allow us to preserve the consistency of these two sets after a call to Depth.

The single loop at Step 2 forms the main body of the procedure: the greatest element of $I$ is removed from it (i.e., the corresponding bit is set), checking if the resulting binary string $s(I \cup J)$ covers some points of $T$. Then, if it is also not contained in the lower shadow of any $y \in F$ and is actually a bottom point for the pair $(T, F)$ (i.e., an element $a \in A$ with $a \leq s(I \cup J)$ cannot be found), the pattern $s(I \cup J)$ is added to the antichain $A$ (Step 2baa).

In the opposite case, further bits are set through a new call to the procedure Depth (Step 2baA) that provides the number of 0s in the examined pattern does not fall below the required minimum $n - d$.

For a better understanding of the behavior of the procedure Depth, let us analyze the following pdpBf reconstruction problem:

**Example 1.** Consider the sets

$$
T = \{00111, 01101, 01110, 10011\},
$$

$$
F = \{00001, 00010, 00101, 01001\},
$$

containing binary strings with $n = 5$ bits, obtained through a random sampling of the positive Boolean function

$$
 f(x_1, \ldots, x_5) = x_1x_2 + x_1x_4x_5 + x_3x_4 + x_2x_3x_5,
$$

and suppose to perform the exhaustive part of LSC by calling the procedure Depth with $A = \emptyset$, $I = \{1, 2, 3, 4, 5\}$, $J = \emptyset$, and $d = 2$.

After having set $k = 6$ at Step 1, the first execution of Step 2a removes the index 5 from $I$, thus, producing the pattern $s(I \cup J) = 00011$. It can be easily seen that this binary string lies in the lower shadow of 00111 if $T$ and 00011 if $F$; therefore, since $I$ is not empty and $|I \cup J| = 4 > 3 = n - d$, a new call to Depth is done at Step 2baA. Again the local variable $k$ is set to 6 since $J = \emptyset$; then, at Step 2a, the element $i = 4$ is deleted from $I$. It follows that $s(I \cup J) = 00011$, which covers both 00111 in $T$ and 01011 in $F$; consequently, it cannot be a prime implicant.

Now, since $|I \cup J| = n - d = 3$, a new call to Depth is not permitted and Step 2c is executed, putting $i = 4$ into $A$. A new iteration of the loop starts by removing the index 3 from $I$, leading to the current pattern $s(I \cup J) = 00101$. Since it lies in the lower shadow of 00111 in $T$ and 01011 in $F$, it cannot be included in $A$. The subsequent iterations of the loop examine the binary strings 01001 and 10001, concluding that neither of them can be implicants since they cover points in $F$.

The execution of the second call to Depth is thus finished and the control returns to Step 2c of the calling procedure, where the index $i = 5$ is added as the first element to the set $J$. Then, a new iteration is performed by removing the value 4 from $I$; the pattern $s(I \cup J) = 00101$ is determined in this way and a new call to Depth is done at Step 2baA with $I = \{1, 2, 3\}$ and $J = \{5\}$.

After having set $k = 5$ at Step 1, the main loop begins by deleting the index $i = 3$ from $I$. The resulting binary string $s(I \cup J) = 00110$ covers only points in $T$, and therefore, can be included in the set $A$. The execution continues along the same lines and returns at the end of the first call to Depth the antichain $A = \{00110, 01100, 11000\}$. Note that only the binary string 10011 is not covered by prime implicants of $A$.

5.2 LSC: Heuristic Part

The simple heuristic method adopted by LAD and shown in Fig. 4 is essentially a greedy technique that starts from a specific point $x \in T$ and subsequently switches on at a time some of its bits from 1 to 0, while ensuring that the lower shadow of any element of $F$ is not reached. However, the way adopted to select the index $i$ to be removed from $I$ at Step 2a is crucial for the quality of the bottom points produced by this approach.

The SC procedure offers a systematic way for performing the selection of the index $i$ when the goal is to minimize a desired quality factor measuring the complexity of the final irredundant PDNF. In particular, the size of the antichain $A$ and the number of literals $\sum_{a \in A} P(a)$ in expression (2) are two important quantities that must be minimized to make the resulting positive Boolean function simpler.

To pursue this aim, SC adopts two possible criteria: the first one, called Maximum covering Shadow Clustering (MSC), tries to increase the number of points in $S$ and $T$ covered by every generated bottom point; the second one, called Deepest Shadow Clustering (DSC), manages to reduce the degree of the prime implicant produced. Other quality factors can be minimized through SC by employing different ad hoc criteria; an analysis of some interesting choices will be the subject of a forthcoming paper. It should be noted that the goal of MSC is very important when the algorithm for pdpBf reconstruction is used in the solution of
machine learning problems. In fact, any bottom point with high covering represents a relevant rule inferred from data.

The approach followed by MSC consists in switching from 1 to 0 the component \( x_i \) of the selected point \( x \in S \), which does not lead to a conflict with the patterns of \( F \) and maximizes a measure of the potential covering associated with the index \( i \). This is given by the cardinality of the set \( S_0^i \), having defined for every \( Z \in \{0,1\}^n \),

\[
Z_0^i = \{ z \in Z : z_i = 0 \}, \quad Z_1^i = \{ z \in Z : z_i = 1 \}.
\]

It can be easily seen that at every execution of Step 2a of the algorithm in Fig. 4, the point \( p(I \cup J) \) does not cover any binary string in \( S_0^i \) for \( i \in I \cup J \). Then, by switching the \( i \)th bit associated with a high value of \( |S_0^i| \), we increase the probability of having many elements of \( S \) in the upper shadow of the new pattern reached by the algorithm.

However, since the size of \( S \) decreases during the execution of SC, bottom points that cover few points of \( T \) can be generated when \( |S| \) becomes small, due to the presence of many binary strings whose upper shadows include the same number of elements in \( S \). To perform a better choice, the selection of the index \( i \) can also be driven by the size \( |T_0^i| \), which takes into account the potential covering of the whole set \( T \).

The alternative approach followed by the DSC criterion tries to minimize the degree of the resulting bottom point by choosing the indices \( i \in I \) that keep as large as possible the distance of the current pattern from the lower shadows of the points in \( F \). A proper measure of distance is offered by the following definition:

**Definition 5.** Let \( x, y \) be two points of the poset \( \{0,1\}^n \). The lower distance \( d_l(x, y) \) between \( x \) and \( y \) is defined as

\[
d_l(x, y) = \sum_{i=1}^n |x_i - y_i|_+,
\]

where

\[
|z|_+ = \begin{cases} z, & \text{if } z \geq 0, \\ 0, & \text{otherwise}. \end{cases}
\]

In the same way, the upper distance \( d_u(x, y) \) between \( x \) and \( y \) is given by

\[
d_u(x, y) = \sum_{i=1}^n |y_i - x_i|_+.
\]

In few words, \( d_l(x, y) \) is equal to the number of indices \( i \) for which \( x_i = 1 \) and \( y_i = 0 \), whereas \( d_u(x, y) \) accounts for the indices \( i \) that give \( x_i = 0 \) and \( y_i = 1 \). Note that \( d_l(x, y) = 0 \) if and only if \( x \leq y \), whereas \( d_u(x, y) = 0 \) if and only if \( x \geq y \).

It should also be observed that neither \( d_l \) nor \( d_u \) provides a (pseudo)metric for \( \{0,1\}^n \), since they are not symmetric, but \( d_l(x, y) = d_u(y, x) \).

By employing the usual definition of distance between a point \( x \) and a set \( Z \) (given by the minimum distance between \( x \) and the elements of \( Z \)), we obtain that the lower distance \( d_l(x, Z) \) is defined as

\[
d_l(x, Z) = \min_{y \in Z} d_l(x, y).
\]

Similarly, \( d_u(x, Z) = \min_{y \in Z} d_u(x, y) \).

**LSC (heuristic part)**

Function **Heuristic**(\(x, T, F, S, A\))

1. Set \( I = P(x) \) and \( J = \emptyset \).
2. For each \( i \in I \) compute \( |S_0^i| \) and \( |T_0^i| \).
3. While \( I \) is not empty do
   3a. For each \( i \in I \) compute \( d_l(p(I \cup J), F_0^i) \).
   3b. Move from \( I \) to \( J \) all the indices \( i \) for which \( d_l(p(I \cup J), F_0^i) = 1 \). If \( I \) becomes empty, then go to Step 4.
   3c. Remove from \( I \) the index \( i \) that maximizes the cost vector
      - \( (|S_0^i|, |T_0^i|, d_l(p(I \cup J), F_0^i)) \) for MSC,
      - \( (d_l(p(I \cup J), F_0^i), |S_0^i|, |T_0^i|) \) for DSC.
4. Return \( p(J) \).

Fig. 6. The heuristic procedure followed by LSC for generating a prime implicant covering a given point \( x \in T \) in the reconstruction of a pdpBlf.

It follows that every bottom point \( a \) for the pair \((T, F)\) satisfies the condition \( d_l(a, F) = 1 \). Thus, the approach adopted by DSC for selecting the index \( i \in I \) consists in maximizing the distance \( d_l(p(I \cup J), F_0^i) \), being \( p(I \cup J) \) the currently considered pattern.

Since the goals of MSC and DSC are not in conflict, both of them can be adopted in the selection of the index \( i \in I \) simply by imposing an order in the optimization of the different measures. In this way, MSC maximizes the vector \( (|S_0^i|, |T_0^i|, d_l(p(I \cup J), F_0^i)) \), where the first component \( |S_0^i| \) is considered more relevant than the second one \( |T_0^i| \), and this, in turn, is more important than the distance \( d_l(p(I \cup J), F_0^i) \).

On the other hand, the DSC criterion adopts the vector \( (d_l(p(I \cup J), F_0^i), |S_0^i|, |T_0^i|) \) with the associated preference order.

Fig. 6 shows in detail the algorithm adopted by MSC and DSC for performing the heuristic part of the LSC technique. This algorithm can be properly modified to take into account different quality measures.

At Step 2, the values of \( |S_0^i| \) and \( |T_0^i| \) for every \( i \in I \) are computed; these values remain unchanged during the execution of the procedure. Then, at Step 3, the basic iteration of the procedure takes place until the set \( I \) becomes empty. The distances \( d_l(p(I \cup J), F_0^i) \) between the current pattern \( p(I \cup J) \) and the sets \( F_0^i \) are evaluated (Step 3a) and the indices \( i \in I \) with \( d_l(p(I \cup J), F_0^i) = 1 \) are moved to the set \( J \) (Step 3b), since they cannot be switched from 1 to 0 without creating a conflict with the elements of \( F \).

Now, if \( I \) becomes empty, the iteration is interrupted and the function returns \( p(J) \) at Step 4. Otherwise, the index \( i \) that maximizes the suitable vector is removed from \( I \) at Step 3c; this is equivalent to clear the corresponding bit in the current pattern \( p(I \cup J) \).

An example may help to illustrate the functioning of the algorithm and the difference between MSC and DSC.

**Example 2.** Consider the problem of reconstructing the positive Boolean function...
from the sets
\[ T = \{000111, 100110, 101010, 101100, 110100, 110110\}, \]
\[ F = \{010101, 011100, 100101, 110101, 110110, 111000\}, \]
containing binary strings with \( n = 6 \) bits. Suppose that the exhaustive part of LSC has not been performed by setting \( d = 0 \); it follows that \( A = \emptyset \) and \( S = T \) at Step 3 of the general procedure in Fig. 2. If at Step 4 the point \( x = 000111 \) is selected, the procedure Heuristic in Fig. 6 sets at Step 1 \( I = \{4, 5, 6\} \) and \( J = \emptyset \). Then, at Step 2, we obtain
\[ S_0^0 = T_4^0 = \{101010, 101100\}, \]
\[ S_0^1 = T_5^0 = \{101100\}, \]
\[ S_0^2 = T_6^0 = \{010101, 101010, 101100, 110100\}, \]
which gives \( |S_0^0| = |T_4^0| = 2 \), \( |S_0^1| = |T_5^0| = 1 \), and \( |S_0^2| = |T_6^0| = 4 \).

Now, the main loop starts by computing at Step 3a
\[ F_0^0 = \{111000\}, \quad F_0^1 = F, \]
\[ F_0^2 = \{011100, 110100, 111000\}, \]
which leads to the distances \( d_i(p(I \cup J), F_0^0) = 3 \), \( d_i(p(I \cup J), F_0^1) = 1 \), and \( d_i(p(I \cup J), F_0^2) = 2 \).

The execution of Steps 3b yields \( J = \{5\} \) and \( I = \{4, 6\} \); consequently, a specific criterion is to be employed at Step 3c to select the index \( i \) to be removed from \( I \). It is straightforward to observe that the maximization of the vector \( (|S_0^0|, |T_0^0|, d_i(p(I \cup J), F_0^0)) \), performed by MSC, leads to the choice \( i = 6 \), whereas the application of \( (d_i(p(I \cup J), F_0^1), |S_0^0|, |T_0^0|) \), followed by DSC, selects \( i = 4 \).

Now, in this simple example, both approaches produce the same bottom point, since in the next iteration, the binary string \( p(J) = 000010 \) is generated both by MSC and DSC and returned to the calling program. Nevertheless, in more complex situations, the antichain \( A \) obtained with the two methods may differ significantly.

The set \( S \) is then simplified at Step 7 of Fig. 2 by removing from it the points covered by the pattern 000010 just inserted into \( A \). After this step, \( S \) contains the single element 101100 and a subsequent execution of the function Heuristic shows that this binary string cannot be further simplified and is to be added to \( A \). The final antichain is therefore \( A = \{000010, 101100\} \), which leads to the irredundant PDNF (5).

Intensive trials have shown that MSC achieves a higher level of accuracy with respect to DSC without increasing the computational burden, and therefore, will be employed in the tests presented in Section 7.

### 5.3 LSC: Antichain Simplification

The set \( A \) generated according to the procedure described in the previous sections, contains consistent bottom points for the pdpBf reconstruction problem at hand. In fact, every pattern in \( A \) covers some points of \( T \) and no elements of \( F \). However, there can be several bottom points in \( A \) that cover the same point \( x \) of \( T \) and, in general, different subsets of \( A \) can be found, which cover the whole set \( T \).

Thus, the final phase of the pdpBf synthesis consists in selecting the subset of \( A \) that minimizes the desired quality measure. In particular, if complexity minimization is one of the goals, the subset \( A^* \) with the lowest size is to be chosen.

If \( c = |A| \) is the number of binary strings in the antichain \( A \), the subset \( A^* \) can be retrieved by solving the following set covering problem [7]:

\[
\min \sum_{k=1}^{c} \tau_k \\
\text{subj to } \sum_{k=1}^{c} \alpha_{jk} \tau_k \geq 1 \text{ for } j = 1, \ldots, |T| \\
\tau_k \in \{0, 1\},
\]

where \( \tau_k = 1 \) if the \( k \)th element of \( A \) is included in the final subset \( A^* \) and 0 otherwise, whereas \( \alpha_{jk} = 1 \) if the \( j \)th pattern of \( T \) is covered by the \( k \)th binary string in \( A \) and 0 otherwise. With this definition, the constraint \( \sum_{k=1}^{c} \alpha_{jk} \tau_k \geq 1 \) ensures that every point in \( T \) is covered by at least one pattern of \( A^* \).

By acting on the objective function of problem (6) or on its constraints, the optimization of other quality measures, such as the degree of the bottom points associated with the elements of \( A^* \) or their average covering, can be pursued. For instance, the cost function can become \( \sum_{k=1}^{c} \gamma_k \tau_k \), being \( \gamma_k \) a positive coefficient that measures the quality of the \( k \)th pattern. The greater is \( \gamma_k \), the poorer is the corresponding \( k \)th bottom point according to the selected quality measure.

Since the set covering problem (6) is known to be NP-complete, approximated approaches are employed to find a good solution in a reasonable time. One of these methods is the greedy technique that subsequently adds patterns one by one until all the points in \( T \) are covered. At each addition, it selects the pattern that covers the highest number of points in \( T \) not yet covered by the binary strings already present in \( A^* \). This procedure is employed in all the tests presented in Section 7.

### 6 Computational Issues

In this section, a theoretical analysis of the computational cost of the algorithms LAD and LSC is performed. In particular, the asymptotic behavior of the execution time and memory occupation is derived by considering a worst-case approach. The characteristic variables of this analysis are: the length \( n \) of the binary strings, the size \( m \) of the portion \( T \cup F \) of the truth table at hand \( (m = |T \cup F|) \), and the maximum degree \( d \) of terms retrieved in the exhaustive part of the algorithm.

Since the computational cost of LAD and LSC is essentially due to the comparison between binary strings, a unitary amount of execution time is conventionally assigned to the comparison between two bits.

#### 6.1 Execution Time

Consider the general procedure in Fig. 2; it can be observed that the time required to extract the pattern \( z \) (Steps 2-4) and to remove from \( S \) the elements \( z \) covered by the current prime implicant \( a \) (Step 7) can be neglected with respect to the execution time of Steps 1 and 5. Consequently, the computational cost of LAD (resp., LSC) is essentially given by the two procedures, Exhaustive (resp., Depth) and Heuristic.

##### 6.1.1 LAD

The worst-case analysis for the exhaustive part of LAD leads to the following result:
Theorem 1. If \( 1 \leq d < n/2 \), the execution time of the procedure \textit{Exhaustive} in LAD is at most \( O(n^{2d} + mn^{d+1}) \).

\textbf{Proof.} The computationally relevant part of the algorithm in Fig. 3 consists in the repeated execution of Steps 2bba and 2bbc. The former is executed only if \( l \geq 2 \); it considers a new candidate bottom point \( p(P(a) \cup \{i\}) \) and checks if the set \( B \) includes all the \( l-1 \) binary strings (different from \( a \)) obtained by \( p(P(a) \cup \{i\}) \) after changing in \( a \) a bit from 1 to 0. Since \( B \) contains only terms of degree \( l-1 \), its cardinality \( |B| \) is at most \( \binom{n}{l-1} \), and consequently, the execution of Step 2bba requires \( n(l-1)\binom{n}{l-1} \) elementary operations.

On the other hand, Step 2bbc checks all the patterns in \( T \) and \( F \); in the worst case, such analysis needs \( n(|T| + |F|) = mn \) comparisons. Now note that both Steps 2bba and 2bbc are repeated \( n-k \) times (Step 2bb), where \( k \) is the highest integer in \( P(a) \), for each possible element \( a \in B \) (Step 2b).

If \( l = 1 \), only the value \( k = 0 \) is considered, which leads to \( n \) executions of the Step 2bbc. If \( l \geq 2 \), the maximum number of strings in \( B \) characterized by a certain \( k \) is equal to the number \( \binom{n}{l-2} \) of possible choices of \( l-2 \) indices from a total set of cardinality \( k-1 \). Hence, an upper bound for the execution time \( t_1 \) of the procedure \textit{Exhaustive} can be written as

\[
t_1 \leq mn^2 + \sum_{l=2}^{d} \sum_{k=2}^{n-1} \binom{k-1}{l-2} (n-k) \cdot \binom{n}{l-1} + mn,
\]

where the first summation on the right-hand side takes into account that the algorithm is repeated for each value of \( l \) not greater than the fixed maximum \( d \). The case \( l = 1 \) is not included in the summation since it gives rise to the term \( mn^2 \).

By employing the identities

\[
k\binom{k-1}{h-1} = h\binom{k}{h}, \quad \sum_{k=h}^{n} \binom{k}{h} = \binom{n+1}{h+1},
\]

which hold for every integers \( h, k, n \), we obtain

\[
\sum_{k=2}^{n-1} \binom{k-1}{l-2} (n-k) = n \sum_{l=2}^{d} \binom{l-1}{2} - \sum_{k=2}^{n-2} (k+1) \binom{k}{l-2} = n \sum_{l=2}^{d} \left( \binom{n-l-1}{l-1} - \binom{n-l-1}{l} \right) = \binom{n}{l} - \binom{n}{l-1} = \binom{n}{l}.
\]

This result can be substituted into (7), thus, obtaining

\[
t_1 \leq \sum_{l=2}^{d} n \binom{n}{l} (l-1) \binom{n}{l-1} + \sum_{l=1}^{d} mn \binom{n}{l} = n^2 \sum_{l=2}^{d} \binom{n}{l} + mn \sum_{l=1}^{d} \binom{n}{l},
\]

where the first identity of (8) has been used again.

Now an upper bound for the first term on the right-hand side can be found by observing that

\[
\binom{n}{l} = \frac{(l-1)(n-l+1)}{l} \binom{n}{l-1}^2 \leq \binom{n}{l-1}^2.
\]

In addition, since \( d < n/2 \), we have for every \( l \leq d \)

\[
\binom{n}{l} \leq \binom{n}{d} \leq \frac{n^d}{d!}.
\]

from which it follows for \( d \geq 2 \)

\[
t_1 \leq n^2 \sum_{l=2}^{d} \binom{n}{l} + mn \sum_{l=1}^{d} \binom{n}{l} \leq n^2 \binom{d-1}{d-1}^2 + mn \frac{n^d}{d!} \leq n^{2d} + mn^{d+1}.
\]

This inequality for \( t_1 \) is also valid for \( d = 1 \), since in this case, we have from (7) \( t_1 \leq mn^2 \).

A simpler analysis allows us to retrieve an upper bound for the computational cost of the heuristic part of LAD.

Theorem 2. The execution time of the heuristic part of LAD is at most \( O(m^2n^2) \).

\textbf{Proof.} The heaviest task of the algorithm in Fig. 4 concerns the analysis of the binary strings in \( F \), performed at Step 2b. Since this check is repeated for each \( i \in I \) (Step 2), a single execution of \textit{Heuristic} requires at most \( n|I||F| \leq mn^2 \) elementary operations.

Now, since the procedure \textit{Heuristic} is called for each pattern \( x \in T \) not yet covered by the implicants in \( A \), the heuristic part of LAD needs a time

\[
t_2 \leq mn^2 |T| \leq m^2n^2.
\]

From Theorems 1 and 2, the behavior for the computational cost of LAD is readily derived.

\textbf{Corollary 1.} If \( 1 \leq d < n/2 \), the execution time needed for LAD is at most \( O(n^{2d} + mn^{d+1} + m^2n^2) \).

Hence, the computational burden of LAD is in the worst case a rapidly increasing function of \( n \), which can lead to an excessive execution time even for moderate values of \( n \) and \( d \).

6.1.2 LSC

Before retrieving similar upper bounds for the computational cost of LSC, let us prove a basic result about the completeness of its exhaustive part.

Theorem 3. The exhaustive part of LSC analyzes at most once all the bottom points for the pair \((T,F)\) with degree \( l \leq d \).

\textbf{Proof.} In the exhaustive part of LSC, bottom points are incrementally built by recursively calling the procedure \textit{Depth} shown in Fig. 5. Initially, the candidate implicant \( s(I_0) \) including only \( 0 \)'s is considered and each subsequent call to the procedure \textit{Depth} changes a bit from 0 to 1 by removing an index in the set \( J \) (Step 2a). Hence, generating a string with \( l \) 1's needs \( l \) recursive calls to \textit{Depth}.

Furthermore, it can be easily seen that every call to \textit{Depth} with actual parameters \( I \) and \( J \) analyzes only
points belonging to the upper shadow of $s(I \cup J)$. Thus, if an implicant $s(I \cup J)$ is added to the antichain $A$, no further calls to \texttt{Depth} are performed since they could not reach other bottom points for the pair $(T, F)$.

In every case, if the binary string $s(I \cup J)$ is analyzed at Step 2b of Fig. 5, we can directly find the exact sequence of calls that produced it. It is sufficient to observe that any call to \texttt{Depth} acts only on the first run of 0s at the left (whose indices are included into the set $I$), leaving unchanged the rest of the string. In particular, Step 2a selects one at a time the bits of this run, starting from the right; in this way, the run of 0s at the left is preserved.

As an example, suppose that the string $00010110$ is analyzed at Step 2b of Fig. 5. It can be readily found that the current call of \texttt{Depth} is the third one in the recursion order (equal to the number of 1s in the string). At the present step, we have $I = \{1, 2, 3\}$ (first run of 0s) and $J = \{5, 8\}$ (remaining 0s); we can also infer the content of $I$ and $J$ for the caller and for the first instance of \texttt{Depth}.

They are $I = \{1, 2, 3, 4, 5\}, J = \{8\}$ and $I = \{1, 2, 3, 4, 5, 6\}, J = \{8\}$, respectively.

Since every call of \texttt{Depth} examines only the bits of the first run of 0s, it is not possible to obtain twice the same binary string, and consequently, every bottom point is considered only once. On the other hand, by construction, all the bits of the run are analyzed through the loop at Step 2 of Fig. 5. Hence, all the bottom points with degree $l \leq d$ are retrieved.

Theorem 3 allows one to readily obtain the computational cost of the exhaustive part of LSC.

**Corollary 2.** If $1 \leq d < n/2$, the execution time needed for the exhaustive part of LSC is at most $O(n^{3d+1} + mn^{d+1})$.

**Proof.** Since the final antichain $A$ can include at most $\binom{n}{d}$ bottom points, the execution of Steps 2b and 2ba requires no more than $mn + n\binom{n}{d}$ elementary operations. Hence, since the number of binary strings examined is at most $\sum_{i=1}^{\binom{n}{d}}$, the total time $t_3$ needed by the recursive execution of \texttt{Depth} verifies the inequality

$$t_3 \leq n \sum_{l=1}^{d} \binom{n}{l} \left( m + \binom{n}{d} \right)$$

$$= nd\binom{n}{d} + mn\binom{n}{d} \leq n^{2d+1} + mn^{d+1},$$

having used again the inequalities (9).

Finally, the computational cost of the heuristic part of LSC is provided by the following result:

**Theorem 4.** The execution time needed for the heuristic part of LSC is at most $O(m^2 n^2)$.

**Proof.** The algorithm in Fig. 6 subsequently removes bits from the set $I$, until $I$ is empty. The computationally relevant part of the procedure consists in Step 3a, which determines the lower distances between the current implicant $p(I \cup J)$ and some patterns of $F$. This calculation needs at most $n|F|$ elementary operations and is repeated for each $i \in I$, with $I \subseteq I_i$. On the other hand, Step 2 requires $n|T|$ calculations to retrieve the sets $|S^o_i|$ and $|T^o_i|$.

Since the procedure in Fig. 6 is repeated for each pattern belonging to $T$, the execution time $t_4$ of the heuristic part of LSC is at most

$$t_4 \leq |T|(n|T| + n^2|F|) \leq m^2 n + m^2 n^2 \leq 2m^2 n^2,$$

which proves the theorem.

The global behavior of LSC directly follows from Theorems 3 and 4.

**Corollary 3.** If $1 \leq d < n/2$, the computational cost of LSC is at most $O(n^{2d+1} + mn^{d+1} + m^2 n^2)$.

Note that, according to Corollaries 1 and 3, in the worst case, the computational burden of LAD is slightly lower than that of LSC. However, it should be noted that the execution time heavily depends on the function to be reconstructed, and in particular, on the possible bottom points for the pair $(T, F)$. In fact, the number of elements in the antichain $A$ to be checked at Step 2b of the procedure \texttt{Depth} cannot be established a priori and significantly affects the performance of LSC. This is clearly evident in the simulations, where LSC shows a lower computational burden with respect to LAD, even if the former adopts a depth-first approach.

### 6.2 Memory Occupation

When calculating the amount of memory needed for LAD and LSC, the storage required for $T$ and $F$ must be taken into account. It is $O(mn)$ and does not depend on the specific procedure adopted for synthesizing the positive Boolean function. Then, since the number of bottom points for the pair $(T, F)$ is supposed to be low (or at most comparable) with respect to $m$, the memory for the antichain $A$ is neglected. Likewise, we do not consider the storage needed for the index sets $I$ and $J$ or for the distances $d_i(p(I \cup J), F^o_i)$ used in the heuristic part of LSC.

#### 6.2.1 LAD

Concerning the exhaustive part of LAD, memory has to be reserved for the sets $B$ and $C$, which contain promising binary strings of degree $l - 1$ and $l$, respectively. On the other hand, nothing computationally relevant is stored during the heuristic part of LAD. This leads to the following result:

**Theorem 5.** The amount of memory required for LAD is at most $O(mn + n^d)$.

**Proof.** Apart from the contribution $O(mn)$ due to the storage of the sets $T$ and $F$, we must consider for $l < d$ the memory needed for the auxiliary sets $B$ and $C$, which include at most

$$n\left(\binom{n}{l-1} + \binom{n}{l}\right) \leq 2n\binom{n}{d-1} \leq 2n^d,$$

bits.

#### 6.2.2 LSC

Since LSC does not require any significant additional storage other than that needed for the sets $T$ and $F$, we can readily obtain the following behavior:

**Theorem 6.** The memory required by LSC is at most $O(mn)$.

As expected, the computational demand of LAD, in terms of memory occupation, is considerably higher than that of LSC, since the former adopts a breadth-first approach, which generally sacrifices storage for speed. However, the results of this section show that the depth-first procedure adopted
by LSC dramatically reduces the memory demand while maintaining almost unchanged the execution time. Simulations reported in the next section point out this fact.

7 Tests and Results

To measure the performance achieved by LSC in reconstructing a pdpBf, two different experiments have been carried out. The first one concerns the ability of retrieving the optimal expression of a positive Boolean function \( f \), whose PDNF is randomly generated. In particular, we want to measure the difference between the considered algorithms LAD and LSC, in terms of complexity of the generated solution and computational cost required to produce the irredundant PDNF.

The second experiment evaluates the possibility of using LSC in the synthesis of general Boolean functions. In fact, every mapping \( g: \{0,1\}^n \rightarrow \{0,1\} \) can be realized through a positive Boolean function \( f: \{0,1\}^n \rightarrow \{0,1\} \) by applying the transformation \( \lambda: \{0,1\}^n \rightarrow \{0,1\}^n \) defined as

\[
\lambda_{i-1}(x) = 1 - x_i, \quad \lambda_i(x) = x_i,
\]

for \( i = 1, \ldots, n \), where \( \lambda_i(x) \) is the \( i \)th bit of the binary string \( \lambda(x) \) having length \( 2n \). Once \( f \) has been generated, the associated Boolean function \( g \) is simply obtained by setting \( g(x) = f(\lambda(x)) \) for every \( x \in \{0,1\}^n \), where \( \lambda \) is given by (10). In this way, every prime implicant for \( f \) can be directly transformed into a minterm for \( g \).

Thus, LSC can be used for reconstructing a Boolean function \( g \) from two sets of binary strings \( T' \) and \( F' \), with \( T' \cap F' = \emptyset \), where we want to have \( g(x) = 1 \) for every \( x \in T' \) and \( g(x) = 0 \) for every \( x \in F' \). The first action to be performed is translating \( T' \) and \( F' \), which include binary strings with length \( n \), into two sets \( T \) and \( F \), containing elements with \( 2n \) bits, by applying the mapping \( \lambda \). It can be shown that the resulting set \( T \) is lower separated from \( F \).

Target Boolean functions are generated by choosing, at random, the minimum true points to be included into the antichain \( A \) of the irredundant PDNF (2). By adopting proper constraints, the complexity of the resulting expression is controlled so as to avoid too simple or too specialized functions. This kind of functions allows a fair evaluation of the quality of the results produced by LSC and LAD. In addition, they lead to reconstruction problems similar to those arising from real-world applications, particularly in the machine learning field.

The performance of LSC in reconstructing general Boolean functions is compared with those of the standard LAD algorithm, as described in [7]. All the experiments have been carried out on a personal computer with a Pentium 4 (CPU 2.80 GHz, RAM 480 MB) running under the Windows XP operating system.

7.1 pdpBf Reconstruction

To analyze the performance of LAD and LSC in reconstructing a pdpBf, several trials have been performed for different values of the input dimension \( n \). In particular, to obtain a reliable value for the measured parameters, i.e., complexity and execution time, 50 different positive Boolean functions have been randomly constructed for \( n \) assuming integer values in the set \( \{5, 10, 20, 50, 100, 200, 500, 1000\} \).

The truth table for each function was generated by building, at random, \( \min(\{n/2\}, 10) \) logical products, which form the PDNF of the target positive Boolean function \( f \). Each logical product contains a number \( \mu \) of terms randomly selected in the range \([3, \min(n, 10)]\) so as to refuse too specialized functions including AND operations (with \( \mu > 10 \)) that assume value 1 only for a very small set of input strings. Note that the presence of logical products with one or two terms would lead to too simple functions, easily recognized by every trivial method; therefore, we have imposed the constraint \( \mu \geq 3 \). The indices of the terms included in each logical product are randomly chosen with uniform probability.

A subset of \( r = \min(2^{n-1}, 1024) \) input-output pairs is then randomly extracted without replacement to form the sets \( T \) and \( F \), which are employed in the reconstruction task.

Experiments have been carried out with the LAD and LSC algorithms, changing the value of the maximum degree \( d \) in their exhaustive part. For each method, the average complexity, i.e., the number of implicants in the PDNF, the classification accuracy, i.e., the percentage of errors in recognizing new samples, and the computational burden, i.e., execution time and memory occupation, have been measured. Some plots comparing the performance of the different methods are reported in Figs. 7 and 8.

The odd values 3, 5, and 7 have been considered for the maximum degree \( d \). The labels LAD-\( d \) (resp., LSC-\( d \)) in the legends refer to the application of algorithm LAD (resp., LSC) using the value \( d \) in its exhaustive part. The direct application of the SC procedure is also considered and denoted by LSC-0 in the legends.

![Fig. 7](image_url)
As expected, the plot in Fig. 7b shows that the accuracy of the PDNF increases when higher values of the maximum degree \(d\) are used. On the other hand, even for low values of \(d\), both LAD and LSC require an excessive computational cost when the number \(n\) of inputs increases, as shown in Fig. 8.

In particular, some trials are not reported since their execution exceeds the available time (about one day for each pdpBf reconstruction) or the physical memory of the computer. Table 1 shows the maximum dimension \(\bar{n}\) of the input space for which a function \(f\) can be reconstructed by a specific method.

When \(n > 200\), the application of LAD is not possible since its exhaustive part requires the memorization of a huge amount of data also for \(d = 3\). In contrast, the depth-first approach followed by LSC permits a more general applicability; in this case, the constraint on the execution time is the most relevant. As an example, the pdpBf reconstruction with LSC in the case \(n = 1000\) and \(d = 3\) requires few hours of CPU and less than 1 MB of memory.

Plots in Figs. 7 and 8 show that LSC with the choice \(d = 3\) in its exhaustive part is always advantageous, since it improves complexity and accuracy while keeping the computational cost unchanged with respect to the simple SC procedure (LSC-0).

However, the complexity of the solution (Fig. 7a) does not grow indefinitely, but starts decreasing when \(n > 200\), since the portion of the truth table included in the sets \(T\) and \(F\) becomes almost negligible (less than \(10^{-60}\) of the total). Consequently, the number of bottom points for the pair \((T, F)\) increases accordingly, thus making it more probable to find prime implicants with low degree.

Looking at Figs. 7a and 7b, it can be noticed that the LSC approach obtains better values of complexity and accuracy with respect to LAD. This is mainly due to the adoption of SC in the heuristic part, which employs a smart strategy for selecting the bits to be changed for retrieving a good bottom point covering the pattern at hand.

The Monotone Dualization (MD) algorithm [24] has also been tested on some problems with low input dimension \(n\). MD performs the reconstruction of a positive Boolean function \(f\) under the assumption that the value of \(f(x)\) is known for any \(x \in \{0, 1\}^n\).

In particular, MD groups the binary strings of \(\{0, 1\}^n\) to form totally ordered sequences \(C_i\), called Hansel chains. Each \(C_i\) is then analyzed to determine the first element \(x\) of the chain having corresponding output \(f(x) = 1\). Since all the subsequent points of \(C_i\) lie in the upper shadow of \(x\), their analysis is not needed. By examining in this way all the Hansel chains, the collection of minimum true points for \(f\) is readily obtained.

Note that the generation of the Hansel chains requires a memory occupation \(O(2^n)\), since all the binary strings of length \(n\) must be stored simultaneously. For this reason, the MD method cannot be employed for problems with high input dimension \((n > 30)\).

The performed tests show that the MD algorithm is able to retrieve the correct PDNF for a positive Boolean function \(f\) by examining only about 10 percent of the total \(n\)-dimensional Boolean strings. However, the execution time has the same order of magnitude as that required by LAD and LSC when \(d = n\), i.e., when the exhaustive part of the algorithms considers all the possible implicants. As an example, for \(n = 20\), MD retrieves the correct PDNF within 15 minutes, whereas LAD and LSC need 20 minutes to perform the same task.

### 7.2 pdBF Reconstruction

As in the previous case, the truth tables of 50 target Boolean functions have been randomly generated for each input dimension \(n\) in the set \(\{5, 10, 20, 50, 100, 200, 500, 1000\}\), by constructing the corresponding DNF expressions. Each DNF contains \(\min(n, 10)\) logical products, having a number \(\mu\) of operands chosen again, at random, in the interval \([3, \min(n, 10)]\).

In the generation of positive Boolean functions, the indices of the terms included in each logical product are randomly selected with uniform probability.

The quality of the reconstructed function, in terms of complexity and accuracy, is shown in Fig. 9, while the computational effort needed for its generation is reported in Fig. 10. Note that in this set of trials, the standard LAD
approach (described in [7]) has been employed, since it is expressly devoted to pdBf synthesis.

Notwithstanding this choice, the results reported in Figs. 9 and 10 reflect the same behavior pointed out for pdpBf reconstruction. In particular, the quality of functions built by LSC is significantly better than that exhibited by LAD, both in terms of complexity and accuracy. This superiority is not achieved at the expense of an increase in computational burden. In contrast, the execution time and the memory occupation of LSC are again lower than those of LAD if the values of $n$ and $d$ are equal.

It follows that the introduction of the $\lambda$ mapping (10) does not make the performance of the LSC worse, either in terms of quality of the Boolean function or in terms of computational efficiency. In particular, due to its excessive memory occupation, again LAD is not able to treat problems with $n > 200$, whereas LSC achieves a good accuracy even when $n = 1000$. The execution time for LSC ranges from few seconds ($n < 100$) to about one day ($n = 1000$).

8 CONCLUSIONS
A novel algorithm, called LSC, for the reconstruction of a pdpBf has been introduced. It combines two existing techniques, LAD and SC, to produce the irredundant PDNF of the resulting function. In particular, LAD adopts an exhaustive breadth-first approach for examining all the

prime implicants with degree not greater than a maximum $d$, whereas SC employs a smart heuristic to determine a good minterm (according to some measure of quality) covering a given pattern $x$.

As with LAD, the LSC technique is composed by an exhaustive part, which follows a depth-first approach to save memory, and by a heuristic part that adopts SC to complete the irredundant PDNF of the desired positive Boolean function. Whereas [7] proves the correctness of the approach followed by LAD, the theoretical analysis of Section 6 (Theorem 3) shows that LSC is able to solve any consistent pdpBf reconstruction problem, since the correctness of SC can be directly established.

In addition, LSC requires only the memory needed for storing the portion of the truth table at hand. In contrast, LAD must also maintain (in the worst case) all the binary strings with degree $l$ for $l < d$, which leads to an increase $O(n^d)$ in the memory occupation.

A comparison between the performance of LSC and LAD, in terms of complexity and accuracy of the retrieved solution, has been carried out by performing extensive tests on randomly generated Boolean functions. As expected from theory, the reduced memory request allows LSC to treat problems with higher input dimension $n$. However, due to an efficient housekeeping method, even the execution time of LSC is significantly lower than that of LAD.

This decrease in computational cost is not achieved at the expense of a worse final solution. In contrast, positive Boolean functions generated by LSC exhibit, in general, a...
lower complexity and a higher accuracy than those built with LAD. This improvement is essentially due to the employment of SC in the heuristic part of the procedure and degrades when \( d \) increases. Note that in most applications, the high input dimension \( n \) forces the choice of low values for \( d \); in these cases, the adoption of SC leads to resulting functions with better quality. However, the adoption of an exhaustive part improves the accuracy of the solution, as pointed out by the comparison between LSC and SC (denoted by LSC-0 in the plots of Section 7).

The behavior of LSC is advantageous also when dealing with pdBF reconstruction; as a matter of fact, experimental trials have shown that a simple mapping allows LSC to treat successfully classic Boolean function synthesis. The comparison with standard LAD, as described in [7], points out the superiority of LSC in terms of both complexity and accuracy.

In the presence of noisy data, the portion of the truth table to be employed for pdBF reconstruction may be no more consistent, i.e., no positive Boolean function satisfies all the given input-output relations. In this case, the LSC algorithm, as described in this paper, cannot be directly applied.

However, it is still possible to retrieve the irredundant PDNF of a function that behaves well for most input patterns. It is sufficient to relax the control statements at Step 2ba of Fig. 5 (exhaustive part) and Step 3b of Fig. 6 (heuristic part) so as to accept that the resulting positive Boolean function provides a wrong output in correspondence with a small amount of input points. The modification of LSC for permitting the treatment of noisy and missing data will be the subject of a forthcoming paper.

Currently, LSC is adopted for the generation of Switching Neural Networks (SNNs) [12] solving the real-world classification problems. In most applications, the devices produced by LSC show a better accuracy with respect to SNN trained by SC. The possibility of extracting an exhaustive part improves the accuracy of the solution, as pointed out by the comparison between LSC and SC (denoted by LSC-0 in the plots of Section 7).

REFERENCES


Marco Muselli received the MSc degree in electronic engineering from the University of Genoa, Italy, in 1985. He joined the Institute of Electronic Circuits in 1988, now part of the Institute of Electronics, Computer and Telecommunication Engineering, Italian National Research Council, where he is currently a senior researcher. His research interests include machine learning, bioinformatics, neural networks, global optimization, mathematical statistics, and probability theory. His research activity mainly focuses on the development of novel learning algorithms and their applications in several fields, especially in the solution of biomedical problems. He is the author or coauthor of more than 100 scientific publications, most of which on international journals. He is a member of the IEEE.

Enrico Ferrari received the MSc and PhD degrees in physics from the University of Genoa, Italy, in 2003 and 2009, respectively. He is currently a postdoctoral fellow in the Institute of Electronics, Computer and Telecommunication Engineering, Italian National Research Council. His research interests are in the field of probability theory, statistics, and machine learning. His research activity focuses on the development of novel learning algorithms and their application in several real-world problems, especially in the analysis of biomedical data.