DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-9504

Automatic Programming of Binary Morphological Machines by PAC Learning

Junior Barrera
Nina Sumiko Tomita
Flávio Soares C. Silva
Routo Terada

abril 95

Automatic programming of Binary Morphological Machines by PAC Learning

Junior Barrera
Nina Sumiko Tomita
Flávio Soares Corrêa da Silva
Routo Terada

Universidade de São Paulo, Departamento de Ciência da Computação
Cidade Universitária "Armando de Salles Oliveira", Caixa Postal 20570, 01452–990, São Paulo, SP, Brazil

ABSTRACT

Binary Image Analysis problems can be solved by set operators implemented as programs for a Binary Morphological Machine (BMM). This is a very general and powerful approach to solve this type of problems. However, the design of these programs is not a task manageable by non experts on Mathematical Morphology. In order to overcome this difficulty we have worked on tools that help users describe their goals at higher levels of abstraction and to translate them into BMM programs. Some of these tools are based on the representation of the goals of the user as a collection of input–output pairs of images and the estimation of the target operator from these data. PAC learning is a well suited methodology for this task, since in this theory "concepts" are represented as Boolean functions that are equivalent to set operators. However, to apply this technique in practice we must have efficient learning algorithms. In this paper we introduce two PAC learning algorithms, both are based on the minimal representation of Boolean functions, which has a straightforward translation to the canonical decomposition of set operators. The first algorithm is based on the classical Quine–McCluskey algorithm for the simplification of Boolean functions, and the second one is based on a new idea for the construction of Bollean functions: the incremental splitting of intervals. We also present a comparative complexity analysis of the two algorithms. Finally, we give some application examples.

1. INTRODUCTION

Binary Image Analysis is an important tool for various areas, such as industrial process control, office automation, quantitative microscopy, etc. A natural model for a procedure for Binary Image Analysis is a *set operator* (i.e. a function between subsets). *Mathematical Morphology* (MM) is a general framework to study complete lattice operators (i.e. functions between complete lattices) [1], which includes set operators [2 3]. The central paradigm of MM is the decomposition of operators in terms of four classes of *elementary operators*: dilations, erosions, anti–dilations and anti–erosions.

The rules for the representation of set operators in terms of the elementary operators can be described as a formal language [4], the *Binary Morphological Language* (BML). The vocabulary of the BML are the four classes of elementary operators and the operations of union and intersection. A phrase of the BML is called a *morphological operator*. The BML is complete (i.e. it can represent any set operator) and expressive (i.e. many useful operators can be represented as phrases with relatively few words). An implementation of this language is called a *Binary Morphological Machine* (BMM), and a *program of a BMM* is an implementation of a morphological operator in this machine.

Nowadays, there are many commercially available BMM's implemented in hardware [5 6 7 8] or emulated in software [9 10 11], which have been intensively used for Binary Image Analysis [3].

Programming a BMM can be a very difficult task. In order to help the non experts in MM to use BMM's, some tools have been proposed to automate the design of programs. These tools act as translators of the user knowledge about the problem (expressed as high level abstract procedures) into morphological operators.

One of the simplest procedures to represent the user knowledge is as a sequence of input–output pairs of images. The user should get some typical images from a given domain (e.g., geographical maps, handwritten texts, electronic circuits, etc.) and create by hand the corresponding desired outputs. These data would feed a designer system which would give as output a morphological operator that performs the desired task on the images of that domain.

The theory of PAC (Probably Approximately Correct) learning uses set theory and statistics to formalize the process of computational learning of concepts from sequences of examples. A concept is a subset of a given domain or, equivalently, a Boolean function on that domain. An example is an object extracted randomly from the domain, associated with a label indicating if the concept is true or false for that object. The learning process consists of the construction of a Boolean function which is a good approximation for the concept that should be learned (the target concept). This process must be consistent, that is, when the sequence of input examples increases the Boolean function estimated becomes a better approximation of the target concept. Under this theory, it is possible to calculate the minimal number of examples that is needed to estimate a concept under a given quantitative quality criterion.

Locally defined set operators (i.e., operators that depend just on pixels inside a local neighborhood) are equivalent to Boolean functions that map geometrical structures found in the images of the domain into zero or one. Thus, set operators can be seen as learnable concepts and estimated by PAC learning [12]. Given a sequence of pairs of images and a window W, a sequence of examples is built by extracting geometrical structures (or configurations) seeing through translations of W and labelling them according with the corresponding values at the output images.

In order to apply this technique in practice we need efficient learning algorithms. In this paper we introduce two PAC learning algorithms. Both are based on a minimal representation of Boolean functions, which has a straightforward translation to the canonical decomposition of set operators. The first algorithm is based on the classical Quine–McCluskey algorithm for the simplification of Boolean functions and the second one is based on a new idea for the construction of Boolean functions: the incremental splitting of intervals.

In section 2, we give some basic definitions and results from MM theory. In section 3, we present the PAC learning model. In section 4, we present the proposed algorithms for PAC learning and analyze their complexity. In section 5, we show some application examples. Finally, in section 6, we present some further discussion.

2. MATHEMATICAL MORPHOLOGY

For the automatic programming of BMM's some relevant aspects of the theory of MM on sets are: the canonical decompositions of set operators, the modeling of the shape recognition problem and the modeling of a priori information.

2.1 Canonical decompositions

Let $\mathcal{P}(E)$ be the collection of all subsets of a finite non empty subset E (i.e. the collection of all binary images). Let $\subset$ be the usual inclusion relation on sets. Let X^c be the complementary set of a subset X of E. We know that $(\mathcal{P}(E), \subset)$ is a complete Boolean lattice [13]. The intersection and union of X_1 and $X_2 \in \mathcal{P}(E)$ are, respectively, $X_1 \cap X_2$ and $X_1 \cup X_2$.

The set E is assumed to be an Abelian group with respect to a binary operation denoted by $+$. The zero element of $(E, +)$ is denoted by o.

For any $h \in E$ and $X \subset E$, the set $X + h = \{y \in E : y = x + h,\ x \in X\}$ is called the *translate* of X by h. In particular, $X_o = X$.

A *set operator* is any function defined from $\mathcal{P}(E)$ into itself. A set operator ψ is called *translation invariant* (t.i.) if and only if (iff)

$$\psi(X + h) = \psi(X) + h \quad (X \in \mathcal{P}(E),\ h \in E).$$

The *kernel* $\mathcal{K}(\psi)$ of a t.i. set operator ψ is the subcollection of $\mathcal{P}(E)$ defined by

$$\mathcal{K}(\psi) = \{X \in \mathcal{P}(E) : o \in \psi(X)\}.$$

A subcollection $[A, B]$ of $\mathcal{P}(E)$, with $A \subset B$, is called a *closed interval* iff

$$[A,\ B] = \{X \in \mathcal{P}(E) : A \subset X \subset B\}.$$

The sets A and B are called, respectively, the *left* and *right extremities* of the closed interval.

Let $[A,\ B] \subset \mathcal{P}(E)$. The t.i. set operators $\lambda_{[A,B]}$ defined by

$$\lambda_{[A,B]}(X) = \{x \in E : A + x \subset X \subset B + x\} \quad (X \in \mathcal{P}(E))$$

is called *sup−generator operator characterized* by $[A,\ B]$. A sup−generator operator can be decomposed as the infimum of an erosion and anti−dilation.

A useful property of the sup−generator operators is that they are sufficient to decompose any t.i. operator in standard forms [14].

Let ψ be a t.i. operator and $\mathcal{K}(\psi)$ be its kernel, then

$$\psi(X) = \cup \{\lambda_{[A,B]}(X) : [A,\ B] \subset \mathcal{K}(\psi)\} \quad (X \in \mathcal{P}(E)).$$

This result is known as the *canonical decomposition theorem*.

This representation theorem may lead to computational inefficient representations for most t.i. operators, in the sense that a smaller family of sup−generator operators may be sufficient to represent the same operator.

A closed interval contained in a subcollection $\mathcal{G}$ of $\mathcal{P}(E)$ is called *maximal* if no other interval contained in $\mathcal{G}$ properly contains it. The set $\mathcal{B}(\psi)$ of all the maximal closed intervals contained in $\mathcal{K}(\psi)$ is called the *basis* of ψ.

In fact, the kernel $\mathcal{K}(\psi)$ can be replaced by $\mathcal{B}(\psi)$ in the decomposition formula , that is,

$$\psi(X) = \cup \{\lambda_{[A,B]}(X) : [A,\ B] \in \mathcal{B}(\psi)\} \quad (X \in \mathcal{P}(E)).$$

In practice, the interesting operators are the ones that depend on a local neighborhood. A t.i. operator is called *locally defined within a window* $W \subset E$ iff

$$h \in \psi(X) \Leftrightarrow h \in \psi(X \cap (W + h)),$$

for all $h \in E$ and $X \in \mathcal{P}(E)$.

If ψ is a locally defined t.i. operator within a window W and $[A,\ B] \in \mathcal{B}(\psi)$, then $A,\ B^c \in \mathcal{P}(W)$. In other words, the left extremity and the complement of the right extremity of the interval which characterizes the sup−generator operators used in the decomposition are subsets of the window W.

An important property of t.i. operators is that they are closely related to Boolean functions. Let $\{0, 1\}^{\mathcal{P}(W)}$ denote the set of Boolean functions defined from $\mathcal{P}(W)$ to $\{0,\ 1\}$. For each Boolean function $b \in \{0, 1\}^{\mathcal{P}(W)}$ there exists a corresponding locally defined t.i. operator (within a window W) ψ_b given by

$$\psi_b(X) = \{x \in E : b((X - x) \cap W) = 1\} \quad (X \in \mathcal{P}(E)).$$

Conversely, for each locally defined t.i. operator (within a window W) ψ there exists a corresponding Boolean function $b_\psi \in \{0, 1\}^{\mathcal{P}(W)}$ given by

$$b_\psi(X) = 1 \Leftrightarrow o \in \psi(X) \quad (X \in \mathcal{P}(W)).$$

2.2 Shape Recognition

Let M be a subset of E. A *shape* $\mathcal{S}$ *in* M is a collection of subsets of M. A set $X \in \mathcal{S}$ is called *an object of shape* $\mathcal{S}$.A classical problem in Image Analysis is the problem of shape recognition.

Let I be a set of indices. Given a collection of shapes $\{\mathcal{S}_i : i \in I\}$, such that $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for $i \neq j$, $i,j \in I$, and a set X, such that $X \in \cup\{\mathcal{S}_i : i \in I\}$, of unknown shape, what is the shape of X?

A collection $\{\psi_i : i \in I\}$ of set operators can be used to solve this problem. A set operator ψ_i indicates if X is of shape $\mathcal{S}_i$ or not, respectively, if it satisfies the following property

$$\psi_i(X) \neq \emptyset, \ \forall X \in \mathcal{S}_i$$

and

$$\psi_i(X) = \emptyset, \forall X \in \cup\{\mathcal{S}_j : j \in I, \ j \neq i\}.$$

The operator ψ_i is called the *marker of the shape* $\mathcal{S}_i$.

Let W and X be subsets of M. The *model of* X *through* W is the collection given by

$$X_W = \{W + h \cap X, \ h \in E\}.$$

A shape recognition problem is said to be of *dimension* W If, for all $i \in I$, there exists $\mathcal{M}_{\mathcal{O}_i} \subset \mathcal{P}(W)$, $\mathcal{M}_{\mathcal{O}_i} \neq \{\emptyset\}$, such that

$$\forall X \in \mathcal{S}_i, \ \mathcal{M}_{\mathcal{O}_i} \subset X_W \text{ and } \forall j \in I, \ i \neq j, \ \forall Y \in \mathcal{S}_j, \mathcal{M}_{\mathcal{O}_i} \subset Y_W .$$

This condition implies that there exists a collection of operators locally defined within the window W that can solve the shape recognition problem.

2.3 Modeling of a priori information

In the application of set operators to solve image analysis problems there exist two types of a priori information available: the knowledge about the images of the domain (i.e., the context) and the knowledge about the tasks to be executed (i.e., the constraints).

We will model the *context* as a subset $\mathcal{A}$ of $\mathcal{P}(E)$. In practice, we will be interested in studying the family of set operators defined from $\mathcal{A}$ to $\mathcal{P}(E)$. These operators can be interpreted as an equivalence class on the family of set operators defined from $\mathcal{P}(E)$ to $\mathcal{P}(E)$, that is,

$$\psi_1 \equiv \psi_2 \Leftrightarrow \psi_1(X) = \psi_2(X), \ \forall X \in \mathcal{A}.$$

Thus, the introduction of a context implies in the reduction of the number of different set operators available to be chosen by the analyst. In the problem of shape recognition, for example, the context is the set $\mathcal{A} = \cup\{\mathcal{S}_i : i \in I\}$.

The knowledge about the tasks to be executed on images can be modeled by constraints imposed on the set operators. For example, based on the knowledge of a given task the analyst may be able to state that the desired operator is locally defined within a window W, monotone (i.e., $X \subset Y \Rightarrow \psi(X) \subset \psi(Y), \forall X, \ Y \in \mathcal{P}(E)$), anti−extensive (i.e., $\psi(X) \subset X, \ \forall X \in \mathcal{P}(E)$), etc.

Thus, the introduction of constraints also implies in the reduction of the number of interesting set operators available to be chosen by the analyst. In the problem of shape recognition of dimension W, for example, two constraints are that the marker is locally defined within the window W and anti−extensive.

Hence, the introduction of a context and the definition of constraints on set operators imply in the reduction of the number of different and interesting set operators available to be chosen by the analyst. In real world problems, we can count on these two properties to reduce the research space for set operators.

3. PAC LEARNING

Computational Learning Theory [15] [16] is one of the first attempts to construct a mathematical model for the cognitive process of learning concepts. We understand *concept* as a subset of objects in a predefined domain. An *example* of a concept is an object from the domain together with a label indicating whether the object belongs to the concept. If the object belongs to the concept, it is a *positive example*, otherwise it is a *negative example*. *Concept learning* is the process by which a *learner* constructs a good approximation to an unknown concept, given a small number of *examples* and some prior information about the concept to be learned. In the following, we formalize these ideas.

Let Σ be a set, called the *alphabet* to describe examples. In this paper, Σ will be the *Boolean alphabet* $\{0, 1\}$. We denote the set of $n-$tuples of elements of Σ by Σ^n. Let X be a subset of Σ^n. We define a *concept*, over the alphabet Σ, as a function $c : X \to \{0, 1\}$.

The set X will be referred to as the *example space*, and its members as *examples*. An example $y \in X$ for which $c(y) = 1$ is known as a *positive example*, and an example for which $c(y) = 0$ is known as a *negative example*. So, provided that the domain is known, c determines, and is determined by, its set of positive examples. Sometimes it is helpful to think of a concept as a set in that way.

The set of all possible concepts to be learned will be referred to as the *hypothesis space* and denoted by H. The concept $t \in H$ to be determined is called the *target concept*. The problem is to find a concept $h \in H$, called the *hypothesis*, which is a good approximation for t.

A *sample of length m* is just a sequence of m examples, that is, an $m-$tuple $x = \{x_1, x_2, ..., x_m\}$ in X^m. The sequence may contain the same value more than once. A *training sample* s is an element of $\{X^m \times \{0, 1\}\}$, that is, $s = ((x_1, b_1), (x_2, b_2), \ldots, (x_m, b_m))$, where the x_i are examples and the b_i are bits. The value of b_i is given by a *teacher* and specifies whether x_i is a positive or a negative example. There are no contradictory labels, so that if $x_i = x_j$ then $b_i = b_j$.

A *learning algorithm* is simply a function L which assigns to any training sample s for a target concept $t \in H$ a hypothesis $h \in H$. We write $h = L(s)$.

Let μ be a *probability distribution* (or *probability measure*) on X. Given a target concept $t \in H$, we define the *error of any hypothesis* $h \in H$, with respect to t, as the probability of the event $h(x) \neq t(x)$, that is, .

$$er_\mu(h, t) = \mu\{x \in X : h(x) \neq t(x)\}.$$

When a given set X is provided with the structure of a probability space, the product set X^m inherits this structure from X. It is sufficient to remark that the construction allows us to regard the components of the $m-$tuple $(x_1, x_2, ..., x_m)$ as "independent" variables, each distributed according to the probability distribution μ on X. The corresponding distribution on X^m is denoted by μ^m.

Let $S(m, t)$ denote the set of training samples of length m for a given target concept t, where the examples are drawn from an example space X. As there is a bijection $\phi : X^m \to S(m, t)$ for which $\phi(x) = s$, the following equality hold

$$\mu^m\{s \in S(m, t) : s \text{ has property } P \} = \mu^m\{x \in X^m : \phi(x) \in S(m, t) \text{ has property } P \}.$$

We say that the algorithm L is a *probably approximately correct* (*PAC*) learning algorithm for the hypothesis space H if, given two real numbers ε and δ $(0 < \varepsilon, \delta < 1)$, then there is a positive integer $m_0 = m_0(\varepsilon, \delta)$ such that for any target concept $t \in H$, and for any distribution μ on X, whenever $m \geq m_0$,

$$\mu^m\{s \in S(m,t) : er_\mu(L(s)) < \varepsilon\} > 1 - \delta.$$

The function $m_0 = m_0(\varepsilon, \delta)$ is called *sample complexity of the problem.*

A learning algorithm L for H *is consistent if, given any training sample* s *for a target concept* $t \in H$, the output hypothesis agrees with t on the examples in s, that is, $h(x_i) = t(x_i)$ $(1 \leq i \leq m)$. For a given $s \in S(m,t)$, we denote by $H[s]$ the set of all hypotheses consistent with s:

$$H[s] = \{h \in H : h(x_i) = t(x_i)\ (1 \leq i \leq m)\}.$$

Given $\varepsilon \in (0,1)$ the set

$$B_\varepsilon = \{h \in H : er_\mu(h) \geq \varepsilon\}$$

is called the set of $\varepsilon-$*bad hypothesis for* t.

We say that the hypothesis space H is potentially learnable if, given two real numbers ε and δ $(0 < \varepsilon, \delta < 1)$, there is a positive integer $m_0 = m_0(\varepsilon, \delta)$ such that, whenever $m \geq m_0$

$$\mu^m\{s \in S(m,t) : H[s] \cap B_\varepsilon = \emptyset\} > 1 - \delta$$

for any probability distribution μ on X and any $t \in H$.

The following statements are proved to hold [16]: If H *is potentially learnable*, and L is a consistent learning algorithm for H, then L is PAC; any finite hypothesis space is potentially learnable. Hence, any consistent learning algorithm applied to a finite hypothesis space is PAC.

Let $|H|$ denote the cardinality of the set H. According to Anthony and Biggs [16], the sample complexity of a PAC learning algorithm is

$$m_0(\varepsilon, \delta) = \lceil (1/\varepsilon)\ \ln\ (|H|/\delta) \rceil,$$

for any distribution μ and with the only restriction that the hypothesis space H be finite.

Here, we will use the PAC learning theory to learn locally t.i. set operators within a finite window W. As this hypothesis space is finite, all the consistent learning algorithms applied to it are PAC and, consequently, the last expression can be used to compute the minimum number of examples required (i.e., the sample complexity) for those algorithms. Usually, in this expression the value $|H|$ is much larger than the other values and the way for getting reasonable values for $m_0(\varepsilon,\ \delta)$ is to state properly the context and add constraints.

4. ALGORITHMS FOR PAC LEARNING

In this section, we will present two consistent learning algorithms for Boolean functions. The first one is based on the classical algorithm of Quine−McCluskey for the simplification of Boolean functions and the second one is based on the incremental splitting of intervals.

4.1 Quine−McCluskey algorithm

Motivated by the necessity of designing digital circuits, the problem of minimization of Boolean functions has been studied for a long time. A classical computational algorithm for the minimization of Boolean functions was created by W. V. Quine and E. J. McCluskey in the fifties [17,18].

The n variables of a Boolean function can be represented as points in a $n-$space. The collection of all 2^n possible points will be said to form the *vertices of an* $n-$*cube*. The *cubical representation* of *a function of* n *variables* consists of the set

of vertices of the $n-$cube for which the function has value 1. These vertices are called the *0−cubes of the function*. Two $0-$cubes are said to form a *1−cube* if they differ in only one coordinate, two $1-$cubes are said to form a *2−cube* if they differ in only one coordinate and so on.

When all the vertices ($0-$cubes) of a $k-$cube are in the set of vertices making up a larger $k'-$cube ($k < k'$), we will say that the smaller cube is contained, or *covered*, by the larger cube.

The Quine−McCluskey method is composed of two steps: $i-$ all $1-$cubes that cover a $0-$cube are created and all the $0-$cubes covered by the $1-$cubes created are eliminated, all $2-$cubes that cover a $1-$cube are created and all the $1-$cubes covered by the $2-$cubes created are eliminated and so on; $ii-$ a minimal combination (involving a minimal number of cubes) of the remaining cubes is selected to realize the function.

When there are *don't cares* in the definition of the function, in order to optimize the minimization process, they are taken as 1 value vertices in the first step of the method and are ignored (i.e., the combination of cubes selected does not cover necessarily the *don't cares*) in the second step.

The Quine−McCluskey method applied to a function with *don't cares,* where the defined values of the function are a sequence of given examples (after the elimination of eventual repetitions), can be seen as a component of a consistent learning algorithm. The critical problem with the use of this algorithm for learning Boolean functions is that it uses a substantial amount of memory space in the initial states, since it must store the positive examples as well as the other unknown cases (taken as 1 value).

4.2 Incremental Splitting of Intervals algorithm

This algorithm explores the fact that the domain of the Boolean functions $\mathcal{P}(W)$ is a complete Boolean lattice and that any subset of a complete Boolean lattice can be represented by an union of closed intervals [14]. Here, a Boolean function f will also be represented by a collection of maximal closed intervals B, such that

$$f(X) = 1 \Leftrightarrow \exists [A,\ B] \in B : X \in [A,\ B].$$

The states of this algorithm are the Boolean functions learned and they are indexed by the sequence of input examples (without repetitions). For each new example $(X_{i+1},\ b_{i+1})$ that arrives, the current Boolean function f_i learned so far is updated to f_{i+1}.The initial state is the constant function equal to 1.

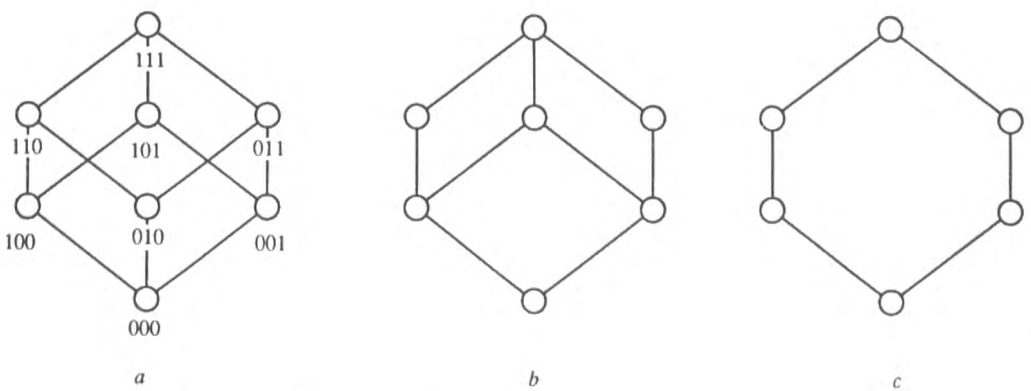


Fig. 1 − a) Initial state. b) First state. c) Second state.

The mechanism of updating the learned function is the following: *i*– when a negative example (X, 0) arrives, it looks for the closed intervals that contain X, eliminates X from these intervals and represent each of the resulting collection of sets by an union of maximal closed intervals; *ii*– when a positive example (X, 1) arrives nothing is done. When the learning process is finished , a minimal combination of closed intervals is chosen (as in the step *ii* of the Quine–McCluskey algorithm).

Figure 1 illustrates the *step i* of the algorithm. In this example, W is a set with three points. In figure 1a, we have the initial state. In figure 1b, we have the first state, after the input of the example (010, 0). In figure 1b, we have the second state after the input of the example (101, 0). Note that the elimination of an element of a closed interval represented by n Boolean variables induces the creation of n closed intervals represented by $n - 1$ Boolean variables.

The arrival of a negative example induces the split of a closed interval or of a set of closed intervals. Figure 2 shows the tree of closed intervals that is built when the example of figure 1 is processed.

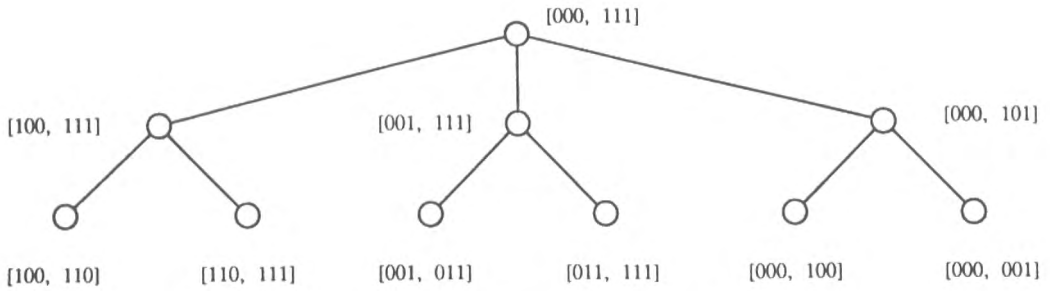


Fig. 2 – Tree of closed intervals.

Other variants of this algorithm can be built just by modifying the way of dealing with the positive examples. A first variant is built by eliminating all the intervals that do not cover at least a positive example. A second variant is built by creating intervals just until all the positive examples are covered. A third variant is built by computing a minimal number of closed intervals that is sufficient to cover all positive examples, after the creation of the new collection of closed intervals. These actions should be performed as the step *ii* of the algorithm. All these variants are equivalent under the point of view of the PAC learning theory, however they have important differences in terms of efficiency.

4.3 Quine–McCluskey x Incremental Splitting of Intervals

The basic difference between the Quine–McCluskey and the Incremental Splitting of Intervals algorithm is that the first one always gives an optimal solution to the problem of learning Boolean functions (i.e., gives a function that is representable by a number of maximal closed intervals that is less than or equal to the number of maximal closed intervals required to represent the other equivalent functions), while the second one may give suboptimal solutions.

Another important difference between these two algorithms is that the time and storage space complexities of the Quine–McCluskey algorithm is a function of the size of the sequence of positive input examples (without repetitions) and of the number of unknown cases, while the complexities of the Incremental Splitting of Intervals algorithm is a function just of the size of the sequence of input examples (without repetitions).

The standard Incremental Splitting of Intervals algorithm gives the largest consistent hypothesis (in the sense of the partial ordering for functions, inherited of the fact that $0 < 1$) for the input data. Its main drawback, compared with the three variants, is that it uses more storage space. Yet, the fact of creating too many intervals affects its time efficiency. The three variants constitute attempts to minimize these problems.

We have implemented the Quine−McCluskey algorithm as well as the Incremental Splitting of Intervals algorithms. Table 1 shows some experimental results obtained by the application of these algorithms to the problem of learning Boolean functions. This table gives the performance of these algorithms in terms of storage space, running time and number of maximal closed intervals used to represent the learned function, as a function of the cardinality of the window W, the number of positive examples and the number of negative examples. These experiments were performed on a SPARC station 2.

The columns of Table 1 correspond, respectively, to the cardinality of the function domain, the algorithm name, the number of positive examples given, the number of negative examples given, the storage space, the running time of the algorithm and the number of maximal closed intervals used to represent the learned function. The Quine−McCluskey algorithm is denoted *QM* and the incremental splitting of intervals algorithms are denoted *ISI−N*, where *N* indicates the variant of the algorithm used. *ISI−0* is the standard algorithm, *ISI−1* is the first variant and so on.

Table 1. Performance evaluation of the algorithms

$\|W\|$ (number)	Algorithm (name)	Positive Ex. (number)	Negative Ex. (number)	Storage (*bytes*)	Run Time (*seconds*)	Closed Interv. (number)
12	QM	622	764	1.722.888	3.425	63
12	ISI–0	622	764	450.560	25	63
12	ISI–1	622	764	393.208	24	63
12	ISI–2	622	764	55.672	5	94
12	ISI–3	622	764	95.696	11	69
16	QM	17	511	–	>78.118	–
16	ISI–2	17	511	25.512	1	14
42	ISI–2	180	7214	304.376	34	167
42	ISI–3	180	7214	304.376	69	144
42	ISI–2	2025	5369	304.368	6.107	1478
49	ISI–2	5194	8772	620.352	56.858	3185

The experimental results show that the ISI−2 is the best between the ISI algorithms in terms of time and storage space efficiency, but its learned functions usually need larger numbers of maximal closed intervals to be represented. ISI−O is the worst between the ISI algorithms in terms of time and storage space efficiency,, but its learned functions usually need smaller numbers of closed intervals to be represented. ISI−1 and ISI−3 have intermediate performances.

These experiments indicate that the use of the ISI algorithms permits to approach the problem of learning Boolean functions with large number of variables, what is not possible using the QM algorithm.

5. APPLICATION EXAMPLES

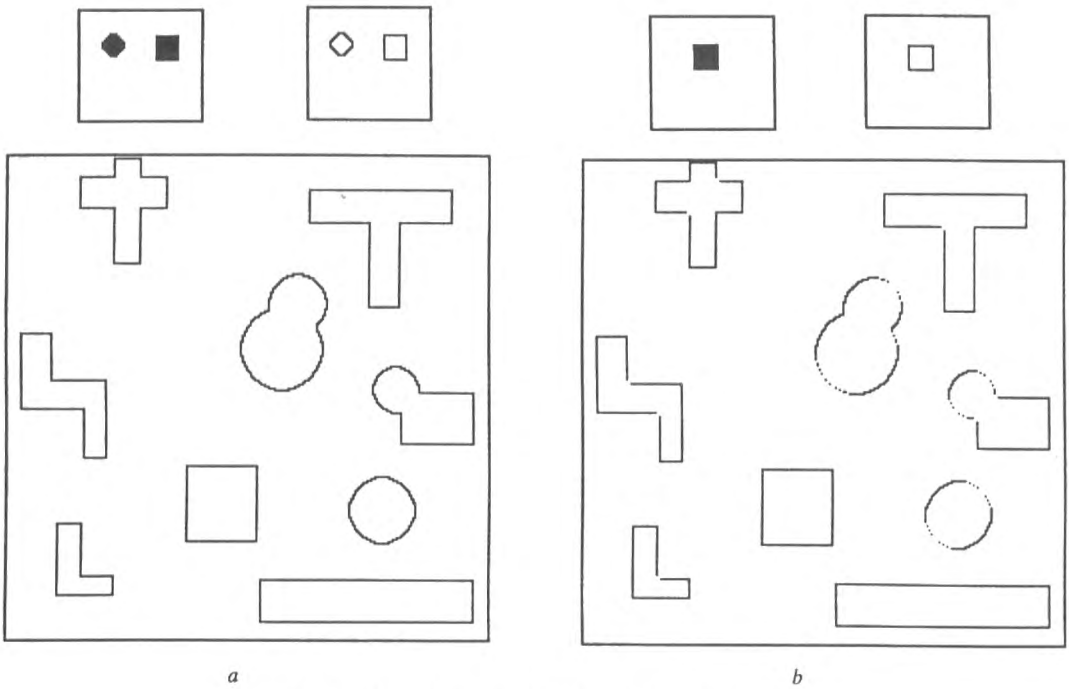
In this section, we are going to present two application examples: extraction of internal edges from the domain of disks and rectangles; recognition of shapes in the domain of printed characters.

5.1 Internal edges from the domain of disks and squares

The goal of this experiment is to learn a set operator that can extract internal edges in a domain whose images are created by the superposition of disks and rectangles. Let take as constraint that the hypothesis space is the collection of set operators locally defined within the 3x3 window, centered at the origin, that is, $\mathcal{A} \subset \mathcal{P}(\{-1, ,0, 1\}^2)$. Let us take $\varepsilon = \delta = 0.02$. Experimentally we verified that $|\mathcal{A}| = 66$ (i.e., we could find 66 different configurations given the constraints above). Hence, $m_0(\varepsilon, \delta) = (1/0.02).\ln((1/0.02).2^{66}) = 2488$. Figure 3 shows two experiments, where the size of the sequence of examples used was $m = 3600$. In each experiment, the sequence of input examples is built from the

pair of small images (60 x 60). In the first experiment, the number of distinct input examples was 46, while in the second one it was 26.

The collection of intervals that describes the function learned in the first experiment is: $A_1 = A_2 = A_3 = A_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}$, $B_1^c = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{0} & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $B_2^c = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{0} & 0 \\ 1 & 0 & 0 \end{bmatrix}$, $B_3^c = \begin{bmatrix} 0 & 0 & 1 \\ 0 & \mathbf{0} & 0 \\ 0 & 0 & 0 \end{bmatrix}$ e $B_4^c = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathbf{0} & 0 \\ 0 & 0 & 0 \end{bmatrix}$, where the boldfaced element denotes the origin.



a *b*

Fig. 3 – a) First experiment. b) Second experiment.

Observe that the visual result of the first experiment, figure 3a, is slightly better than the visual result of the second experiment, figure 3b. However, both experiments have a precision better than 2%.

5.2 Shape recognition from the domain of printed characters

The goal of this experiment is to learn a set operator that can recognize the digit 4 from an image containing the 10 digits. In this case, the shape associated to the digit 4 is the collection of ten different subsets, each one representing the digit 4 in one of the ten different alphabets that constitute the context. Let us take as constraint that the hypothesis space is the collection of set operators locally defined within a rectangle window. Let us built the sequence of input examples taking into account all the objects in the domain. Figure 4a (without the edges of disks and rectangles) shows the input image and figure 4b shows the output image used to build the sequence of examples. A configuration of the domain was associated to label 1 if and only if it belongs to one object that represents the digit 4 and did not belong to any other object

of the domain. In this way, the algorithm could never learn a function that marked a character that was not of the target shape.

Taken as window W the 3x3 rectangle, centered at the origin, the operator learned recognizes solely the digits indicated by a circle in figure 4a. The collection of intervals which describe this function is: $A_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$, $B_1^c = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$, $A_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, $B_2^c = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, $A_3 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$, $B_3^c = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $A_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $B_4^c = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$. Taken as window W a 3x4 rectangle, with the origin in the position (2, 2) of the matrix that describes the window, the operator learned recognizes the digits indicated by a circle or a rectangle. In this case, the number of intervals necessary to describe the function is *11*. Taken as window a 3x5 rectangle, centered at the origin, the learned operator recognizes all the objects representing the digit 4. In this case, the number of intervals necessary to describe the function is 21.

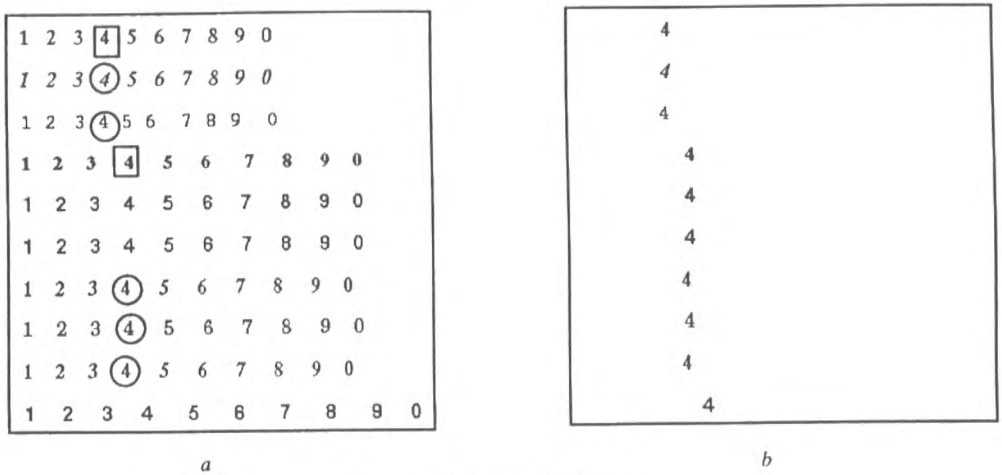


Fig. 4 – a) The ten alphabets. b) The digits 4.

6. CONCLUSION

We have presented some efficient and consistent learning algorithms (i.e., the Incremental Splitting of Intervals and its variants) that can be applied for the learning of locally defined t.i. operators. The algorithm proposed showed to be useful in practice, for example, to the design of marker operators for shape recognition. These algorithms could still be applied to the classical problem of designing combinatorial logic circuits with *don't cares*, with a time and storage space performance much better than the classical Quine–McCluskey algorithm.

An important characteristic of the proposed algorithms is that their storage space and time complexity are a function of the number of distinct elements in the sequence of input examples. This opens a way to deal with the learning of locally defined operators within large windows, at least when the number of distinct input examples is not too large.

It seems very useful for practical applications to study how the proposed algorithms could be simplified under a priori constraints. It would be important to find a systematic way of dealing with constraints in the algorithms, without introducing undesired drawbacks as the necessity of implementing a new algorithm for each combination of constraints.

A point that should be investigated in the future is the estimation of the size of the hypothesis space from the sequence of input examples available. The theory of PAC learning itself could be used to approach this problem, since the domain $\mathcal{A} \subset \mathcal{P}(W)$ can be understood as a concept in the hypothesis space $\mathcal{P}(W)$.

The ideas presented here could also be extended to the learning of locally defined set operators that perform optimal or suboptimal filtering for the restoration of images. In this case, the theoretical bases would be the extension of the PAC learning theory for the case where the professor can make some mistakes [19].

Finally, we believe that our initial experimental results indicate that we should investigate deeper the yet practically unexplored field of automatic programming of morphological machines by PAC learning techniques.

7. ACKNOWLEDGMENTS

The authors have received partial financial support from Olivetti do Brasil, CNPq (Conselho Nacional de Desenvolvimento Cientifico e Tecnológico), grants PROTEM–CC–ANIMOMAT and PROTEM–CC–TCPAC, and FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo), grants 91/3532–2 and 93/0603–01.

8. REFERENCES

1. J. Serra, *Image Analysis and Mathematical Morphology. Volume 2: Theoretical Advances*, Academic Press, London, 1988.

2. G. Matheron, *Random Sets and Integral Geometry*, John Wiley, New York, 1975.

3. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, New York, 1982.

4. J. Barrera and G. Banon, Expressiveness of the morphological language, *Image Algebra and Morphological Image Processing III*, pp. 264–275, SPIE, San Diego, California, July 1992.

5. J. C. Klein and J. Serra, "The texture analyser," *Journal of Microscopy*, vol. 95, pt. 2, pp. 343–356, Apr. 1972.

6. P. Husson, J. P. Derutin, P. Bonton and J. Gallice, "Elementary processor with mathematical morphology implemented in VLSI and intented for systolic architecture," *Signal Processing IV: Theories and Aplication*, EURASIP, pp. 1561–1564, 1988.

7. J. C. Klein and R. Peyhard, "PIMM1, An image processing ASIC based on mathematical morphology," *IEEE's A ASIC Seminar and Exhibit*, Rochester, New York, pp. 25–28, Sep. 1989.

8. K. S. Huang, B. K. Jenkins and A. A. Sawchuk, "Binary image algebra and optical cellular logic processor design," *Computer Vision, Graphics and Image Processing*, vol. 45, pp. 295–345, 1989.

9. B. Läy, "Description des programmes du logiciel morpholog," *CGMM Rapport interne N–904*, Fontainebleau, 1984.

10. J. Barrera; G. Banon; R. Lotufo A Mathematical Morphology Toolbox for the KHOROS System, *Image algebra and Morphological Image Processing V*, SPIE, Vol. 2300, pp. 241–252, San Diego, California, July 1994.

11. C. Gratin, "Élaboration d'un logiciel de morphologie mathématique sur micro–ordinateur," *CMM Rapport interne*, Fontainebleau, 1989.

12. J. Barrera, F. S. C. da Silva, G. J. F. Banon Automatic Programming of Binary Morphological Machines. *Image Algebra and Morphological Image Processing V*, SPIE, Vol. 2300, pp. 229–240, San Diego, California, July 1994.

13. G. Birkhoff, *Lattice Theory*, American Mathematical Society, Providence, Rhode Island, 1967.

14. G. J. F. Banon and J. Barrera, "Minimal representations for translation invariant set mappings by mathematical morphology," vol. 51, no. 6, *SIAM Journal of Applied Mathematics*, pp. 1782–1798, Dec. 1991.

15. L. Valiant A Theory of the Learnable, *Comm. ACM*, v. 27, pp. 1134–1142, 1984.

16. M. Anthony and N. Biggs *Computacional Learning Theory. An introduction*, Cambridge University Press, 1992.

17. Quine, W. V. The problem of simplifying truth functions, *Am. Math. Monthly*, 59, No. 8, 521–531, Oct. 1952.

18. McCluskey, E. J. Minimization of Boolean functions, *Bell System Tech. J.*, 35, No. 5, 1417–1444, Nov. 1956.

19. D. Haussler Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications. *Information and Computation*, 100, pp. 78–150, 1992.

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 1992 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail(mac@ime.usp.br).

J.Z. Gonçalves, Arnaldo Mandel
COMMUTATIVITY THEOREMS FOR DIVISION RINGS AND DOMAINS
RT-MAC-9201, Janeiro 1992, 12 pp.

J. Sakarovitch
THE "LAST" DECISION PROBLEM FOR RATIONAL TRACE LANGUAGES
RT-MAC 9202, Abril 1992, 20 pp.

Valdemar W. Setzer, Fábio Henrique Carvalheiro
ALGOTRITMOS E SUA ANÁLISE (UMA INTRODUÇÃO DIDÁTICA)
RT-MAC 9203, Agosto 1992, 19 pp.

Claudio Santos Pinhanez
UM SIMULADOR DE SUBSUMPTION ARCHITECTURES
RT-MAC-9204, Outubro 1992, 18 pp.

Julio M. Stern
REGIONALIZAÇÃO DA MATRIZ PARA O ESTADO DE SÃO PAULO
RT-MAC-9205, Julho 1992, 14 pp.

Imre Simon
THE PRODUCT OF RATIONAL LANGUAGES
RT-MAC-9301, Maio 1993, 18 pp.

Flávio Soares C. da Silva
AUTOMATED REASONING WITH UNCERTAINTIES
RT-MAC-9302, Maio 1993, 25 pp.

Flávio Soares C. da Silva
ON PROOF-AND MODEL-BASED TECHNIQUES FOR REASONING WITH UNCERTAINTY
RT-MAC-9303, Maio 1993, 11 pp.

Carlos Humes Jr., Leônidas de O.Brandão, Manuel Pera Garcia
A MIXED DYNAMICS APPROACH FOR LINEAR CORRIDOR POLICIES
(A REVISITATION OF DYNAMIC SETUP SCHEDULING AND FLOW CONTROL IN MANUFACTURING SYSTEMS)
RT-MAC-9304, Junho 1993, 25 pp.

Ana Flora P.C.Humes e Carlos Humes Jr.
STABILITY OF CLEARING OPEN LOOP POLICIES IN MANUFACTURING SYSTEMS (Revised Version)
RT-MAC-9305, Julho 1993, 31 pp.

Maria Angela M.C. Gurgel e Yoshiko Wakabayashi
THE COMPLETE PRE-ORDER POLYTOPE: FACETS AND SEPARATION PROBLEM
RT-MAC-9306, Julho 1993, 29 pp.

Tito Homem de Mello e Carlos Humes Jr.
SOME STABILITY CONDITIONS FOR FLEXIBLE MANUFACTURING SYSTEMS WITH NO SET-UP TIMES
RT-MAC-9307, Julho de 1993, 26 pp.

Carlos Humes Jr. e Tito Homem de Mello
A NECESSARY AND SUFFICIENT CONDITION FOR THE EXISTENCE OF ANALYTIC CENTERS IN PATH FOLLOWING METHODS FOR LINEAR PROGRAMMING
RT-MAC-9308, Agosto de 1993

Flavio S. Corrêa da Silva
AN ALGEBRAIC VIEW OF COMBINATION RULES
RT-MAC-9401, Janeiro de 1994, 10 pp.

Flavio S. Corrêa da Silva e Junior Barrera
AUTOMATING THE GENERATION OF PROCEDURES TO ANALYSE BINARY IMAGES
RT-MAC-9402, Janeiro de 1994, 13 pp.

Junior Barrera, Gerald Jean Francis Banon e Roberto de Alencar Lotufo
A MATHEMATICAL MORPHOLOGY TOOLBOX FOR THE KHOROS SYSTEM
RT-MAC-9403, Janeiro de 1994, 28 pp.

Flavio S. Corrêa da Silva
ON THE RELATIONS BETWEEN INCIDENCE CALCULUS AND FAGIN-HALPERN STRUCTURES
RT-MAC-9404, abril de 1994, 11 pp.

Junior Barrera; Flávio Soares Corrêa da Silva e Gerald Jean Francis Banon
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES
RT-MAC-9405, abril de 1994, 15 pp.

Valdemar W. Setzer; Cri. ina G. Fernandes; Wania Gomes Pedrosa e Flavio Hirata
UM GERADOR DE ANALISADORES SINTÁTICOS PARA GRAFOS SINTÁTICOS SIMPLES
RT-MAC-9406, abril de 1994, 16 pp.

Siang W. Song
TOWARDS A SIMPLE CONSTRUCTION METHOD FOR HAMILTONIAN DECOMPOSITION OF THE HYPERCUBE
RT-MAC-9407, maio de 1994, 13 pp.

Julio M. Stern
MODELOS MATEMATICOS PARA FORMAÇÃO DE PORTFÓLIOS
RT-MAC-9408, maio de 1994, 50 pp.

Imre Simon
STRING MATCHING ALGORITHMS AND AUTOMATA
RT-MAC-9409, maio de 1994, 14 pp.

Valdemar W. Setzer e Andrea Zisman
*CONCURRENCY CONTROL FOR ACCESSING AND COMPACTING B-TREES**
RT-MAC-9410, junho de 1994, 21 pp.

Renata Wassermann e Flávio S. Corrêa da Silva
TOWARDS EFFICIENT MODELLING OF DISTRIBUTED KNOWLEDGE USING EQUATIONAL AND ORDER-SORTED LOGIC
RT-MAC-9411, junho de 1994, 15 pp.

Jair M. Abe, Flávio S. Corrêa da Silva e Marcio Rillo
PARACONSISTENT LOGICS IN ARTIFICIAL INTELLIGENCE AND ROBOTICS.
RT-MAC-9412, junho de 1994, 14 pp.

Flávio S. Corrêa da Silva, Daniela V. Carbogim
A SYSTEM FOR REASONING WITH FUZZY PREDICATES
RT-MAC-9413, junho de 1994, 22 pp.

Flávio S. Corrêa da Silva, Jair M. Abe, Marcio Rillo
MODELING PARACONSISTENT KNOWLEDGE IN DISTRIBUTED SYSTEMS
RT-MAC-9414, julho de 1994, 12 pp.

Nami Kobayashi
THE CLOSURE UNDER DIVISION AND A CHARACTERIZATION OF THE RECOGNIZABLE Z-SUBSETS
RT-MAC-9415, julho de 1994, 29pp.

Flávio K. Miyazawa e Yoshiko Wakabayashi
AN ALGORITHM FOR THE THREE-DIMENSIONAL PACKING PROBLEM WITH ASYMPTOTIC PERFORMANCE ANALYSIS
RT-MAC-9416, novembro de 1994, 30 pp.

Thomaz I. Seidman e Carlos Humes Jr.
SOME KANBAN-CONTROLLED MANUFACTURING SYSTEMS: A FIRST STABILITY ANALYSIS
RT-MAC-9501, janeiro de 1995, 19 pp.

C.Humes Jr. and A.F.P.C. Humes
STABILIZATION IN FMS BY QUASI- PERIODIC POLICIES
RT-MAC-9502, março de 1995, 31 pp.

Fabio Kon e Arnaldo Mandel
SODA: A LEASE-BASED CONSISTENT DISTRIBUTED FILE SYSTEM
RT-MAC-9503, março de 1995, 18 pp.

Junior Barrera, Nina Sumiko Tomita, Flávio Soares C. Silva, Routo Terada
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES BY PAC LEARNING
RT-MAC-9504, abril de 1995,16 pp.