A New Paradigm for the Architecture of Morphological Machines: Binary Decision Diagrams

Heraldo M. F. Madeira[1],
J. Barrera[2],
R. Hirata Jr.[2] and
N. S. T. Hirata[2]

[1] Departamento de Informática - UFPR
PO Box 19081
81531-990 Curitiba - PR - Brazil
heraldo@inf.ufpr.br

[2] Instituto de Matemática e Estatística - USP
PO Box 66.281
05315-970 São Paulo - SP - Brazil
<jb,hirata,nina>@ime.usp.br

Abstract. A central paradigm in Mathematical Morphology (MM) is the representation of set operators in terms of erosions, intersection, union, complementation and composition. A hardware or software that implements this decomposition structure is called a morphological machine (MMach). The architecture of all known MMachs has as central characteristic a small kernel with very fast procedures. Another well known decomposition result in MM is the representation of W-operators (*i.e.*, translation invariant and locally defined) as an union of sup-generating (*i.e.*, hit-or-miss) operators. In particular, erosion is a sup-generating operator. A remarkable property of this decomposition structure is that it can be represented efficiently by a graph called Binary Decision Diagram (BDD). In this paper, we propose a new architecture for an MMach that is based on BDDs and we compare it with the conventional architecture.

1 Introduction

Mathematical Morphology (MM) is a theory that studies images and signals based on transformations of their shapes. The simplest class of useful images are the binary images. In MM, binary images and their transformations are modeled, respectively, by subsets of the integer plane and by mappings on the powerset of the integer plane, called *set operators*.

A central paradigm in MM is the representation of set operators in terms of erosions, intersection, union, complementation and composition of set operators. This decomposition structure can be described by a formal language, called *morphological language*. An implementation of the morphological language in hardware or software is called a *morphological machine* (MMach). A phrase of the morphological language or, equivalently, a program of a morphological machine is called a *morphological operator*.

The first known MMach was the Texture Analyzer, created in the late sixties in Fontainebleau by Jean Serra and Jean Claude Klein. Nowadays, a large number of these machines are available: from software for conventional architectures to implementations in silicon or optical technologies. This family of computational systems has spread out so powerfully thanks to its adequacy to extract image information. This capacity is largely evidenced by the solution of hundreds of image processing problems in domains so diverse as cytology, automation, cartography, remote sensing, etc.

The architecture of all known MMachs have exactly the same structure: very fast dedicated processors or algorithms for erosion, intersection, and complementation; several image planes and facilities for programming (i.e., scheduling long sequences of calls of operators and operations). The most complex and critical part of these architectures are the processors and algorithms for erosion.

In this paper, we propose a new architecture for MMachs implemented as software for sequential machines. This new architecture is based on the representation of Boolean functions by Binary Decision Diagrams (BDD). The inspiration for this new concept

comes from the original work of Luc Robert[1], that presented a fast BDD implementation for a particular morphological operator.

A classical result in Mathematical Morphology is that any W-operator (i.e., translation invariant and locally defined in a finite window W) can be represented by a finite union of sup-generating operators (*i.e.*, intersection of an erosion and an erosion composed with complement – also called *hit-or-miss* operator). This representation is called *standard representation*. A property of the standard representation is that it can be trivially converted in an equivalent Boolean function, represented in the disjunctive form. There are well known algorithms to convert Boolean formulas into BDDs. Furthermore, there are well known procedures for converting any morphological operator in the standard representation form.

Therefore, we have a systematic way to convert any morphological operator in a BDD: compute the standard representation; transform the standard representation into a canonical form Boolean function; transform the canonical form into a BDD.

This procedure permits the construction of an MMach, whose elementary processor is a BDD interpreter. Thus, instead of fixing erosion as the elementary operator, this new architecture permits that any W-operator in the standard form plays this role. In this context, morphological operators are concatenations of operators in the standard form by the operations of composition, intersection, union and complementation.

We give some examples of applications of this new architecture to represent useful set operators and we do comparisons with other implementations of architectures.

Following this Introduction, Section 2 recalls the standard representation of W-operators and formulas for computing the standard representation for any morphological operator. Section 3 presents the technique for the conversion of a standard form morphological operator into an equivalent canonical form Boolean function. Section 4 recalls the main concepts on BDDs and the technique for converting a generic Boolean function into a BDD. Section 5 gives several illustrative examples of morphological operators implemented in an MMach with the proposed architecture. Section 6 gives some conclusions and future steps of this research.

2 Decomposition of binary image operators

Let $\mathbb{E}$ be the discrete plane, that is, $\mathbb{E} = \mathbb{Z} \times \mathbb{Z}$, and $\mathcal{P}(\mathbb{E})$ be the powerset of $\mathbb{E}$. The translation of a subset $X \in \mathcal{P}(\mathbb{E})$ by a vector $h \in \mathbb{E}$ is denoted $X + h$, or X_h, and defined as $X + h = \{x + h : x \in X\}$, where $+$ denotes the usual vector addition.

A binary image operator Ψ can be defined as a transformation between subsets of $\mathbb{E}$, i.e., $\Psi : \mathcal{P}(\mathbb{E}) \to \mathcal{P}(\mathbb{E})$. We say that $\Psi : \mathcal{P}(\mathbb{E}) \to \mathcal{P}(\mathbb{E})$ is translation invariant (t.i.) if and only if (iff) $\forall X \in \mathcal{P}(\mathbb{E})$ and $h \in \mathbb{E}$, $\Psi(X + h) = \Psi(X) + h$.

Let $W \subseteq \mathbb{E}$ be a non empty and finite subset of $\mathbb{E}$, called *window*. We say that Ψ is locally defined (l.d.) in W iff, $\forall X \in \mathcal{P}(\mathbb{E})$ and $\forall h \in \mathbb{E}$, $h \in \Psi(X) \Leftrightarrow h \in \Psi(X \cap (W + h))$. The family of operators that are t.i. and l.d. in a window W is denoted by Ψ_W and their elements are called W-operators.

If Ψ is a W-operator, $z \in \Psi(X)(z)$ iff $\psi(X \cap (W + z)) = 1$, where $\psi : \mathcal{P}(W) \to \{0, 1\}$. Hence, $\Psi \in \Psi_W$ can be characterized by a unique Boolean function ψ on $\mathcal{P}(W)$. This Boolean function ψ so defined is called the *characteristic function* of Ψ [2].

The set $K_W(\Psi)$, $\Psi \in \Psi_W$, defined by $K_W(\Psi) = \{X \in \mathcal{P}(W) : \psi(X) = 1\}$ is the *kernel* of Ψ.

Let $A, B \in \mathcal{P}(W)$, the subset $[A, B] \subseteq \mathcal{P}(W)$ defined by $[A, B] = \{X \in \mathcal{P}(W) : A \subseteq X \subseteq B\}$ is an interval with extremities A and B.

Let $A, B \in \mathcal{P}(W)$, $A \subseteq B$, the operator defined by, $\lambda^W_{[A,B]}(X) = \{x \in \mathbb{E} : A \subseteq (X - x) \cap W \subseteq B\}$, for $X \in \mathcal{P}(\mathbb{E})$, is called *sup-generating* operator [3]. Note that $\lambda^W_{[A,B]}$ is locally defined within W and can be decomposed in terms of erosions [3].

Let $\mathbf{X}$ be a collection of intervals $[A, B] \subseteq \mathcal{P}(W)$. An element $[A, B] \in \mathbf{X}$ is said to be *maximal* in $\mathbf{X}$ if no other element of $\mathbf{X}$ properly contains it, i.e., $\forall [A, B] \in \mathbf{X}$, $[A, B]$ is maximal in $\mathbf{X}$ iff, $\forall [A', B'] \in \mathbf{X}$, $[A, B] \subseteq [A', B'] \Rightarrow [A, B] = [A', B']$. We denote by $\mathrm{Max}(\mathbf{X})$ the collection of all maximal intervals in $\mathbf{X}$.

Let $\mathcal{X}$ be a subcollection of $\mathcal{P}(W)$. The collection of all maximal intervals contained in $\mathcal{X}$, $M(\mathcal{X})$, is defined by $M(\mathcal{X}) = \mathrm{Max}(\{[A, B] \subseteq \mathcal{P}(W) : [A, B] \subseteq \mathcal{X}\})$.

The set $\mathcal{B}_W(\Psi)$ of all maximal intervals contained in $K_W(\Psi)$ is called the basis of Ψ, i.e., $\mathcal{B}_W(\Psi) = M(K_W(\Psi))$.

The following theorem shows how a W-operator can be represented in terms of its basis [3].

Theorem 2.1 *Let Ψ be a W-operator. For any $X \in \mathcal{P}(\mathbb{E})$,*

$$\Psi(X) = \cup\{\lambda^W_{[A,B]}(X) : [A, B] \in \mathcal{B}_W(\Psi)\} \qquad (1)$$

The above representation of $\Psi \in \Psi_W$ is known as the *standard representation* and it shows how Ψ is uniquely defined by its basis [3].

Theorem 2.1 does not show, however, a direct way to find the basis of a given operator Ψ. In the remain-

ing of this section, we recall some constructive results for finding the basis of any morphological operator.

Let $\mathbb{I}_W$ denote the set $\{M(\mathcal{X}) : \mathcal{X} \subseteq \mathcal{P}(W)\}$. It can be proved[2] that $\mathbb{I}_W$ is a complete Boolean lattice [4], where the infimum, supremum and negation are given, respectively, by: $\forall \mathbf{X}, \mathbf{Y} \in \mathbb{I}_W$,

$$\mathbf{X} \sqcap \mathbf{Y} = M(\mathcal{X} \cap \mathcal{Y}) \quad (2)$$

$$\mathbf{X} \sqcup \mathbf{Y} = M(\mathcal{X} \cup \mathcal{Y}) \quad (3)$$

$$\overline{\mathbf{X}} = M(\mathcal{X}^c) \quad (4)$$

where $\mathbf{X} = M(\mathcal{X})$ and $\mathbf{Y} = M(\mathcal{Y})$.

We first recall how to compute the basis of the negation of a W-operator. Let Ψ be a W-operator with basis $\mathcal{B}_W(\Psi)$ then

$$\mathcal{B}_W(\nu\Psi) = \overline{\mathcal{B}}_W(\Psi) \quad (5)$$

i.e., the basis of the negation of Ψ is the negation of the basis of Ψ.

Let Ψ_1 be a W_1-operator and Ψ_2 be a W_2-operator with bases $\mathcal{B}_{W_1}(\Psi_1)$ and $\mathcal{B}_{W_2}(\Psi_2)$, respectively. The bases of the infimum and supremum of Ψ_1 and Ψ_2 are given, respectively, by:

- $\mathcal{B}_{W_1 \cup W_2}(\Psi_1 \wedge \Psi_2) = \mathcal{B}_{W_1 \cup W_2}(\Psi_1) \sqcap \mathcal{B}_{W_1 \cup W_2}(\Psi_2)$
- $\mathcal{B}_{W_1 \cup W_2}(\Psi_1 \vee \Psi_2) = \mathcal{B}_{W_1 \cup W_2}(\Psi_1) \sqcup \mathcal{B}_{W_1 \cup W_2}(\Psi_2)$

Note that the resulting operator is a $W_1 \cup W_2$-operator.

Let Ψ be a W-operator with basis $\mathcal{B}_W(\Psi)$. The basis of Ψ composed with the dilation by B, δ_B, or with the erosion by B, ε_B, are given, respectively by:

$$\mathcal{B}_{W \oplus B^t}(\delta_B \psi) = \sqcup_{b \in B} \mathcal{B}_{W \oplus B^t}(\psi + b) \quad (6)$$

$$\mathcal{B}_{W \oplus B}(\varepsilon_B \psi) = \sqcap_{b \in B^t} \mathcal{B}_{W \oplus B}(\psi + b) \quad (7)$$

Since any W-operator can be expressed in terms of erosions, dilations, unions, intersections and complements, Eq. 5 to 7 are sufficient to generate the basis of any W-operator from the identity operator.

In Section 5, we give an example of application of these properties.

3 Conversion of Representations

The characteristic function $\psi : \mathcal{P}(W) \to \{0,1\}$ of a W-operator $\Psi : \mathcal{P}(\mathbb{E}) \to \mathcal{P}(\mathbb{E})$ is equivalent to a Boolean function $f : \{0,1\}^{|W|} \to \{0,1\}$, with $|W|$ variables. This function f is unique, and it takes the value 1 whenever the input variables combination corresponds to an input set $X \in \mathcal{P}(\mathbb{E})$ satisfying the condition $0 \in \Psi(X)$, and takes 0 otherwise.

Since the operator is a member of Ψ_W, only the elements of $(X_{-x} \cap W)$ are needed to evaluate ψ. Thus, for the position x on the output set, the value of input variables, given by the vector $\mathbf{w} = (w_1, w_2, \ldots, w_{|W|}) \in \{0,1\}^{|W|}$, is determined by the *indicator function* taken in the set $X_{-x} \cap W$, that is,

$$\mathbf{w} = \mathbb{1}(X_{-x} \cap W).$$

In the canonical form representation of f by a sum of *minterms*, each minterm corresponds to an element of the kernel of Ψ. The usual simplification techniques for Boolean functions of the switching theory [5] can be applied to simplify the representation of Ψ to the form of Theorem 2.1. In fact, the set of all *prime implicants* of the Boolean function determined by the first stage of the Quine-McCluskey method matches exactly the basis of Ψ [3].

Barrera and Salas [2] introduce a method to incrementally compute $\mathrm{B}(\Psi)$, given the description of the operator in the morphological language. Now, given $\mathrm{B}(\Psi)$, represented by a set of intervals $[A, B]$, with $A \subseteq B \subseteq W$, the function f is straightforwardly obtained.

The function of f will be represented by the following disjunction of its prime implicants:

$$f = \bigvee_{i=1}^{|\mathcal{B}(\Psi)|} \pi^{\mathbf{w}}_{\mathbf{a_i},\mathbf{b_i}} \quad (8)$$

where $(\mathbf{a_i}, \mathbf{b_i}) = (\mathbb{1}(A_i), \mathbb{1}(B_i))$, and each prime implicant is evaluated by the expression

$$\pi^{\mathbf{w}}_{\mathbf{a},\mathbf{b}} = \prod_{j=1}^{|W|} (\overline{a_j} \cdot \overline{w_j} \vee b_j \cdot w_j) \quad (9)$$

Note the similarity between Eq. 8 and Theorem 2.1. In Eq. 9 we observe that, if some element (a_j, b_j) appears as $(0,1)$, the corresponding variable does not influence the product. If they appear both as 0 or as 1, then the variable appears complemented or non-complemented in the product, respectively. Note that the combination $(1,0)$ is not possible because $A \subseteq B$. Hence, directly looking at the extremities of the intervals of the operator's basis, we get the exact expression of f, which is commonly the input for the algorithm that computes the decision diagram for the operator.

4 Binary Decision Diagrams

Given a Boolean function f, its cofactors with respect to x_i and $\overline{x_i}$ are $f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ and $f_{\overline{x_i}} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$. The Shannon expansion of f around a variable x_i is given by

$$f = x_i \cdot f_{x_i} + \overline{x_i} \cdot f_{\overline{x_i}} \quad (10)$$

A Boolean function can be represented by a rooted, directed acyclic graph with two type of nodes: terminal and nonterminal. Each terminal node v has as attribute a value $value(v) \in \{0,1\}$. Each nonterminal node v is assigned to a variable $var(v) \in \{x_1, x_2, \ldots, x_n\}$ and has two children nodes, $low(v)$ and $high(v)$. For a given assignment to the variables $x_1, x_2, \ldots, x_n$, the value of the function is determined by traversing the graph from the root to a terminal node: at each nonterminal node v, $var(v) = x_i$, if $x_i = 0$, then the arc to $low(v)$ must be followed and if $x_i = 1$, then the arc to $high(v)$ must be followed. The value of the function is given by the value of the terminal node. These graphs are called binary decision diagrams (BDD) [6].

Let $ord : \{x_1, x_2, \ldots, x_n\} \rightarrow \{1, 2, \ldots, n\}$ be a bijective function which defines an ordering of the variables.

Definition 4.1 *An ordered BDD (OBDD) is a BDD such that any path in the graph from the root to a terminal node visits the variables in ascending order (i.e., $ord(var(v)) < ord(var(low(v)))$ whenever v and $low(v)$ are nonterminal, and $ord(var(v)) < ord(var(high(v)))$ whenever v and $high(v)$ are nonterminal).*

Proposition 4.1 *A node v in an OBDD denotes a Boolean function f^v such that*

- *If v is a terminal node with $value(v) = 0$, then $f^v = \mathbf{0}$*
- *If v is a terminal node with $value(v) = 1$, then $f^v = \mathbf{1}$*
- *if v is a nonterminal node and $var(v) = x_i$, then $f^v = \overline{x_i} \cdot f^{low(v)} + x_i \cdot f^{high(v)}$*

Definition 4.2 *If an OBDD contains no node v such that $low(v) = high(v)$, nor any pair of nodes $\{u, v\}$ representing the same Boolean function, it is called a reduced OBDD (ROBDD).*

Bryant [6] showed that ROBDD is a canonical form for logic functions. The size (number of nodes) of an ROBDD depends on the variable ordering chosen. Choosing the ordering that results in the smallest ROBDD is a difficult problem, and will not be discussed here. An exact solution exists that doesn't test all possible orderings, but is still impractical ($O(n^2 3^n)$) [5]. It is reported that, in practice, many useful functions can be represented by an ROBDD of tractable size.

The first implementations for the building of ROBDDs consisted of two steps: building of the graph followed by reduction of the graph size. The reduction of an OBDD consists of eliminating duplicate terminals, duplicate nonterminals and redundant tests. This top-down building followed by a bottom-up reduction approach has a drawback that the size of non reduced graphs are usually very large. Most recent and efficient implementations are based on a operator called ITE (*if-then-else* operator). ITE is an operator which takes three Boolean functions f, g, h as the input, and returns the result corresponding to the operation : If f then g else h. More formally,

$$ITE(f, g, h) = f \cdot g + \overline{f} \cdot h\,.$$

ITE operator can be used to implement any operation between two Boolean functions, as shown in Table 1.

Operation	Equivalent ITE form
0	0
$f \cdot g$	$ite(f, g, 0)$
$f \cdot \overline{g}$	$ite(f, \overline{g}, 0)$
f	f
$\overline{f} \cdot g$	$ite(f, 0, g)$
g	g
$f \oplus g$	$ite(f, \overline{g}, g)$
$f + g$	$ite(f, 1, g)$
$\overline{f + g}$	$ite(f, 0, \overline{g})$
$\overline{f \oplus g}$	$ite(f, g, \overline{g})$
$\overline{g}$	$ite(g, 0, 1)$
$f + \overline{g}$	$ite(f, 1, \overline{g})$
$\overline{f}$	$ite(f, 0, 1)$
$\overline{f} + g$	$ite(f, g, 1)$
$\overline{f \cdot g}$	$ite(f, \overline{g}, 1)$
1	1

Table 1: The operations implemented by ITE.

A hash table, called unique-table, is used to assure that each node in the ROBDD represents a unique logical function. Each entry in this table is a triple $(var(v), low(v), high(v))$ and is mapped to the node v. Before a new node is inserted to the ROBDD, this table is searched to determine if a node representing the same function already exists in the table. If it exists, then the existing node is used. Otherwise a new node is created and inserted into the table.

Let f, g, and h be three functions represented in ROBDD form respectively by the nodes u, v and w, and let x be the variable with smaller order among the top variables, $var(u)$, $var(v)$ and $var(w)$. The function $z = ITE(f, g, h)$ can be computed by the following

recursive formula:

$$\begin{aligned} z &= x \cdot z_x + \overline{x} \cdot z_{\overline{x}} \\ &= x \cdot (f \cdot g + \overline{f} \cdot h)_x + \overline{x} \cdot (f \cdot g + \overline{f} \cdot h)_{\overline{x}} \\ &= x \cdot (f_x \cdot g_x + \overline{f}_x \cdot h_x) + \overline{x} \cdot (f_{\overline{x}} \cdot g_{\overline{x}} + \overline{f}_{\overline{x}} \cdot h_{\overline{x}}) \\ &= x \cdot ite(f_x, g_x, h_x) + \overline{x} \cdot (f_{\overline{x}}, g_{\overline{x}}, h_{\overline{x}}) \\ &= ite(x, ite(f_x, g_x, h_x), ite(f_{\overline{x}}, g_{\overline{x}}, h_{\overline{x}})) \end{aligned}$$

The stop conditions for this recursion are:
$ite(1, g, h) = g$, $ite(0, g, h) = h$ and $ite(f, 1, 0) = f$.

Next we show an example on how to build the ROBDD for the function $f = x_1 \cdot x_3 + x_2 \cdot x_3$, considering the variable order $ord(x_1) < ord(x_2) < ord(x_3)$. First, the ROBBD for the variables are build. It results in three nodes $v(x_1)$, $v(x_2)$ and $v(x_3)$. The ROBDD for the term $x_1 \cdot x_3$ can be build by calling $v(x_1 \cdot x_3) = ite(x_1, x_3, 0) = ite(x_1, ite(1, x_3, 0), ite(0, x_3, 0)) = (x_1, v(x_3), v(0))$. The ROBDD for the term $x_2 \cdot x_3$ can be constructed in a similar way, i.e., $v(x_2 \cdot x_3) = (x_2, v(x_3), v(0))$. Finally, the ROBDD for f can be constructed calling $v(f) = ite(x_1 \cdot x_3, 1, x_2 \cdot x_3) = ite(x_1, ite(x_3, 1, x_2 \cdot x_3),\ ite(0, 1, x_2 \cdot x_3)) = (x_1, v(x_3), v(x_2 \cdot x_3))$.

The number of comparisons to evaluate a Boolean function f represented by an ROBDD for a given input $(x_1, x_2, \ldots, x_n)$ is, at most, equal to the longest path from the root to a terminal node, that, of course, is not greater than n.

5 Examples

In this section we present the implementation of several morphological operators in an MMach with the new BDD architecture. We also compare the cost of these implementations with classical ones.

A morphological operator Ψ implemented in a conventional MMach have a cost that we generically denote as $k(\Psi)$. The same operator on a BDD based MMach will have a cost denoted by $k_{BDD}(\Psi)$. In our analysis, we consider as cost unit the time spent in applying a basic operation on it, *i.e.*, logical AND, logical OR, negation, and test.

For both approaches, we analyze the costs for: the composition of dilations and of erosions; the median filter; the 4-homotopic thinning; the supremum of openings; and the extreme point detector.

The cost of the elementary operations and operators of the morphological language are summarized in Table 2. N represents the image size in pixels. Table 3 presents the cost for the sup and inf-generator, as well as other useful cost expressions.

In the BDD architecture, the operator is represented by the corresponding ROBDD which is applied for each pixel in the image. The cost of this algorithm is given by

$$k_{BDD}(\Psi) = N \cdot |W(\Psi)| \tag{11}$$

where $W(\Psi)$ is the support of the characteristic function of Ψ, also known as the *window* of Ψ. The cost per pixel of the BDD algorithm is the number of tests made when traversing the diagram from top to bottom, hence, it is the number of Boolean variables that the function depends on. Note that this is the worst case cost, since some or all paths in the BDD may have less variables, meaning that the effective cost depends on the input image.

Identity	$k(\iota) = 0$
Complement	$k(\nu) = N$
Erosion	$k(\varepsilon_B) = N \cdot (\lvert B \rvert - 1)$
Dilation	$k(\delta_B) = N \cdot (\lvert B \rvert - 1)$
Composition	$k(\Psi_1 \Psi_2) = k(\Psi_1) + k(\Psi_2)$
Union	$k(\Psi_1 \vee \Psi_2) = k(\Psi_1) + k(\Psi_2) + N$
Intersection	$k(\Psi_1 \wedge \Psi_2) = k(\Psi_1) + k(\Psi_2) + N$

Table 2: Costs of elementary operators and operations

$k(\lambda_{A,B}) = k(\varepsilon_A) + k(\nu\delta_{B^{ct}}) + N = N \cdot (\lvert A \rvert + \lvert B^c \rvert)$
$k(\nu\lambda_{A,B}) = k(\lambda_{A,B})$
$k(\mu_{A,B}) = k(\delta_A) + k(\nu\varepsilon_{B^{ct}}) + N = N \cdot (\lvert A \rvert + \lvert B^c \rvert)$
$k(\Psi_1 \Psi_2 \ldots \Psi_n) = \sum_{i=1}^{n} k(\Psi_i)$
$k(\vee_{i=1}^{n} \Psi_i) = \sum_{i=1}^{n} k(\Psi_i) + N \cdot (n-1)$
$k(\wedge_{i=1}^{n} \Psi_i) = \sum_{i=1}^{n} k(\Psi_i) + N \cdot (n-1)$

Table 3: Some useful cost expressions

5.1 Composition of Dilations or Erosions

As shown in Table 2, the cost of either a dilation or an erosion is equal to $N \cdot (|B| - 1)$. According to Eq. 11, this is a bit smaller than the cost of δ_B or ε_B in a BDD based MMach. BDDs for each case are shown in Fig. 1, respectively, for the cross and the elementary square structuring elements.

For the case of a composition of dilations ($\delta_B = \delta_{B_1}\delta_{B_2} \ldots\ldots \delta_{B_n}$), with $B = B_1 \oplus B_2 \oplus \cdots \oplus B_N$ and $\oplus$ denoting the Minkowski addition, the costs are given by

$$k(\delta_B) = \sum_{i=1}^{n} k(\delta_{B_i}) = N \cdot \sum_{i=1}^{n} |B_i|$$

and

$$k_{BDD}(\delta_B) = N \cdot |B|$$

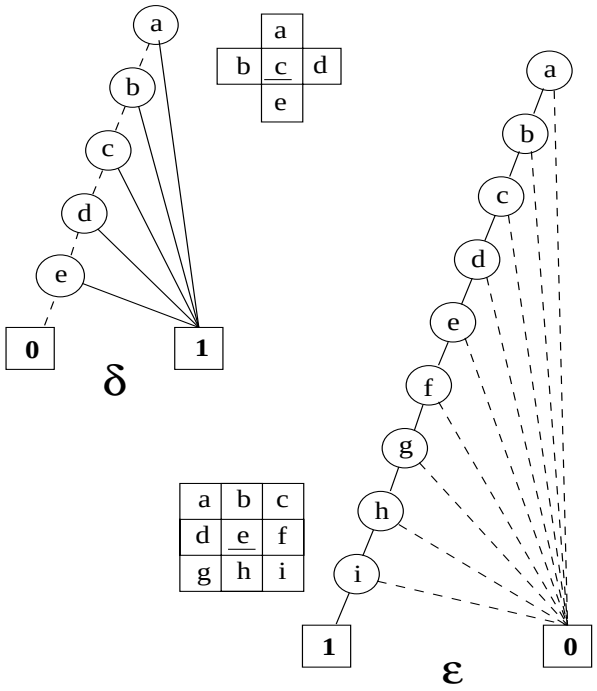

Figure 1: BDDs for dilation and erosion

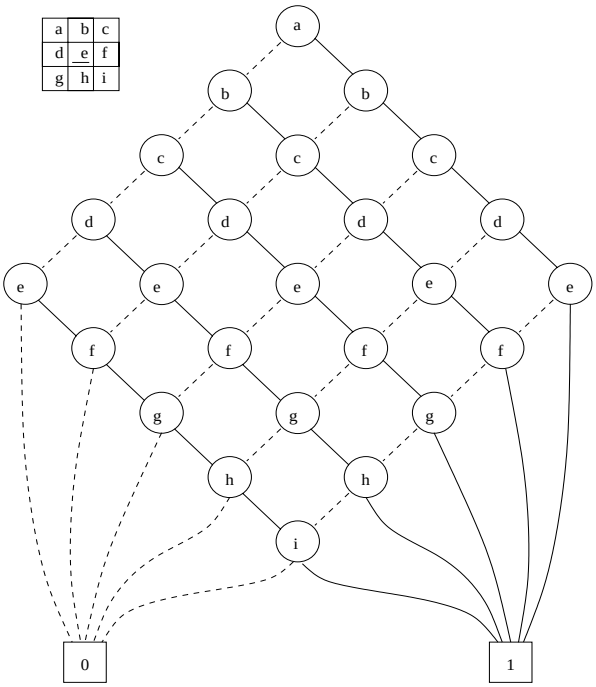

Figure 2: BDD for a 3×3 median filter.

The latter is often much greater than the former. Generally speaking, when we embed compositions in the BDD, the performance of the BDD implementation tends to degrade compared to the conventional one. Although this degradation is not always the case, it turns out to be mandatory that the BDD based MMach also support the composition operation. The analysis for the composition of erosions is analogous and the results are the same.

5.2 Median Filter

The median filter is extensively used in digital image processing and is defined as:

$$\mu_B(X) = \{x \in \mathbb{E} : |X_{-x} \cap B| \geq \frac{|B|+1}{2}\}$$

The kernel of μ_B consists of all the subsets of B that have $\frac{|B|+1}{2}$ elements or more, supposing $|B|$ odd. Its basis consists of all intervals of kind $[A, B]$ such that A are those subsets of B with just $\frac{|B|+1}{2}$ elements. There are $\binom{|B|}{\frac{|B|+1}{2}}$ of them.

By Theorem 2.1 the operator may be written in the morphological language as:

$$\mu_B = \vee\{\varepsilon_A : [A, B] \in \mathcal{B}(\mu_B)\}$$

Thus, the conventional implementation of μ_B costs

$$k(\mu_B) = N \cdot (\binom{|B|}{\frac{|B|+1}{2}} \cdot \frac{|B|+1}{2} - 1)$$

The BDD for the median filter has a very simple structure, and an example for $|B| = 9$ is shown in Fig. 2. For that example, $k(\mu_B) = 630N$, and $k_{BDD}(\mu)_B = 9N$.

The cost of a BDD based median filter is

$$k_{BDD}(\mu_B) = N \cdot |B|$$

which is obviously much better than that of the conventional MMach.

5.3 Four-homotopic Thinning

The thinning operator is defined as

$$\sigma_{A,B} = \iota \wedge (\nu\ \lambda_{A,B})$$

where A and B are the extremities of an appropriate interval. An interval family that leads to a 4-homotopic thinning, denoted here simply by σ, is composed by the eight rotations of the interval $\mathcal{I} = \begin{bmatrix} 0 & 0 & 0 \\ \times & 1 & \times \\ 1 & 1 & 1 \end{bmatrix}$ which has (and all rotations also have): $|A| = 4$ and $|B^c| = 3$.

n	$\lvert W(\sigma_n)\rvert$	$k_{BDD}(\sigma_n)$	$k(\sigma_n)$
1	7	$7N$	$8N$
2	13	$13N$	$16N$
3	23	$23N$	$24N$
4	33	$33N$	$32N$
5	43	$43N$	$40N$
6	55	$55N$	$48N$

Table 4: Window sizes and costs for Thinning.

The cost of this operator is:

$$\begin{aligned} k(\sigma) &= k(\iota \wedge \nu\ (\varepsilon_A \wedge \nu\delta_{B^{ct}})) \\ &= k(\iota \wedge (\nu\varepsilon_A \vee \delta_{B^{ct}})) \\ &= N \cdot (1 + 1 + (|A| - 1) + 1(|B^c| - 1)) = 8N. \end{aligned}$$

In practice, for the thinning of an image, we use the composed operator:

$$\sigma_n = \sigma^{0^o}\ \sigma^{45^o}\ \sigma^{90^o}\ \ldots\ \sigma^{(n-1)\cdot 45^o}$$

which, evidently, costs $8 \cdot N \cdot n$.

Evaluating the basis of σ_n according to [2], we get the data shown in Table 4. We can see that, for the obtained data, the BDD based implementation beats the conventional one, with maximum speedup when $n = 2$. This means that, if we construct the operator as a hybrid composition of four BDD based sub-operators (composition of two consecutive rotation thinnings), the cost will be

$$k_{HYBRID}(\sigma_8) = N \times (13 + 13 + 13 + 13) = 52N$$

then we get a complete turn (360^o) thinning operator implementation which is $\frac{64}{52} - 1 = 23\%$ faster than the conventional one.

5.4 Supremum of Openings

The operator $\Psi = \bigvee_{i=1}^{n}(\gamma_{B_i})$ is called *supremum of openings*, and is very useful for making "soft" openings over an image.

The cost of the conventional implementation of an opening operator is

$$k(\gamma_B) = k(\varepsilon_B) + k(\delta_B) = 2 \cdot N \cdot |B|.$$

Thus, the cost for the supremum of openings is

$$\begin{aligned} k(\bigvee_{i=1}^{n} \gamma_{B_i}) &= \sum_{i=1}^{n} k(\gamma_{B_i}) + N \cdot (n-1) \\ &= N \cdot (2 \sum_{i=1}^{n} |B_i| - n - 1). \end{aligned}$$

Supposing $|B_i| = b$, $\forall i \in \{1, ..., n\}$, then,

$$k(\bigvee_{i=1}^{n} \gamma_{B_i}) = N \cdot (2 \cdot n \cdot b - n - 1).$$

In the following, we calculate the cost k_{BDD} for the supremum of openings. We know that $W(\gamma_{B_i}) = B_i \oplus B_i$. Thus,

$$W(\bigvee_{i=1}^{n} \gamma_{B_i}) = \bigcup_{i=1}^{n} (B_i \oplus B_i).$$

So, the cost of the operator in a BDD based MMach is given by

$$k_{BDD}(\bigvee_{i=1}^{n} \gamma_{B_i}) = N \cdot |\bigcup_{i=1}^{n} (B_i \oplus B_i)|.$$

Comparing the performance of both algorithms, we conclude that the BDD based one will surpass the conventional one if $|\cup (B_i \oplus B_i)| < n + 2\sum |B_i|$. This expression says that, the more the structuring elements B_i overlap each other, the more the BDD based implementation gets better. Also, for a number of terms greater than a certain n, the BDD based algorithm will be better than the classical one. The following two examples illustrate this.

The first example is a union of openings by the 8 structuring elements shown if Fig. 3a. The basis of the operator is contained in the 9×9 square. For it, the BDD based MMach has the following advantage:

$$|\cup (B \oplus B)| = 57 < 71 = 2\sum |B_i| - n - 1$$

The other example is a supremum of openings by the 12 structuring elements shown if Fig. 3b. The basis of the operator is contained in the 13×13 square. For it, the advantage of a BDD implementation is

$$|\cup (B \oplus B)| = 129 < 155 = 2\sum |B_i| - n - 1$$

That is, the BDD implementations are, respectively, 24% and 20% faster than their correspondent in a conventional MMach.

5.5 Extreme Points Detection

The detection of extreme points in an image is made by the hit-miss operator, indicating the points in the input image where it matches one of the eight patterns given by the rotation of the following structuring element:

$$\mathcal{I} = [A, B] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ \times & \times & \times \end{bmatrix}$$

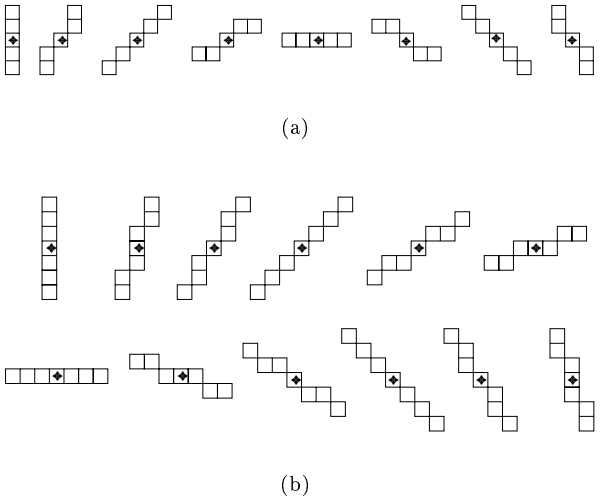

Figure 3: Structuring elements for union of openings

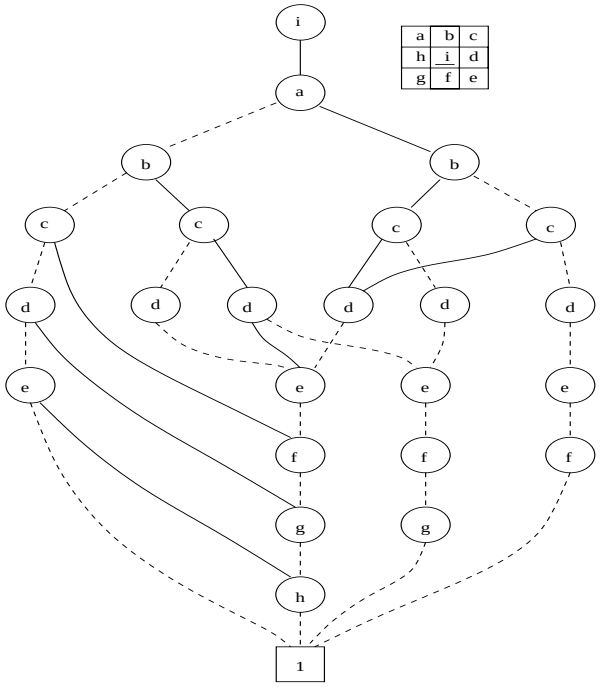

Figure 4: BDD for the extreme points detector

For all rotations $\mathcal{I}_i$ of this interval, we have $|A| = 1$ and $|B^c| = 5$. Therefore, the cost of the corresponding sup-generator is $k(\lambda_{A,B}) = 6N$. The extreme point detector is given by

$$\chi = \bigvee_{i=0}^{7} \lambda_{A_i,B_i}$$

and its cost is $k(\chi) = 8 \times k(\lambda_{A_i,B_i}) + 7 = 55$.

The union of the eight intervals is still restricted to a 3×3 window, so the cost for the BDD implementation of the operator is $k_{BDD}(\chi) = 9N$, which results in a much faster algorithm. Figure 4 shows the BDD for this operator. In this figure, the missing arcs point to the **0** node.

6 Conclusion

In this paper, we have proposed a new architecture for MMachs, where the elementary operator is not just erosion, but any operator in the standard representation implemented as a BDD.

We have presented an automatic procedure for computing the BDD representing any W-operator. This procedure consists of three steps: compute the standard representation of the morphological operator; convert the operator basis into a canonical form Boolean function; compute the BDD by a generic Boolean function conversion.

The proposed architecture is the most efficient alternative to implement a MMach that uses images represented in the conventional representation by an array of directly addressable pixels (*e.g.* one pixel per BYTE.)

The complexity of the erosion implemented as a BDD is the best possible in this data representation, that is, proportional to $|B| - 1$. So the compression of any other morphological operator in a BDD can be only beneficial. MMachs of this kind have the advantage of not changing the image structure before and after applying the morphological operator.

Several examples of implementation of morphological operators in the new architecture have been presented. The analysis of these examples shows that, in general, it is beneficial to compact parallel operators (i.e., built by union of several operators) in BBDs, but not sequential ones (i.e., built by composition of operators). In complex hybrid morphological operators, usually it may be beneficial to compact just pieces of the operator in BDDs. For example, the operator that extracts a succession of end points from lines can be efficiently represented by iterations of a BBD that extracts just one level of end points.

Between the architectures that change the image structure, the most interesting is the one that adopts the pixel BIT array representation. This architecture may be better or not than the BDD one, depending on the morphological operator considered. For example, the median operator (for windows with more than 9 points) is faster in the BDD than in the BIT MMach.

The next step of this research is the study of other option for converting morphological operators into BDDs, the implementation of an MMach based on BDDs and the extension of this study to gray-scale morphological operators.

Acknowlegement

The authors acknowledge partial support from UFPR, CAPES, CNPq and FAPESP.

References

[1] L. Robert and G. Malandain. Fast Binary Image Processing Using Binary Decision Diagrams. *Computer Vision and Image Understanding*, 72(1):1–9, October 1996.

[2] J. Barrera and G. P. Salas. Set Operations on Closed Intervals and Their Applications to the Automatic Programming of Morphological Machines. *Electronic Imaging*, 5(3):335–352, July 1996.

[3] G. J. F. Banon and J. Barrera. Minimal Representations for Translation-Invariant Set Mappings by Mathematical Morphology. *SIAM J. Appl. Math.*, 51(6):1782–1798, December 1991.

[4] G. Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, Rhode Island, 1967.

[5] R. Jacobi. Síntese de Circuitos Lógicos Combinacionais. Décima Escola de Computação, Campinas, Julho 1996.

[6] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.