Pattern Recognition 43 (2010) 557–568

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.de/locate/pr

U-curve: A branch-and-bound optimization algorithm for U-shaped cost functions on Boolean lattices applied to the feature selection problem ☆

Marcelo Ris [a,*], Junior Barrera [b,*], David C. Martins Jr. [a,**]

[a] Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo-SP, Brazil
[b] Faculdade de Filosofia Ciências e Letras de Ribeirão Preto, Universidade de São Paulo, Ribeirão Preto-SP, Brazil

ARTICLE INFO

Article history:
Received 30 October 2008
Received in revised form
7 July 2009
Accepted 16 August 2009

Keywords:
Boolean lattice
Branch-and-bound algorithm
U-shaped curve
Feature selection
Subset search
Optimal search

ABSTRACT

This paper presents the formulation of a combinatorial optimization problem with the following characteristics: (i) the search space is the power set of a finite set structured as a Boolean lattice; (ii) the cost function forms a U-shaped curve when applied to any lattice chain. This formulation applies for feature selection in the context of pattern recognition. The known approaches for this problem are branch-and-bound algorithms and heuristics that explore partially the search space. Branch-and-bound algorithms are equivalent to the full search, while heuristics are not. This paper presents a branch-and-bound algorithm that differs from the others known by exploring the lattice structure and the U-shaped chain curves of the search space. The main contribution of this paper is the architecture of this algorithm that is based on the representation and exploration of the search space by new lattice properties proven here. Several experiments, with well known public data, indicate the superiority of the proposed method to the sequential floating forward selection (SFFS), which is a popular heuristic that gives good results in very short computational time. In all experiments, the proposed method got better or equal results in similar or even smaller computational time.

1. Introduction

A combinatorial optimization algorithm chooses the object of minimum cost over a finite collection of objects, called search space, according to a given cost function. The simplest architecture for this algorithm, called full search, access each object of the search space, but it does not work for huge spaces. In this case, what is possible is to access some objects and choose the one of minimum cost, based on the observed measures. Heuristics and branch-and-bound are two families of algorithms of this kind. A heuristic algorithm does not have formal guaranty of finding the minimum cost object, while a branch-and-bound algorithm has mathematical properties that guarantee to find it.

Here, it is studied a combinatorial optimization problem such that the search space is composed of all subsets of a finite set with n points (i.e., a search space with 2^n objects), organized as a Boolean lattice, and the cost function has a U-shape in any chain of the search space or, equivalently, the cost function has a U-shape in any maximal chain of the search space.

This structure is found in some applied problems such as feature selection in pattern recognition Duda et al. [5], Jain et al. [7] and W-operator window design in mathematical morphology [8]. In these problems, a minimum subset of features, that is sufficient to represent the objects, should be chosen from a set of n features. In W-operator design, the features are points of a finite rectangle of Z^2 called window. The U-shaped functions are formed by error estimation of the classifiers or of the operators designed or by some measures, as the entropy, on the corresponding estimated join distribution. This is a well known phenomenon in pattern recognition: for a fixed amount of training data, the increasing number of features considered in the classifier design induces the reduction of the classifier error by increasing the separation between classes until the available data become too small to cover the classifier domain and the consequent increase of the estimation error induces the increase of the classifier error. Some known approaches for this problem are heuristics. A relatively well succeeded heuristic algorithm is the sequential floating forward selection (SFFS) [11], which gives good results in relatively small computational time.

There is a myriad of branch-and-bound algorithms in the literature that are based on monotonicity of the cost-function [6,10,14,15]. For a detailed review of branch-and-bound algorithms, refer to Somol and Pudil [13]. If the real distribution of the joint probability between the patterns and their classes were known, larger dimensionality would imply in smaller

☆ Funded by: BZG.
* Corresponding authors.
** Principal corresponding author.
E-mail addresses: mris@ime.usp.br (M. Ris), jb@ime.usp.br (J. Barrera), davidjr@ime.usp.br (D.C. Martins Jr.).

0031-3203/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2009.08.018

classification errors. However, in practice, these distributions are unknown and should be estimated. A problem with the adoption of monotonic cost-functions is that they do not take into account the estimation errors committed when many features are considered (“curse of dimensionality” also known as “U-curve problem” or “peaking phenomena” [7]).

This paper presents a branch-and-bound algorithm that differs from the others known by exploring the lattice structure and the U-shaped chain curves of the search space.

Some experiments were performed to compare the SFFS to the U-curve approach. Results obtained from applications such as W-operator window design, genetic network architecture identification and eight UCI repository data sets show encouraging results, since the U-curve algorithm beats (i.e., finds a node with smaller cost than the one found by SFFS) the SFFS results in smaller computational time for 27 out of 38 data sets tested. For all data sets, the U-curve algorithm gives a result equal or better than SFFS, since the first covers the complete search space.

Though the results obtained with the application of the method developed to pattern recognition problems are exciting, the great contribution of this paper is the discovery of some lattice algebra properties that lead to a new data structure for the search space representation, that is particularly adequate for updates after up-down lattice interval cuts (i.e., cuts by couples of intervals [0,X] and [X,W]). Classical tree based search space representations do not have this property. For example, if the Depth First Search were adopted to represent the Boolean lattice only cuts in one direction could be performed.

Following this Introduction, Section 2 presents the formalization of the problem studied. Section 3 describes structurally the branch-and-bound algorithm designed. Section 4 presents the mathematical properties that support the algorithm steps. Section 5 presents some experimental results comparing U-curve to SFFS. Finally, Conclusion discusses the contributions of this paper and proposes some next steps of this research.

2. The Boolean U-curve optimization problem

Let W be a finite subset, $\mathcal{P}(W)$ be the collection of all subsets of W, $\subseteq$ be the usual inclusion relation on sets and, $|W|$ denote the cardinality of W. The search space is composed by $2^{|W|}$ objects organized in a Boolean lattice.

The partially ordered set $(\mathcal{P}(W), \subseteq)$ is a complete Boolean lattice of degree $|W|$ such that: the smallest and largest elements are, respectively, $\emptyset$ and W; the sum and product are, respectively, the usual union and intersection on sets and the complement of a set X in $\mathcal{P}(W)$ is its complement in relation to W, denoted by X^c.

Subsets of W will be represented by strings of zeros and ones, with 0 meaning that the point does not belong to the subset and 1 meaning that it does. For example, if $W=\{(-1,0),(0,0),(+1,0)\}$, the subset $\{(-1,0),(0,0)\}$ will be represented by 110. In an abuse of language, $X=110$ means that X is the set represented by 110.

A chain $\mathcal{A}$ is a collection $\{A_1,A_2,\ldots,A_k\}\subseteq\mathcal{X}\subseteq\mathcal{P}(W)$ such that $A_1\subseteq A_2\subseteq\cdots\subseteq A_k$. A chain $\mathcal{M}\subseteq\mathcal{X}$ is maximal in $\mathcal{X}$ if there is no other chain $\mathcal{C}\subseteq\mathcal{X}$ such that $\mathcal{C}$ contains properly $\mathcal{M}$.

Let c be a cost function defined from $\mathcal{P}(W)$ to $\mathbb{R}$. We say that c is decomposable in U-shaped curves if, for every maximal chain $\mathcal{M}\subseteq\mathcal{P}(W)$, the restriction of c to $\mathcal{M}$ is a U-shaped curve, i.e., for every $A,X,B\in\mathcal{M}$, $A\subseteq X\subseteq B\Rightarrow\max(c(A),c(B))\geq c(X)$.

Fig. 1 shows a complete Boolean lattice $\mathcal{L}$ of degree 4 with a cost function c decomposable in U-shaped curves. In this figure, it is emphasized a maximal chain in $\mathcal{L}$ and its cost function. Fig. 2 presents the curve of the same cost function restricted to some maximal chains in $\mathcal{L}$ and in $\mathcal{X}\subseteq\mathcal{L}$. Note the U-shape of the curves in Fig. 2.

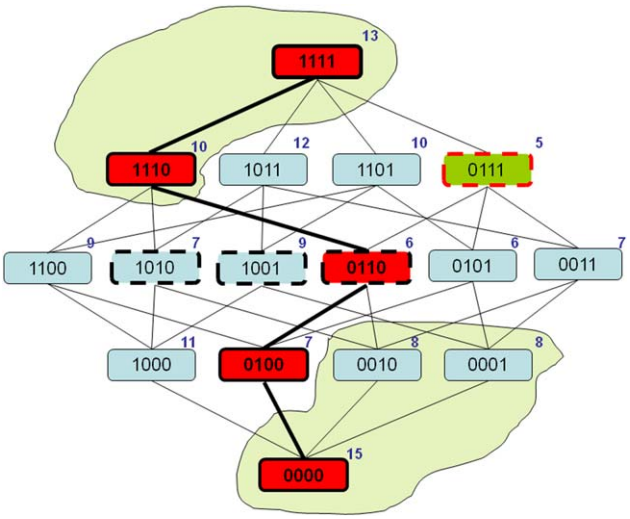

Fig. 1. A complete Boolean lattice $\mathcal{L}$ of degree 4 and the cost function decomposable in U-shaped curves. $\mathcal{X}=\mathcal{L}-\{0000,0010,0001,1110,1111\}$ is a poset obtained from $\mathcal{L}$. A maximal chain in $\mathcal{L}$ is emphasized. The element 0111 is the global minimum element and 0101 is the local minimum element in the maximal chain.

Our problem is to find the element (or elements) of minimum cost in a Boolean lattice of degree $|W|$. The full search in this space is an exponential problem, since this space is composed by $2^{|W|}$ elements. Thus, for moderately large $|W|$, the full search becomes unfeasible.

3. The U-curve algorithm

The U-shaped format of the restriction of the cost function to any maximal chain is the key to develop a branch-and-bound algorithm, the *U-curve algorithm*, to deal with the hard combinatorial problem of finding subsets of minimum cost.

Let A and B be elements of the Boolean lattice $\mathcal{L}$. An *interval* $[A,B]$ of $\mathcal{L}$ is the subset of $\mathcal{L}$ given by $[A,B]=\{X\in\mathcal{L}:A\subseteq X\subseteq B\}$. The elements A and B are called, respectively, the left and right extremities of $[A,B]$. Intervals are very important for characterizing decompositions in Boolean lattices [2,4].

Let R be an element of $\mathcal{L}$. In this paper, intervals of the type $[\emptyset,R]$ and $[R,W]$ are called, respectively, lower and upper *intervals*. The right extremity of a lower interval and the left extremity of an upper interval are called, respectively, lower and upper *restrictions*. Let $\mathcal{R}_L$ and $\mathcal{R}_U$ denote, respectively, collections of lower and upper intervals. The search space will be the *poset* $\mathcal{X}(\mathcal{R}_L,\mathcal{R}_U)$ obtained by eliminating the collections of lower and upper restrictions from $\mathcal{L}$, i.e., $\mathcal{X}(\mathcal{R}_L,\mathcal{R}_U)=\mathcal{L}-\cup\{[\emptyset,R]:R\in\mathcal{R}_L\}-\cup\{[R,W]:R\in\mathcal{R}_U\}$. In cases in which only the lower or the upper intervals are eliminated, the resulting search space is denoted, respectively, by $\mathcal{X}(\mathcal{R}_L)$ and $\mathcal{X}(\mathcal{R}_U)$ and given, respectively, by $\mathcal{X}(\mathcal{R}_L)=\mathcal{L}-\cup\{[\emptyset,R]:R\in\mathcal{R}_L\}$ and $\mathcal{X}(\mathcal{R}_U)=\mathcal{L}-\cup\{[R,W]:R\in\mathcal{R}_U\}$.

The search space is explored by an iterative algorithm that, at each iteration, explores a small subset of $\mathcal{X}(\mathcal{R}_L,\mathcal{R}_U)$, computes a local minimum, updates the list of minimum elements found and extends both restriction sets, eliminating the region just explored. The algorithm is initiated with three empty lists: minimum elements, lower and upper restrictions. It is executed until the whole space is explored, i.e., until $\mathcal{X}(\mathcal{R}_L,\mathcal{R}_U)$ becomes empty. The subset of $\mathcal{X}(\mathcal{R}_L,\mathcal{R}_U)$ eliminated at each iteration is defined from the exploration of a chain, which may be done in down-up or up-down direction. Algorithm 1 describes this process. The direction selection procedure (line 5) can use a random or an adaptative

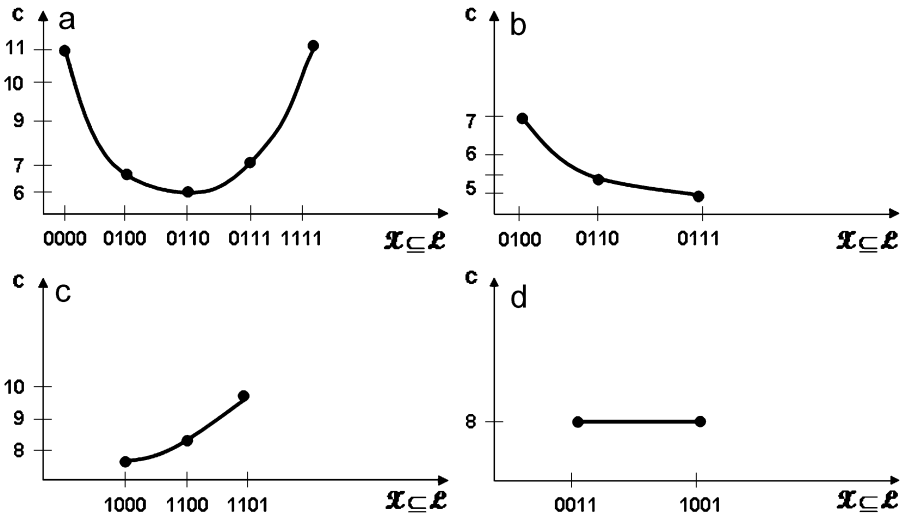

Fig. 2. The four possible representation of the cost function c restricted to some maximal chains in $\mathcal{L}$ (a) and in $\mathcal{X} \subseteq \mathcal{L}$ (b)–(d) of Fig. 1.

method. The random method states a static probability to select the down-up or up-down direction. The adaptative method calculates a new probability to each direction giving more probability to down-up direction if most of the local minima is closest to the bottom of the lattice and up-down otherwise.

Algorithm 1. U-curve-algorithm()

1: $\mathcal{M} \Leftarrow \emptyset$
2: $\mathcal{R}_L \Leftarrow \emptyset$
3: $\mathcal{R}_U \Leftarrow \emptyset$
4: **while** $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U) \neq \emptyset$ **do**
5: *direction* $\Leftarrow$ Select-Direction()
6: **if** *direction* is *UP* **then**
7: Down-Up-Direction($\mathcal{R}_L, \mathcal{R}_U$)
8: **else**
9: Up-Down-Direction($\mathcal{R}_L, \mathcal{R}_U$)
10: **end if**
11: **end while**

An element C of the poset $\mathcal{X} \subseteq \mathcal{L}$ is called a *minimal* element of $\mathcal{X}$, if there is no other element C' of $\mathcal{X}$ with $C' \subset C$. In Fig. 1, the minimal elements of $\mathcal{X}(\mathcal{R}_L)$ are: 1000, 0100 and 0011. If the down-up direction is chosen, the Down-Up-Direction procedure is performed (Algorithm 2):

- Minimal-Element procedure calculates a *minimal* element B of the poset $\mathcal{X}(\mathcal{R}_L)$. Only the lower restriction set is used to calculate the minimal element B. An element B is said to be *covered by* the lower restriction set $\mathcal{R}_L$, if $\exists R \in \mathcal{R}_L : B \subseteq R$, and B is said to be covered by the upper restriction set $\mathcal{R}_U$, if $\exists R \in \mathcal{R}_U : R \subseteq B$. When the calculated B is covered by an upper restriction, it is discarded, i.e., the lower restriction set is updated with B and a new iteration begins (lines 1–5).
- The down-up direction chain exploration procedure begins with a minimal element B and flows by random selection of upper adjacent elements from the current poset $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ until it finds the *U-curve condition*, i.e., the last element selected (B) has cost bigger than the previous one (M) (lines 7–11).
- At this point, the element M is the minimum element of the chain explored, A and B are, respectively, the lower and upper adjacent elements of M (i.e., $A \subset M \subset B$ and, by construction, $c(A) \leq c(M) \leq C(B)$). It can be proved that any element C of $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$, with $C \subset A$, has cost bigger than A and, any element D of $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$, with $B \subset D$, has cost bigger than B. By using this property, the lower and upper restrictions can be updated, respectively, by A and B (lines 12–17). Fig. 3 shows a schematic representation of the first iteration of the algorithm and the elements contained in the intervals $[\emptyset, A = 1 \ldots 1010 \ldots 0]$ and $[B = 1 \ldots 11110 \ldots 0, W]$. The result list can be updated with M (line 18), i.e., M will be included in the result list if it has cost lower than the best m elements found in the search till the current step.
- In order to prevent visiting the element M more than once, a recursive procedure called *minimum exhausting* procedure is performed (line 19).

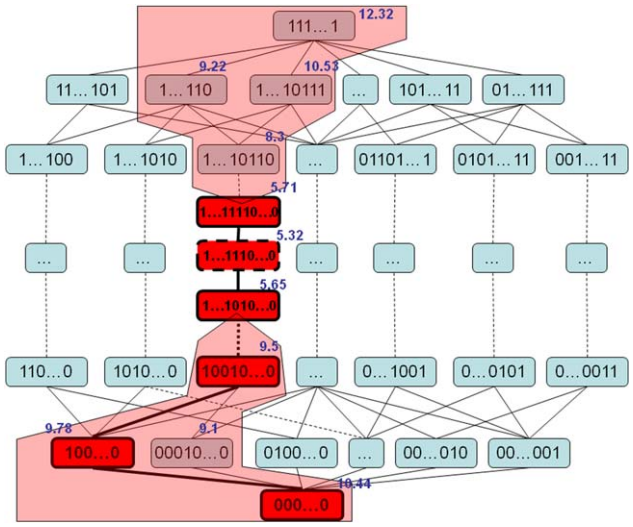

Fig. 3. A schematic representation of a step of the algorithm, the detached areas represent the elements contained in a lower and upper restrictions.

Algorithm 2. Down-Up-Direction(ElementSet $\mathcal{R}_L$, ElementSet $\mathcal{R}_U$)

1: $B \Leftarrow$ Minimal − Element($\mathcal{R}_L$)
2: **if** B is covered by $\mathcal{R}_U$ **then**
3: Update-Lower-Restriction($B, \mathcal{R}_L$)
4: **return**
5: **end if**
6: $M \Leftarrow$ **null**

7: **repeat**
8: $A \Leftarrow M$
9: $M \Leftarrow B$
10: $B \Leftarrow$ Select − Upper − Adjacent$(M, \mathcal{R}_L, \mathcal{R}_U)$
11: **until** $c(B) > c(M)$ or $B =$ **null**
12: **if** $A \neq$ **null**
13: Update-Lower-Restriction$(A, \mathcal{R}_L)$
14: **end if**
15: **if** $B \neq$ **null then**
16: Update-Upper-Restriction$(B, \mathcal{R}_U)$
17: **end if**
18: Update-Results(M)
19: Minimum-Exhausting$(M, \mathcal{R}_L, \mathcal{R}_U)$

An element is called a *minimum exhausted* element in $\mathcal{L}$ if all its adjacent elements (upper and lower) have cost bigger than it. This definition can be extended to the poset $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$, i.e., all its adjacent elements (upper and lower) in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ have cost bigger than it. In Fig. 1 we can see that the elements 1010, 1001 and 0111 are minimum exhausted elements in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$, but 1001 is not a minimum exhausted element in $\mathcal{L}$. In this paper, the term minimum exhausted will be applied always referring to a poset $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$.

Algorithm 3. Minimum-Exhausting(Element M, ElementSet $\mathcal{R}_L$, ElementSet $\mathcal{R}_U$)

1: Push M to $\mathcal{S}$
2: **while** $\mathcal{S}$ is not empty **do**
3: $T \Leftarrow$ Top$(\mathcal{S})$
4: *MinimumExhausted* $\Leftarrow$ true
5: **for all** A adjacent of T in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ and $A \notin \mathcal{S}$ **do**
6: **if** $c(A) \leq c(T)$ **then**
7: Push A to $\mathcal{S}$
8: *MinimumExhausted* $\Leftarrow$ false
9: **else**
10: **if** A is upper adjacent of T **then**
11: Update-Upper-Restriction$(A, \mathcal{R}_U)$
12: **else**
13: Update-Lower-Restriction$(A, \mathcal{R}_L)$
14: **end if**
15: **end if**
16: **end for**
17: **if** *MinimumExhausted* **then**
18: Pop T from $\mathcal{S}$
19: Update-Results(T)
20: Update-Lower-Restriction$(T, \mathcal{R}_L)$
21: Update-Upper-Restriction$(T, \mathcal{R}_U)$
22: **end if**
23: **end while**
24: **return**

The *minimum exhausting* procedure (Algorithm 3) is a recursive process that visit all the adjacent elements of a given element M and turn all of them into minimum exhausted elements in the resulting poset $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$. It uses a stack $\mathcal{S}$ to perform the recursive process. $\mathcal{S}$ is initialized by pushing M to it and the process is performed while $\mathcal{S}$ is not empty (lines 2–22). At each iteration, the algorithm processes the top element T of $\mathcal{S}$: all the adjacent elements (upper and down) of T in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ and not in $\mathcal{S}$ are checked. If the cost of an adjacent element A is lower (or equal) than the cost of T then A is pushed to $\mathcal{S}$. If the cost of A is bigger than the cost of T then one of the restriction sets can be updated with A, lower restriction set if A is lower adjacent of T and upper restriction set if A is upper adjacent of T (lines 5–16). If T is a minimum exhausted element in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$, i.e., there is no adjacent element A in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ with cost lower (or equal) than T, then T is removed from $\mathcal{S}$ and, also, the restriction sets and the result list are updated with T (lines 19–21). At the end of this procedure all the elements processed are minimum-exhausted elements in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$.

Fig. 4 shows a graphical representation of the minimum exhausting process. Fig. 4A shows a chain construction process in up direction, the chain has its edges emphasized. The element $M = 010101$ (orange-colored) has the minimum cost over the chain. The elements in black are the elements eliminated from the search space by the restrictions obtained by the lower and upper adjacent elements of the local minimum M. The stack begins with the element M. Fig. 4B shows the first iteration of the minimum exhausting process. The arrows in red and the elements in red indicate the adjacent elements of M (top of the stack) that have cost lower (or equal) than it. These elements 010001 and 010111 are pushed to the stack. The adjacent elements of M with cost bigger than it can update the restriction sets, i.e., the lower adjacent element 000101 updates the lower restriction set and the upper adjacent element 000101 updates the upper restriction set. Fig. 4C shows the second iteration: the adjacent elements 010011 and 000111 with cost lower (or equal) than the new top element 010111 are pushed to the stack and the other adjacent elements 010110 and 011111 with cost bigger than 010111 update, respectively, the lower and upper restriction sets. In Fig. 4D the element 000111 is a minimum exhausted element (gray color) in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ and it is removed from stack. In Fig. 4E the elements eliminated by the new interval $[\emptyset, 000111]$ and $[000111, W]$ are turned into black color. At this point, 010011 is a minimum exhausted (gray color) in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ and it is removed from stack. From Figs. 4F to H all the elements are removed from stack and the elements removed by the new restrictions are turned into black color. Fig. 4H shows all the elements removed from a single minimum exhausted process.

The procedures to calculate minimal and maximal elements and the procedure to update lower and upper restriction sets will be discussed in the next section.

4. Mathematical foundations

This section introduces mathematical foundations of some modules of the algorithm.

4.1. Minimal and maximal construction procedure

Each iteration of the algorithm requires the calculation of a minimal element in $\mathcal{X}(\mathcal{R}_L)$ or a maximal element in $\mathcal{X}(\mathcal{R}_U)$. It is presented here a simple solution for that. The next theorem is the key for this solution.

Theorem 1. *For every* $A \in \mathcal{X}(\mathcal{R}_L)$,

$$A \in \mathcal{X}(\mathcal{R}_L) \;\Leftrightarrow\; A \cap R^c \neq \emptyset, \quad \forall R \in \mathcal{R}_L.$$

Proof. See the Appendix section.

Algorithm 4 implements the *minimal construction* procedure. It builds a minimal element C of the poset $\mathcal{X}(\mathcal{R}_L)$. The process begins with $C = (\underbrace{1 \ldots 1}_{n})$ and $S = (\underbrace{1 \ldots 1}_{n})$ and executes a n-loop (lines 3–16) trying to remove components from C. At each step, a component k, $k \in \{1, \ldots, n\}$ is chosen exclusively from S (S prevents multi-selecting). If the element C' resulted from C by

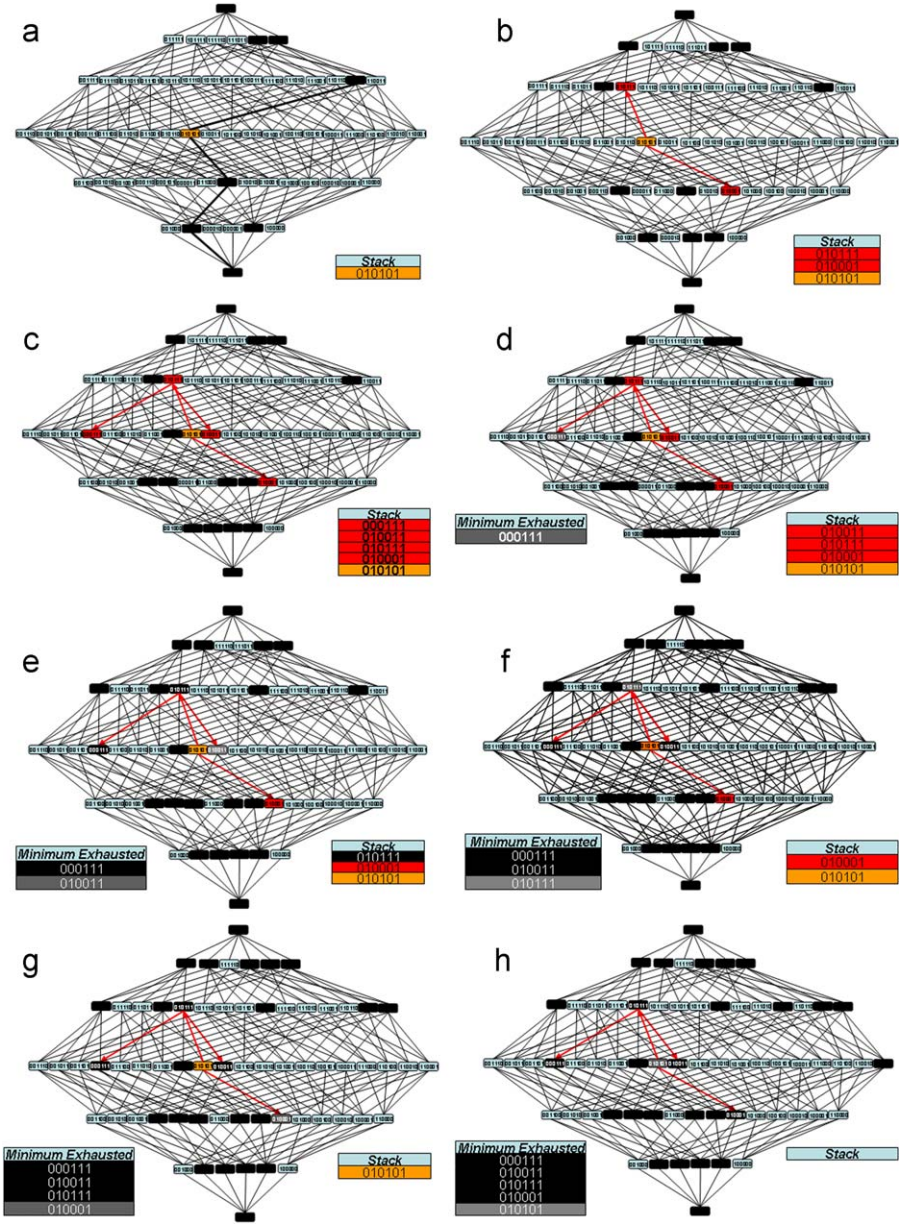

Fig. 4. Representation of the minimum exhausting process.

removing the component k is contained in $\mathcal{X}(\mathcal{R}_L)$ then C is updated with C' (lines 7–15).

Algorithm 4. Minimal-Element(ElementSet $\mathcal{R}_L$)

1: $C \Leftarrow \underbrace{1 \dots 1}_{n}$
2: $S \Leftarrow \underbrace{1 \dots 1}_{n}$
3: **while** $S \neq \underbrace{0 \dots 0}_{n}$ **do**
4: $k \Leftarrow$ random index in$\{1, \dots, n\}$ where $S[k] = 1$
5: $S[k] \Leftarrow 0$
6: $C' \Leftarrow C \backslash k$
7: *RemoveElement* $\Leftarrow$ true
8: **for all** R in $\mathcal{R}_L$ **do**
9: **if** $R^c \cap C' = \emptyset$ **then**
10: *RemoveElement* $\Leftarrow$ false
11: **end if**
12: **end for**
13: **if** *RemoveElement* **then**
14: $C \Leftarrow C'$
15: **end if**
16: **end while**
17: **return** C

The minimal element calculated is equal to $\underbrace{1 \dots 1}_{n}$ when $\mathcal{R}_L = \{\underbrace{1 \dots 1}_{n}\}$. At this point, the poset $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ is empty and the algorithm stops in the next iteration.

The next theorem proves the correctness of Algorithm 4.

Theorem 2. *The element C of $\mathcal{X}(\mathcal{R}_L)$ returned by the minimal construction process (Algorithm 4) is a minimal element in $\mathcal{X}(\mathcal{R}_L)$.*

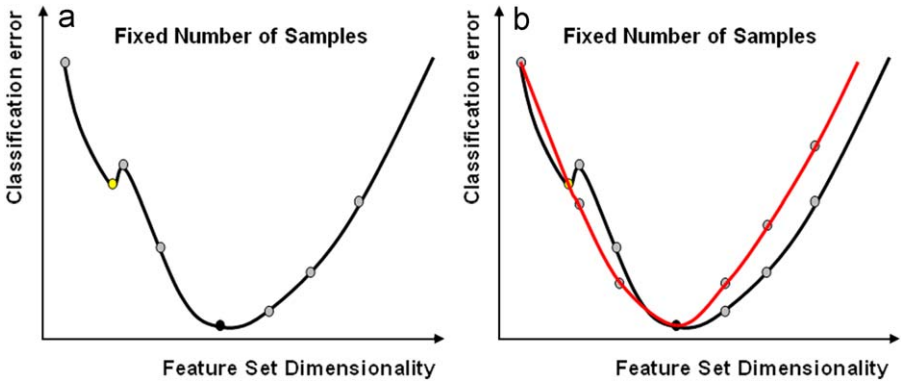

Fig. 5. Illustration of error curve oscillation and alternative way.

Proof. See the Appendix section.

The process to calculate a maximal element in $\mathcal{X}(\mathcal{R}_U)$ is *dual* to the one to calculate a minimal, i.e., it begins with $C = \underbrace{0 \ldots 0}_{n}$ and, at each step, when the complement C'^c of the resulting C' has not empty intersection to all the elements of $\mathcal{R}_U$, adds a component k to C.

4.2. Lower and upper restrictions update

The restriction sets $\mathcal{R}_L$ and $\mathcal{R}_U$ represent the search space. Thus, they are updated after each new search by the following rule: an element A is added to the lower (or upper) restriction set if all elements of $[\emptyset, A]$ (or $[A, W]$) have costs bigger or equal to A.

The next theorem establishes the U-curve condition that permits to stop the chain construction process and to update the restriction sets.

Theorem 3. *Let $C_0, \ldots, C_{k-1}, C_k$ be the chain constructed by Algorithm 2 (or its dual version). Let c be the cost function from $\mathcal{L}$ to $\mathbb{R}$ decomposable in U-shaped curves and $c(C_k) > c(C_{k-1})$, then*

$$\forall A \in \mathcal{L}, \quad C_k \subseteq A \Rightarrow c(A) \geq c(C_k).$$

Proof. See the Appendix section.

By a similar proof to the one of Theorem 3, it can be proved that all the elements in $\mathcal{L}$ contained in C_{k-2} have also cost bigger or equal to it. Fig. 3 shows the chain obtained by the chain construction process and the resulted poset. The elements detached have always cost bigger than the elements $C_k = (1 \ldots 11110 \ldots 0)$ or $C_{k-2} = (1 \ldots 1010 \ldots 0)$.

Algorithm 5 describes the update process of the lower restriction set by an element A. If A is already covered by $\mathcal{R}_L$, i.e., there exists an element of $\mathcal{R}_L$ that contains A then the process stops (lines 1–3). Otherwise, all the elements in $\mathcal{R}_L$ contained in A are removed from $\mathcal{R}_L$ and A is added to $\mathcal{R}_L$ (lines 4–9). This procedure may diminish the cardinality of the restriction set, but does not diminish the cardinality of the resulting poset $\mathcal{X}(\mathcal{R}_L)$, since the removed restrictions are contained in A.

Algorithm 5. Update-Lower-Restriction(Element A, ElementSet $\mathcal{R}_L$)

```
1:  if there exists R from R_L where A ⊆ R then
2:      return
3:  end if
4:  for all R in R_L do
5:      if R ⊆ A then
6:          R_L = R_L \ {R}
7:      end if
8:  end for
9:  R_L = R_L ∪ {A}
10: return
```

The upper restriction list updating procedure is *dual* to the lower one, i.e., in this case we look for elements contained in A instead of elements that contain A.

4.3. Minimum exhausting procedure

The computation of the cost function in general is heavy. Thus, it is desirable that each element be visited (and its cost computed) a single time. A way of preventing this reprocessing is to apply the minimum exhausting procedure. This procedure is a recursive function (Algorithm 3). It uses a stack $\mathcal{S}$ to process recursively all the neighborhood of a given element M contained in the poset $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$. At each recursion, it visits the upper and lower adjacent elements of T, the top of $\mathcal{S}$, in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ and not in $\mathcal{S}$. The adjacent elements with cost bigger than the cost of T are elements satisfying the U-curve condition, so they can update the restriction sets and, consequently, be removed from the search space. The adjacent elements with cost lower or equal to T are pushed to $\mathcal{S}$ to be processed in later iterations. Note that elements are not reprocessed during the exhausting procedure, since this procedure checks if a new element explored is in an interval or in $\mathcal{S}$, before computing its cost. If T is a minimum exhausted element in $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ then T is removed from $\mathcal{S}$. After the whole procedure is finished, all elements processed are out of the resulting poset $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$, so they will not be reprocessed in the next iterations. The fact that an element cannot be reprocessed along the procedure implies that the cardinality of $\mathcal{X}(\mathcal{R}_L, \mathcal{R}_U)$ is an upper limit for the procedure number of steps. In search spaces that are lattices with high degree, this procedure can have to process a huge number of elements and some heuristics should be necessary. For example, to stop the search for adjacent elements smaller than a minimum after some badly succeeded trials.

The minimum exhausting procedure gives another interesting property to the U-curve algorithm. If the cost function on maximal chains are U-shaped curves with oscillations, as illustrated in Fig. 5A, the U-curve algorithm may lose a local minimum element. Note that, in this case, the local minimum element after the oscillation has cost smaller than the cost of one before. However, this minimum is not lost if there is another chain, with a true U-shaped cost function, containing both local minimum elements. Fig. 5B shows an alternative chain (chain in red) that reaches the true minimum element of the chain (element in black). Note that

the first local minimum (element in yellow) is contained in both chains. The true minimum, reached by the alternative chain, is obtained exactly by the exhausting of the first minimum found. Hence, the exhausting procedure permits to relax the class of problems approached by the U-curve algorithm.

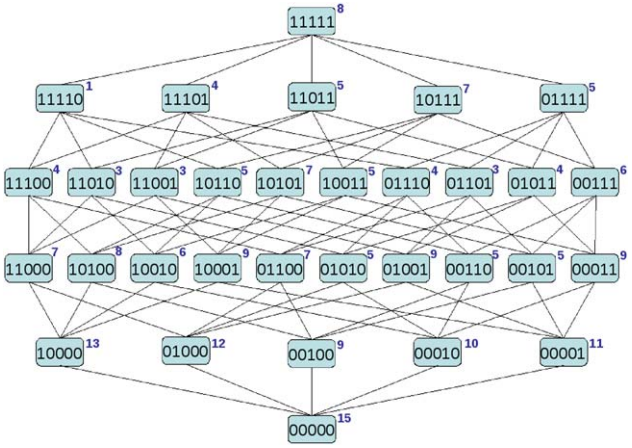

Fig. 6. Boolean lattice with dimension 5 used to illustrate the behavior of each compared feature selection technique. Its costs are designed in such way that most of the chains respect the U-shape property and a small number of chains present oscillations.

4.4. *Toy example to compare U-curve, SFFS and original branch-and-bound algorithms*

The purpose of this section is to illustrate the advantages of the U-curve algorithm over the other two feature selection techniques mentioned here (SFFS and original branch-and-bound) by showing the simulation of these methods over a small example. Fig. 6 contains a dimension 5 lattice with costs associated to the elements. Such costs are defined in such way that most of the chains from the element 00000 to 11111 respect the U-shaped cost property, while a small set of the chains present oscillations.

Starting with the classical SFFS algorithm, its application to the Boolean lattice of Fig. 6 leads to a sequence of steps shown in Fig. 7. In each step of the algorithm, it analyses the costs of the neighborhood of the current element in order to take the decision of moving forward or backward. It is a greedy algorithm in the sense that the direction taken in each iteration is based on the minimum cost found in the considered neighborhood. Because of its greedy nature, such method may not reach the optimal solution. In fact, for the Boolean lattice of Fig. 6, it does not return the optimal solution as can be seen in Fig. 7. In this sequence of

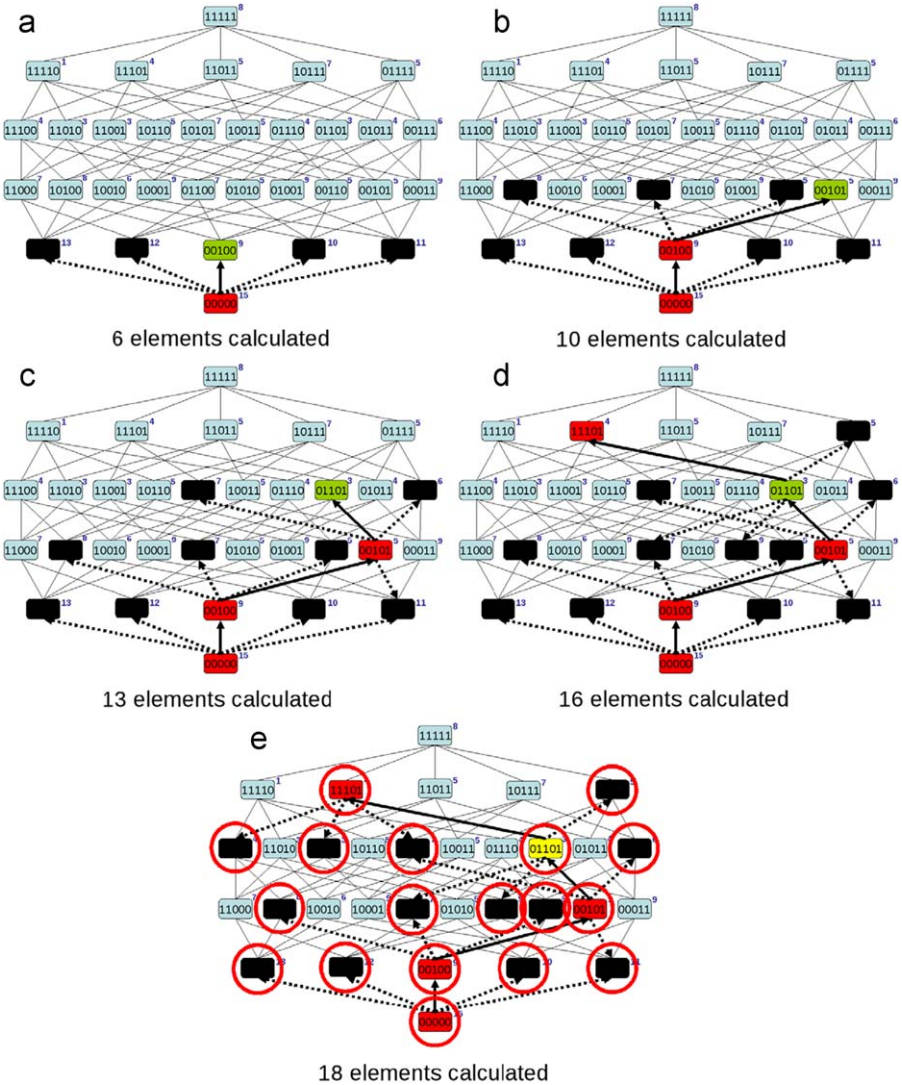

Fig. 7. Execution of the SFFS algorithm on the Boolean lattice of Fig. 6. The bold arrows indicate the chain that the algorithm went through. The slashed arrows correspond to the discarded edges to compose the selected pathway. There were 18 computed elements in this example.

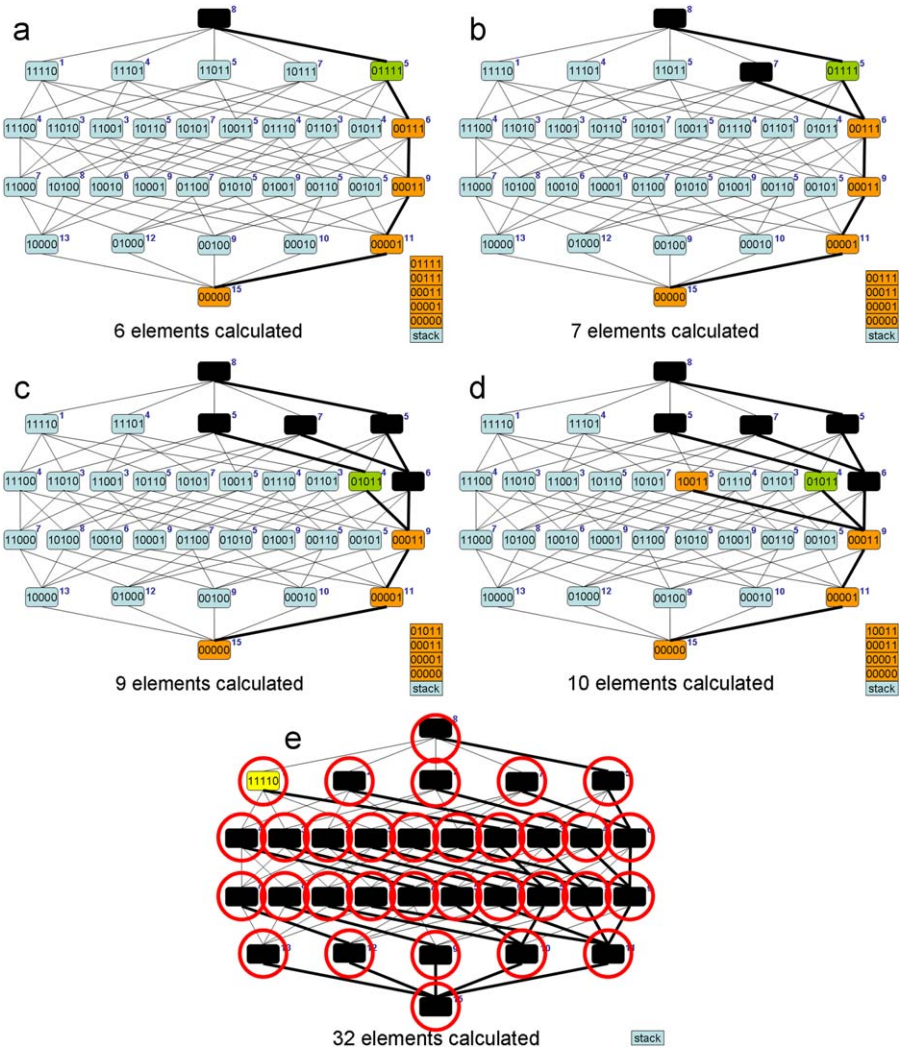

Fig. 8. Execution of the original branch-and-bound algorithm on the Boolean lattice of Fig. 6 (first four iterations shown). The bold edges indicate the chains that the algorithm went through. All elements of the lattice were calculated in this example.

slides, the red elements are those belonging to the selected chain. The discarded elements are in black. The green element is the current minimum cost element achieved and the yellow element in slide (e) is the element returned by the algorithm. The red circles in slide (e) correspond to the computed elements.

Fig. 8 shows the first four iterations along with the final state of the original branch-and-bound algorithm for the lattice of Fig. 6. The original branch-and-bound algorithm behaves similarly to the depth-first-search (DFS) in graphs, except for the fact that the branch-and-bound cuts all the elements above the minimum element of the current chain (upward cut). In this particular case, the performance of the branch-and-bound is not better than exhaustive search, since it finishes by calculating all the elements of the lattice. In the sequence of slides present in Fig. 8, the orange elements correspond to the current elements in the stack, the black elements are discarded elements, the green element is the current minimum, and the yellow element is the returned element. The red circles in slide (e) correspond to the computed elements.

Lastly, the U-curve algorithm execution on the example of Fig. 6 is illustrated in Fig. 9. The key point of the algorithm is the minimum exhausting procedure, which starts by looking at the neighborhood of the minimum element of a chain in order to search for other elements that improve the cost. If such elements are found, they are stacked and the same procedure is applied recursively to the stacked elements. Differently from the branch-and-bound algorithm, in the U-curve algorithm the cuts can be in both directions (upward or downward) depending on the location of the element to be discarded with relation to the stacked element (respectively above or below). This procedure tries to avoid local minima and at the same time to discard as many elements as possible, avoiding unnecessary computing of a subset of elements. In this example, 10 elements were discarded without need of computing them. Although it needs to calculate a few more elements than SFFS does, the U-curve finishes by returning the optimal solution. In the sequence of slides given in Fig. 9, the red elements are those calculated in the current chain, the orange elements are stacked for the application of the minimum exhausting procedure, the black elements are discarded, the green and yellow elements are the current minimum and the returned elements, respectively. The red circles in slide (e) correspond to the computed elements.

5. Experimental results

In this section, some results of applications of U-curve algorithm to feature selection are given and compared to SFFS

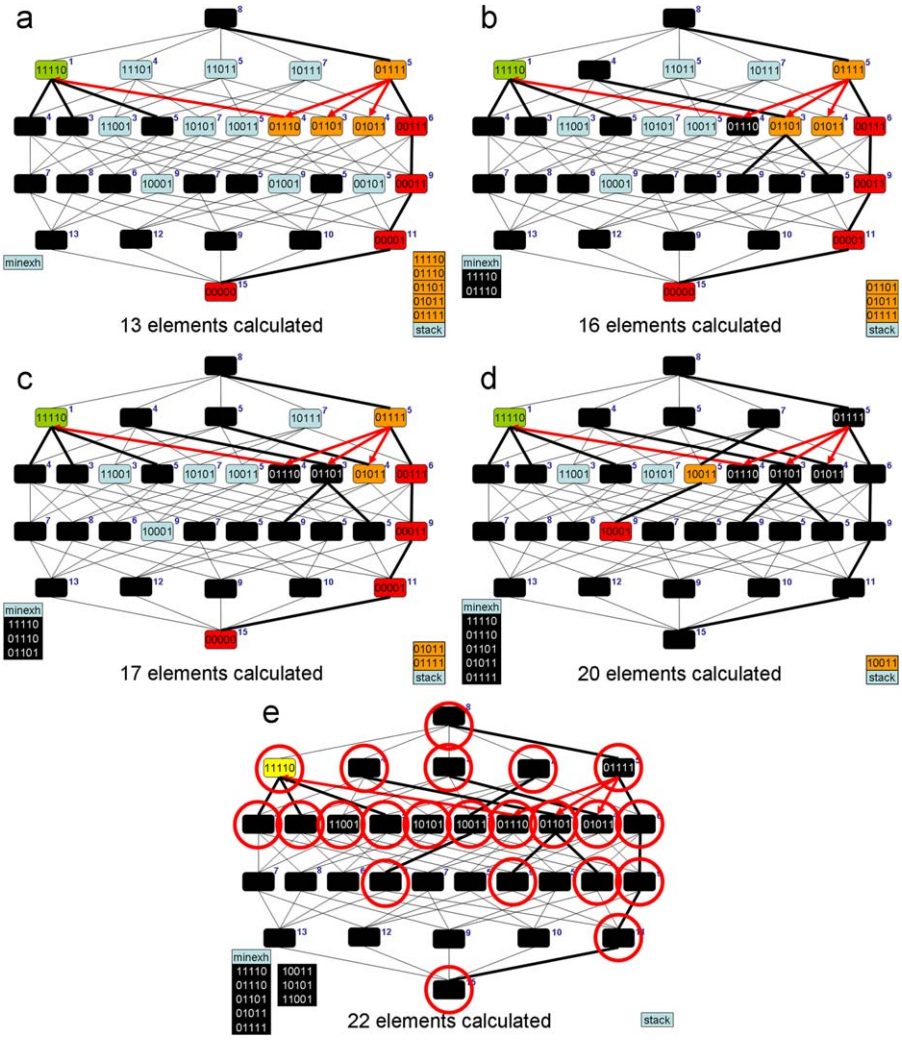

Fig. 9. Execution of the U-curve algorithm on the Boolean lattice of Fig. 6. The bold edges indicate the pathways that the algorithm went through. The red arrows correspond to the paths that originated elements stacked for the application of the minimum exhausting procedure. There were 22 computed elements in this example. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

[11]. For this study several data sets were used: W-operator window design [8], architecture identification in genetic networks and several data sets from the UCI Machine Learning Repository [1]. In all cases, it was attributed the value 3 for the parameter δ of SFFS. This parameter is a stop criterion of SFFS. Usually, $0 < \delta \leq 3$ in order to avoid that the algorithm stops at the first moment that it reaches the desired dimension. In this way, it performs more feature inclusion and deletion before returning the subset with the desired dimension, alleviating the nesting effect. The value $\delta = 3$ used as default here is the same default value adopted by the original algorithm implementation [11].

All data sets used and the binary program with some documentation can be found at the supplementary material web page (http://www.vision.ime.usp.br/~davidjr/ucurve).

5.1. Cost function adopted: penalized mean conditional entropy

The information theory was originated from Shannons works [12] and can be employed on feature selection problems [5]. Shannon's entropy H is a measure of randomness of a random variable Y given by

$$H(Y) = -\sum_{y \in Y} P(y)\log P(y) \tag{1}$$

in which P is the probability distribution function and, by convention, $0 \cdot \log 0 = 0$.

The conditional entropy is given by the following equation:

$$H(Y|\mathbf{X}=\mathbf{x}) = -\sum_{y \in Y} P(y|\mathbf{X}=\mathbf{x})\log P(y|\mathbf{X}=\mathbf{x}) \tag{2}$$

in which $\mathbf{X}$ is a feature vector and $P(Y|\mathbf{X}=\mathbf{x})$ is the conditional probability of Y given the observation of an instance $\mathbf{x} \in \mathbf{X}$. Finally, the mean conditional entropy of Y given all the possible instances $\mathbf{x} \in \mathbf{X}$ is given by

$$E[H(Y|\mathbf{X})] = \sum_{\mathbf{x} \in \mathbf{X}} P(\mathbf{x})H(Y|\mathbf{x}). \tag{3}$$

Lower values of H yield better feature subspaces (i.e., the lower H, the larger is the information gained about Y by observing $\mathbf{X}$).

In practice, $H(Y)$ and $H(Y|\mathbf{X})$ are estimated. A way to embed the error estimation, committed by using feature vectors with large dimensions and insufficient number of samples, is to attribute a high entropy (i.e., penalize) to the rarely observed instances. The penalization adopted here consists in changing the conditional probability distribution of the instances that present just a unique observation to uniform distribution (i.e., the highest entropy). This makes sense because if an instance $\mathbf{x}$ has only 1 observation, the value of Y is fully determined (i.e., $H(Y|\mathbf{X}=\mathbf{x}) = 0$), but the confidence about the real distribution of $P(Y|\mathbf{X}=\mathbf{x})$ is very low.

Adopting this penalization, the estimation of the mean conditional entropy becomes

$$\hat{E}[H(Y|\mathbf{X})] = \frac{N}{t} + \sum_{\mathbf{x} \in \mathbf{X}: \hat{P}(\mathbf{x}) > 1/t} \hat{P}(\mathbf{x})\hat{H}(Y|\mathbf{x}) \quad (4)$$

in which t is the number of training samples and N is the number of instances with $P(\mathbf{x}) = 1/t$ (i.e., just one observation). In this formula, it is assumed that the logarithm base is the number of possible classes $|Y|$, thus, normalizing the entropy values to the interval $[0, 1]$. This cost function exhibits U-shaped curves, since, for a sufficiently large dimension, the number of instances with a single observation starts to increase, increasing the penalization and, consequently, increasing the cost function value (i.e., next features included do not give enough information to compensate the error estimation).

5.2. Data sets description

5.2.1. W-operator window design

The W-operator window design problem consists in looking for subsets of a size n window for which the designed operator has the lowest estimation error (i.e., the transformed images generated by the operator are as similar as possible of the expected images). The training samples were obtained from the images presented in [8]. It is composed by 20 files with 18,432 samples each. There are 16 features assuming binary values and two classes.

5.2.2. Biological classification

The biological classification problem studied is the problem of estimating a subset of predictor genes for a specific target gene from a time-course microarray experiment. The data set used for the tests is the one presented in paper [9]. They are normalized and quantized in three levels using the same method described in Barrera et al. [3]. The subset of predictors is obtained from a set of 27 genes. Thus, there are 27 features assuming three distinct values and three possible classes. It is composed by 10 files with 15 samples each.

5.2.3. UCI Machine Learning Repository

UCI Machine Learning Repository data sets considered are: *pendigits*, *votes*, *ionosphere*, *dorothea_filtered*, *dexter_filtered*, *spambase*, *sonar and madelon*. For all data sets, the feature values were normalized by subtracting them from their respective means and dividing them by their respective standard deviations. After that, all values were binarized (i.e., associated to 0, if the normalized value is non-positive, and to 1, otherwise). Except for dorothea_filtered and dexter_filtered, all features were taken into account. The *dorothea_filtered* and *dexter_filtered* are files post-processed from *dorothea* and *dexter* data sets, respectively. In the *dorothea* and *dexter* data sets, most features display null value for almost every sample. So, *dorothea_filtered* considered only the features with 100 or more non-null values, while *dexter_filtered* considered the features with 50 or more non-null values.

A description of each data set is presented in the following list:

- *pendigits*: composed by 7494 samples, 16 binary features and 10 classes;
- *votes:* composed by 435 samples, 16 ternary features and 2 classes;
- *ionosphere*: composed by 351 samples, 34 binary features and 2 classes;
- *dorothea_filtered*: composed by 800 samples, 38 binary features and 2 classes;
- *dexter_filtered*: composed by 300 samples, 48 binary features and 2 classes;
- *spambase*: composed by 4601 samples, 57 binary features and 2 classes;
- *sonar*: composed by 208 samples, 60 binary features and 2 classes;
- *madelon*: composed by 2000 samples, 500 binary features and 2 classes.

5.3. Results

The feature selection problem may have cost functions with chains that present oscillations and there is no theoretical guaranty of the existence of alternative chains to achieve the local minima lost because of the oscillations. However, these cases were tested experimentally and in all observed cases the minimum exhausting procedure could find the local minimum elements using alternative chains. We have examined 100,000 random curves in all data sets studied. For example, in the W-operator window design almost 24,000 curves (24%) contains oscillatory parts and in the biological classifier design almost 15,000 curves (15%) contain oscillatory parts. For all these oscillatory curves and also for those found in the UCI data sets, the minimum exhausting procedure got the local minimum by alternative chains.

The results of the U-curve algorithm are divided into two sets: (i) until it beats the SFFS result (UC); (ii) until the search space is completely processed (UCC). The U-curve algorithm is stochastic and at each test it can reach the best result in different processing time. So, the U-curve was processed 5 times for each test and the quantitative results presented are means of values gotten in these five processes. The machine used for the tests was an AMD Turion 64 with 2 Gb of RAM.

In the following, each of the three experiments performed is summarized by a table and all these tables have the same structure. The first column presents the winner of the comparison of SFFS with UC. The other columns present the cost in terms of processed nodes and computational time of SFFS, UC and UCC.

Table 1 shows the results for the W-operator window design experiment. Twenty tests were performed using the available training samples. UC beats SFFS in 8 of the 20 tests and reaches the same result in the remaining ones. In these last cases, both reach the global minimum element. In all cases, UC processes a

Table 1
Comparison between SFFS and U-curve results for the W-operator window design.

Test	Winner	Computed nodes			Time (s)		
		SFFS	UC	UCC	SFFS	UC	UCC
1	EQUAL	358	73	373	8	2	393
2	EQUAL	333	31	154	7	1	392
3	EQUAL	417	17	137	10	1	393
4	UC	435	58	5,965	9	1	541
5	UC	357	101	223	7	3	385
6	UC	384	66	345	9	2	399
7	EQUAL	302	111	266	6	4	392
8	UC	1217	158	13,963	21	2	591
9	UC	330	31	274	8	1	385
10	EQUAL	406	113	825	10	4	408
11	EQUAL	329	70	544	7	2	387
12	EQUAL	336	17	17	8	0.5	0.5
13	EQUAL	310	26	26	8	1	384
14	UC	328	67	67	8	4	421
15	EQUAL	425	66	671	8	1	391
16	UC	333	31	151	8	1	377
17	EQUAL	1257	659	11,253	31	16	717
18	UC	336	39	218	7	1	385
19	EQUAL	296	32	137	6	2	379
20	EQUAL	323	31	151	8	2	376

Table 2
Comparison between SFFS and U-curve results for the biological classification design.

Test	Winner	Computed nodes			Time (s)		
		SFFS	UC	UCC	SFFS	UC	UCC
1	EQUAL	135	777	9964	0.5	0.6	3.1
2	UC	135	9252	30,724	0.5	2.1	11.2
3	UC	135	1037	9410	0.5	0.6	3.1
4	UC	164	786	9276	0.5	0.6	3.1
5	UC	281	247	6126	0.5	0.6	1.5
6	EQUAL	135	2675	11,031	0.5	0.7	7.3
7	EQUAL	135	998	10,836	0.5	0.6	6.9
8	UC	135	463	5381	0.5	0.5	1.5
9	UC	135	246	4226	0.5	0.5	1.5
10	UC	191	474	8930	0.5	0.5	2.9

Table 3
Comparison between SFFS results and U-curve algorithm for the UCI Machine Learning Repository data sets.

Test	Winner	Computed nodes			Time (s)		
		SFFS	UC	UCC	SFFS	UC	UCC
Pendigits (16)	EQUAL	358	124	1292	5	1	19
Votes (16)	EQUAL	128	81	12,670	0.03	0.02	87
Ionosphere (34)	UC	4782	1139	NA	1	0.25	NA
dorothea_filtered (37)	UC	7004	799	NA	10	1	NA
dexter_filtered (48)	UC	8071	596	NA	3	1	NA
Spambase (57)	UC	24,265	1608	NA	441	21	NA
Sonar (60)	UC	540	784	NA	0.09	0.10	NA
Madelon (500)	UC	66,403	159,745	NA	1000	3008	NA

smaller number of nodes, in a smaller time, than SFFS. The complete search (UCC) frequently needs to process more nodes (17/20), taking more time (19/20), than SFFS.

Table 2 shows the results for the biological classifier design experiment. Ten tests were performed using different target genes. In these examples, the complete search space is quite big (2^{27} nodes). SFFS reaches the best element, equalling UC, only 3/10 times. The processing of the whole space (UCC) improved the result of UC in 7/10 times. UC processed many more nodes than SFFS, but their computational times are very similar. This happens because these experiments involve small number of samples and, therefore, the computational time spent to process a node is very small. The pre-processing overhead is the major responsible for the time consuming in this case.

Table 3 shows the results of eight tests using public data sets. For each test, the value in parenthesis is the number of features (n) in the data set. For tests with high number of features, the results for the complete search (UCC) are not available. We can see that UC obtained better results than SFFS in 6/8 of the tests and equal results in two tests with small number of features. In these two cases, SFFS reaches the best result but UC reaches them faster, processing less nodes.

These results show that UC is more efficient than SFFS for low order problems, obtaining the same results with less processing. For high order problems, UC is more accurate, but in some cases it process more nodes and takes more time.

6. Conclusion

This paper introduces a new combinatorial problem, the Boolean U-curve optimization problem, and presents a stochastic branch-and-bound solution for it, the U-curve algorithm. This algorithm gives the optimal elements of a cost function decomposable in U-shaped chains that may even be oscillatory in a given sense. This model permits to describe the feature selection problem in the context of pattern recognition. Thus, the U-curve algorithm constitutes a new tool to approach feature selection problems.

The U-curve algorithm explores the domain and cost function particular structures. The Boolean nature of the domain permits to represent the search space by a collection of upper and lower restrictions. At each iteration, a beginning of chain node is computed from the search space restrictions. The current explored chain is constructed from this node by choosing upper or lower adjacent nodes. The choice of a beginning of chain and of an adjacent node usually has several options and one of them is taken randomly. The cost function and domain structure permit to make cuts in the search space, when a local minimum is found in a chain. After a local minimum is found, all local minimum nodes connected to it are computed, by the minimum exhausting procedure, and the corresponding cuts, by up-down intervals, executed. The adjacency and connectivity relations adopted are the ones of the search space Hesse diagram that is a graph in which the connectivity is induced by the partial order relation. The minimum exhausting procedure avoids that a node be visited more than once and generalizes the algorithm to cost functions decomposable in some class of U-shaped oscillatory chain functions. The procedures of the U-curve algorithm are supported by formal results.

In fact, the U-curve optimization technique constitutes a new framework to study a family of optimization problems. The restrictions representation and the intervals cut, based on Boolean lattice properties, constitutes a new optimization structure for combinatorial problems, with properties not found in conventional tree representations.

The U-curve was applied to practical problems and compared to SFFS. The experiments involved window operator design, genetic network identification and six public data sets obtained from the UCI repository. In all experiments, the results of the U-curve algorithm were equal or better than those obtained from SFFS in precision and, in many cases, even in performance. The results of the U-curve algorithm considered for comparison are the mean of several executions for the same input data, since it is a stochastic algorithm that may have different performances at each run.

The efficiency of the U-curve algorithm depends on the relative position of the local minima on the search space. The algorithm is more efficient when the local minima are near the search space extremities. The worst cases are the ones in which the local minima are near the middle of the lattice.

The results obtained until now are encouraging, but the present version of the U-curve algorithm is not a fast solution for high dimension problems with many local minima in the center of the search space lattice. The efficient addressing of these problems in the U-curve optimization approach opens a number of subjects for future researches such as: to develop additional cuts to the branch-and-bound formulation; to design and estimate distributions for the random parameters used in the choice of beginning nodes or adjacent paths in the construction of a chain, with the goal of reaching earlier to the best nodes; to build parallelized versions of the algorithm; and others.

Acknowledgments

The authors are grateful to FAPESP (99/12765-2, 01/09401-0, 04/03967-0 and 05/00587-5), CNPq (300722/98-2, 468 413/00-6, 521097/01-0 474596/04-4 and 491323/05-0) and CAPES for financial support. This work was partially supported by Grant 1 D43 TW07015-01 from the National Institutes of Health, USA. We also thank Helena Brentani by her helpful in the data for biological

analysis and Roberto M. Cesar Jr. by his helpful in SFFS comparisons. The data sets used to generate Table 3 results were obtained from UCI Machine Learning Repository [1].

Appendix

Theorem 1. *For every* $A \in \mathcal{X}(\mathcal{R}_L)$,

$A \in \mathcal{X}(\mathcal{R}_L) \Leftrightarrow A \cap R^c \neq \emptyset, \forall R \in \mathcal{R}_L$.

Proof.

$A \in \mathcal{X}(\mathcal{R}_L) \Leftrightarrow A \in \mathcal{L}$

$- \bigcup\{[\emptyset, R]$

$: R \in \mathcal{R}_L\} \Leftrightarrow A \notin \bigcup\{[\emptyset, R]$

$: R \in \mathcal{R}_L\} \Leftrightarrow A \notin [\emptyset, R], \forall R \in \mathcal{R}_L \Leftrightarrow A \not\supseteq R, \forall R \in \mathcal{R}_L \Leftrightarrow A \cap R^c \neq \emptyset, \forall R \in \mathcal{R}_L$ □

Theorem 2. *The element* C *of* $\mathcal{X}(\mathcal{R}_L)$ *returned by the minimal construction process* (*Algorithm* 4) *is a minimal element in* $\mathcal{X}(\mathcal{R}_L)$.

Proof. By looking into the steps of the minimal construction procedure:

- Lines 7–15 guarantee that at any step of the procedure the resulted C is contained in $\mathcal{X}(\mathcal{R}_L)$, i.e., it is updated only when the resulted C' satisfies the condition shown in Theorem 1.
- Let $C_1, \ldots, C_n$ be the sequence of resulting elements at each step i ($i = 1, \ldots, n$) and $C_0 = \underbrace{1 \ldots 1}_{n}$ be the initial element. As an index k is chosen to be removed from C_{i-1} (lines 4–6) at each step i, it implies that $C_n \subseteq C_{n-1} \subseteq \cdots \subseteq C_0$.
- Proving that the resulting element C_n is minimal in $\mathcal{X}(\mathcal{R}_L)$ is equivalent of proving that $\forall l \in C_n, C_n \backslash \{l\} \notin \mathcal{X}(\mathcal{R}_L)$.
- Let $k = l, l \in C_n$ and i be the step of the procedure when the index l is chosen to be removed from C_{i-1}. $C_n \subseteq C_i$ and $l \in C_n$ imply that $l \in C_i$, i.e., l cannot be removed from C_{i-1} at the end of step i. This is avoided by the algorithm (lines 8–12), when there exists an element $R \in \mathcal{R}_L$ with $R^c \cap (C_{i-1} \backslash \{l\}) = \emptyset$. As $C_n \backslash \{l\} \subseteq C_{i-1} \backslash \{l\}$, then $R^c \cap (C_n \backslash \{l\}) = \emptyset$ and, by Theorem 1, $C_n \backslash \{l\} \notin \mathcal{X}(\mathcal{R}_L)$. This implies that C_n is a minimal element in $\mathcal{X}(\mathcal{R}_L)$. □

Theorem 3. *Let* $C_0, \ldots, C_{k-1}, C_k$ *be the chain constructed by Algorithm* 2 (*or its dual version*). *Let* c *be the cost function from* $\mathcal{L}$ *to* $\mathbb{R}$ *decomposable in U-shaped curves and* $c(C_k) > c(C_{k-1})$. *It is true that*

$\forall A \in \mathcal{L}, \quad C_k \subseteq A \Rightarrow c(A) \geq c(C_k)$.

Proof. Suppose that $\exists B \in \mathcal{L}, C_k \subseteq B$ and $c(B) < c(C_k)$. It contradicts the hypothesis that c is a function decomposable in U-shaped curves, since $C_{k-1} \subseteq C_k \subseteq B$, but $\max(c(C_{k-1}), c(B))$ is either $c(C_{k-1}) < c(C_k)$ or $c(B) < c(C_k)$, contradicting $\max(c(C_{k-1}), c(B)) > c(C_k)$. □

References

[1] A. Asuncion, D. Newman, UCI Machine Learning Repository, 2007.
[2] G.J.F. Banon, J. Barrera, Minimal representations for translation-invariant set mappings by mathematical morphology, SIAM Journal on Applied Mathematics 51 (6) (1991) 1782–1798.
[3] J. Barrera, R.M. Cesar Jr., D.C. Martins Jr., R.Z.N. Vencio, E.F. Merino, M.M. Yamamoto, F.G. Leonardi, C.A.B. Pereira, H.A. del Portillo, Constructing Probabilistic Genetic Networks of *Plasmodium falciparum* from Dynamical Expression Signals of the Intraerythrocytic Development Cycle, Springer, Berlin, 2006, pp. 11–26 (Chapter 2).
[4] J. Barrera, G.P. Salas, Set operations on collections of closed intervals and their applications to the automatic programming of morphological machines, Electronic Imaging 5 (3) (1996) 335–352.
[5] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, vol. 1, second ed., Wiley-Interscience, New York, pp. 1–19.
[6] A. Frank, D. Geiger, Z. Yakhini, A distance-based branch and bound feature selection algorithm, in: Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03). Morgan Kaufmann Publishers, San Francisco, CA, 2003, pp. 241–248.
[7] A.K. Jain, R.P.W. Duin, J. Mao, Statistical pattern recognition: a review, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (1) (2000) 4–37.
[8] D.C. Martins Jr., R.M. Cesar Jr., J. Barrera, W-operator window design by minimization of mean conditional entropy, Pattern Analysis and Applications 9 (2006) 139–153.
[9] C. Lin, A. Ström, V.B. Vega, S.L. Kong, A.L. Yeo, J.S. Thomsen, W.C. Chan, B. Doray, D.K. Bangarusamy, A. Ramasamy, L.A. Vergara, S. Tang, A. Chong, V.B. Bajic, L.D. Miller, J. Gustafsson, E.T. Liu, Discovery of estrogen receptor α target genes and response elements in breast tumor cells, Genome Biology 5 (9) (2004) 1–18.
[10] S. Nakariyakul, D.P. Casasent, Adaptive branch and bound algorithm for selecting optimal features, Pattern Recognition Letters 28 (2007) 1415–1427.
[11] P. Pudil, J. Novovicová, J. Kittler, Floating search methods in feature selection, Pattern Recognition Letters 15 (1994) 1119–1125.
[12] C.E. Shannon, A mathematical theory of communication, Bell System Technical Journal 27 (1948) 379–423 623–656.
[13] P. Somol, P. Pudil, Fast branch and bound algorithms for optimal feature selection, Pattern Analysis and Machine Intelligence 26 (7) (2004) 900–912.
[14] Z. Wang, J. Yang, G. Li, An improved branch and bound algorithm in feature selection, in: Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing: 9th International Conference, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Chongqing, China, 2003, pp. 549–556.
[15] S. Yang, P. Shi, Bidirectional automated branch and bound algorithm for feature selection, Journal of Shanghai University 9 (3) (2005) 244–248.

About the Author—MARCELO RIS received a B.Sc. in Computer Science (Universidade de São Paulo-USP, Brazil), a M.Sc. in Computer Science (Universidade de São Paulo-USP, Brazil) and a Ph.D. in Bioinformatics (Universidade de São Paulo-USP, Brazil). His main research topics are in bioinformatics, including algorithms design for gene network identification, pattern recognition for computer vision and algorithm parallelism.

About the Author—JUNIOR BARRERA received a B.Sc. in Electrical Engineering (Universidade de São Paulo-USP, Brazil), a M.Sc. in Applied Computing (Instituto Nacional de Pesquisas Espaciais - INPE, Brazil) and a Ph.D. in Electrical Engineering (Universidade de São Paulo-USP, Brazil). His main research topics are study of lattice operator representation and design, lattice dynamical systems, image processing, bioinformatics and computational biology. He is currently a full professor at the Department of Physics and Mathematics of Faculdade de Filosofia Ciências e Letras de Ribeirão Preto (Universidade de São Paulo-SP, Brazil) and president of the Brazilian Society for Bioinformatics and Computational Biology.

About the Author—DAVID C. MARTINS Jr. received his B.Sc., M.Sc. and Ph.D. in Computer Science at Universidade de São Paulo - USP, Brazil. His main research topics are in pattern recognition for computer vision and bioinformatics, including but not limited to gene network identification. Recently he did a research stage at the Genomic Signal Processing Laboratory - Texas A. & M. University during one year. He is currently a post doctoral researcher at Instituto de Matemática e Estatística (Universidade de São Paulo - USP, Brazil).