A Self-Organizing System for Image Recognition

P. Bergmann, J. Keller and W. J. Paul - University Saarbrücken, Germany

Abstract

We describe a System which constructs image descriptions. An image description is a program reproducing this image. Hence the system constructs programs, which produce images or programs encoding strategies for program construction. In the beginning programs are guessed. Later on when the system has already found some programs, new programs are formed out of existing programs. The system evaluates a program by comparing the produced image with the input image. The memory of the system contains all found and well evaluated programs, so that new programs can be constructed using old ones beeing useful in the past.

Keywords: Learning, Self-organization

1 Introduction

Building theories starts with observating phenomena in the surrounding world. The occurence of regularities makes us assume the existence of general laws behind the phenomena. We try to predict according to our level of knowledge, which phenomenon will occur the next. If a prediction is always or in most of the cases right, we will take this as a confirmation for the correctness of our theory. Hence, the first criterion for the qualitity of a theory is, how often it can be used for the prediction of events it is supposed to describe. But how can two theories describing the same phenomena be compared? A first approach to this question was supplied by Occam. The principle of Occam's razor is that from two theories describing the same set of events, the simpler one is the one to be chosen. The notion of simplicity is easy to understand in an intuitive sense, but not formally defined. An important step towards a formal definition of simplicity was done independently by Kolmogorow [1] and Solomonoff [2]. For Kolmogoroff, a theory is only a set of descriptions for observations. A mesure for the quality of the theory is the length of the descriptions. A theory is non-trivial if the descriptions are shorter than the observations. described by it. Formally, Kolmogoroff defines observations as strings over the alphabet {0,1} and descriptions as programs for universal Turing machines.

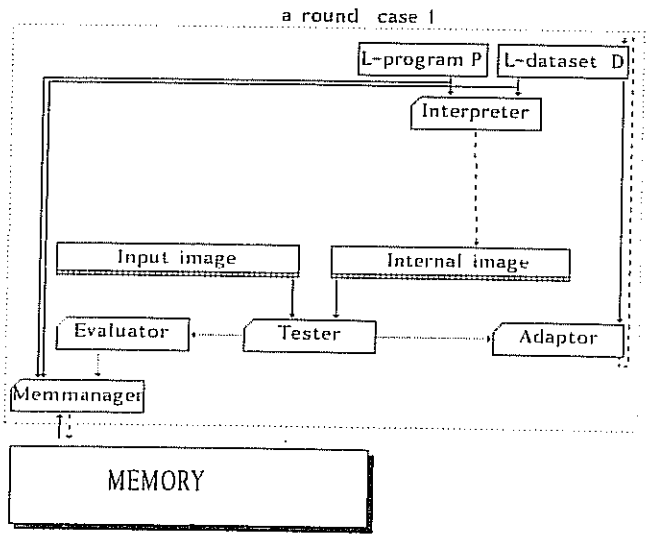
It is our opinion that constructing a theory and understanding are closely related concepts. For this reason, understanding an input image in our system is the ability to construct a short program reproducing the input image. For that, the system constructs small programs, which reproduce (parts of) the input image and hence are encoded images, or which are coded strategies for program generation. In the beginning the programs are guessed. Later on when the system has already learnt some descriptions, new programs are formed out of existing programs. The memory of the system is not simply a list of actual programs found. It rather keeps book of certain relations between the programs. These relations are influenced by the order in which the input images are shown to the system. The memory organizes itself. The conception of the memory allows, as we will point out, recognition- and association-like effects.

2 The architecture of the system

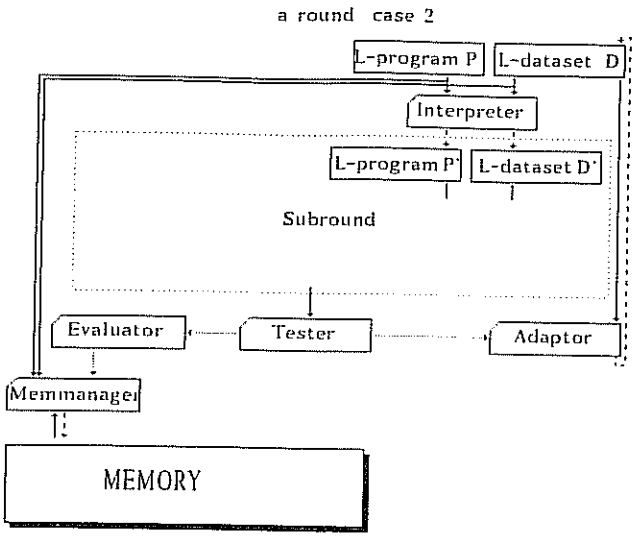
The system works in rounds. During a round the input image is not changed. At the beginning of the round a (short) program is given which is either an image description (case 1) or the description of a strategy for generating or modifying a program (case 2). The system interprets the program. In the first case an internal picture is generated. In the second one a new program is generated and a subround is started. The adjustment of the data-values (describing e.g. height, size or position of the object the data-value is associated with) is achieved by an adaption process (see section 4). The decision to stop the adaption process (and hence the round) is made by a tester. The adaption process is terminated, if either a fixed time has passed, or no further improvement is possible or if the input picture changes. Finally the program's quality is determined by the evaluator (see section 5) and if it is good enough, the program is added to the memory (see section 6). The system is started with the strategy "guess a program".

The number of programs of variable length k grows exponentially in k. Hence, it is important to choose a powerful description language that allows very short descriptions for simple images (i.e. primitive geometric objects). This allows a reasonable time for finding a description for such a simple image by guessing.

The number of programs of fixed length k grows polynomially in the cardinality of the set of instructions. For this reason, it is convenient to choose a language with a small instruction set.



1. generate an internal image
2. evaluate the internal image
3. modificate Dataset D and iterate step 1 and 2 until stop from tester
4. evaluate P and accept (add to the memory) or reject P

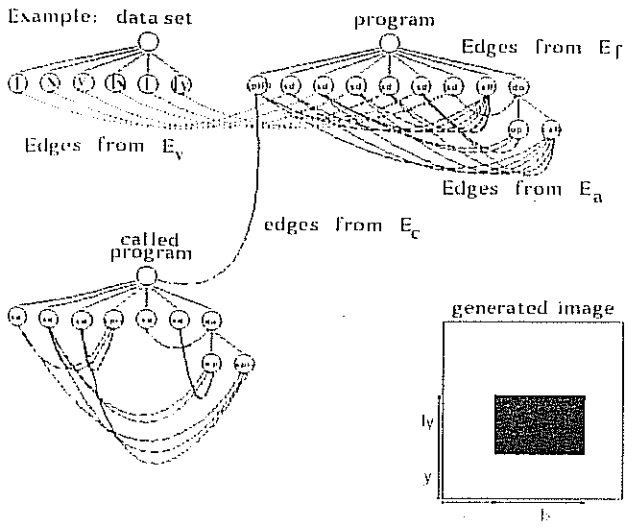


1. generate a L-program P' and an empty L-Datenset D'
2. start a subround
3. modificate Dataset D and iterate step 1 and 2 until stop from tester
4. evaluate P and accept (add to the memory) or reject P

3 The Language L

L is a simple programming language. It contains a small instruction code. In L we can describe images as well as program modification and generation strategies. Here are some properties of L: L contains simple arithmetic operations such as addition and subtraction, a loop statement, subroutine calls, an instruction to set a pixel, elementary mutations on programs (insert, delete, move or replace an instruction), recombination of two programs (crossover), guessing a program and concatenating of a few already known programs.

Programs are directed graphs G=(V, E). Each node represents an instruction. The edges can be decomposed in two disjoint sets: The set of flow edges E_f and the set of access edges E_a. The edges in E_f span a tree $G_f = (V, E_f)$, the processing order is given by the DFS-numbering of G_f. The edges in E_a indicate accesses to pointers of data values. A L-program does not contain any data values, they are stored in a data set. The data set D itself is a graph. Its structure is similar to the structure of the program P which D is associated with. Access to concrete values is achieved by value edges E_v. Elements of E_v are arcs from a node in a program to a node in a data set.



Another set of Edges is E_c , the set of code edges. Elements of E_c are arcs from a node in a program to a node in another program. They are used for references to (sub-)programs.

Data values may be fuzzy. A fuzzy value is represented by a binary string, in which only some of the leftmost bits are specified. Hence, a fuzzy value in fact is an interval.

Separation of program and data, although theoretically not necessary, turned out to be useful for two reasons: firstly the adaption process should have an easy access to data. So we keep the somewhat "conceptual" part (the program) separated from the properties of a concrete object such as position or height denoted by the data. Secondly the deletion of data sets associated with a program is easily accomplished. This simple implementation of "forgetting details" helps to bound the space requirements of the system.

By the use of fuzzy values we can make the system ignore small details by terminating the adaption process early (e.g. after a fixed time) or forget details partially by deletion of some bits in one or more values.

Why did we choose graphs as a syntactic form for L-data sets? In the first place it is easier to enumerate graphs rather than strings because an enumeration algorithm does not need to care about complete bracketing (e.g. loops) [6]. Furthermore, some modification strategies are easier to define on graphs than on strings. Since programs and data sets have similar structures, the structure of a data set can be exploited during the adaption process by inferring a favourable order of values to be adapted.

4 The adaption process

Given a fixed input image I and a fixed program P (which is an image description), the adaptor computes the data values in the data set associated to P and I. For that, I is "smeared": the values black and white are transformed to potentials. The hard cuts due to the black & white scaling are replaced by smooth transitions.

Let the Dimension of the images be NxN. R=[-(N/2-1),...N/2-1] is called the range. Let $B : R^2 \to \{0,1\}$ be an input image and let $A: R^2 \to \{-1,0,1\}$ be an internal image. The meaning of 1 is black, 0 means white and -1 means no description (undefined) of the pixel. The potential of B in a point p generated by a point p_0 relating to color f, $f \in \{0,1\}$ is a function $P_{B,f} : R^2 \times R^2 \to \mathbb{R}$

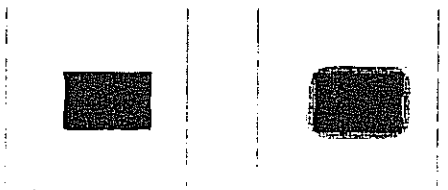
$$P_{B,f}(p_0,p) = \begin{cases} C_{HP} & \text{if } p_0=p \text{ and } B(p_0)=f \\ F(dist(p_0,p)) & \text{if } p_0 \neq p \text{ and } B(p_0)=f \\ 0 & \text{else} \end{cases}$$

dist() is a distance function, e.g. the euclidian distance or the Minkowsky distance, $C_{HP} > 0$ a suitable constant and F a monotonically decreasing function of positive range. Each point p_0 generates a potential relating to the color of p_0 in the input picture. The value of the generated potential decreases by the distance.The potential of an input image B relating to color f is a function $BP_{B,f}: R^2 \to \mathbb{R}$

$$BP_{B,f}(p) = C_{ANULL} + \sum_{p \in R^2} P_{B,f}(p_0,p)$$

Here, $C_{ANULL} < 0$ is a suitable constant.

Example:



input image — potential of input image relating to black

The potentials of an input image make an optimization process possible (the adaption process): the quantity to be maximized is the potential of the internal image. This is defined as follows: Let $M_{A,f} := \{p \in R^2 \mid A(p)=f\}$ be the set of all points being f-colored in the internal image A. The potential sum of an internal image A relating to color f is defined as

$$PS_{A,f} := \sum_{p \in M_{A,f}} BP_{B,f}(p)$$

The mean potential of an internal image A relating to color f is $PM_{A,f} := PS_{A,f} / |M_{B,f}|$. The potential of an internal image A relating to color f is a linear combination

$$P_{A,f} := k_1 \cdot PM_{A,f} + k_2 \cdot PS_{A,f} .$$

k_1, k_2 are suitable constants. The first term 'pulls' the position of the internal image over the position of the input image, the second term allows adjustment of shape.

5 The Evaluator

Programs which encode image descriptions are judged by the agreement of the internal image with the input, their lengths, their space requirements and the space requirements of the associated data set. The quality of an image description is computed as follows:

Let B_i be an input image, P a L-program and D a L-data set, A_i an internal image, which is generated by (P,D). Furthermore we define:

$Pixb_{A_i,B_i} = \#\{p \mid A_i(p) \neq -1\}$ as the number of described pixels in A_i .

$Pixg_{A_i,B_i} = \#\{p \mid B_i(p)=A_i(p)\}$ as the number of correct pixels in A_i (with respect to B_i) .

T(P) as the runtime of P, (each instruction counts 1),

L(P) as the number of instructions (nodes) in P,

S(D) as the sum of the bits representing all values in D.

These parameters of P, D and B_i form a 6-tupel

$$GK_{P,D,B_i} = (Pixb_{A_i,B_i}, Pixg_{A_i,B_i}, P_{A_i}, T(P), L(P), S(D))$$

The quality of P with D for B_i is the inner product of these 6-tupel and a weight vector $\alpha=(\alpha_1,\alpha_2,...,\alpha_6)$. In our first implementation we used the following values:

α_1: -2 α_2: 4 α_3: 0 α_4: 0.1 α_5: 1 α_6: 0

Programs which are codes for program modification and generation strategies are judged by the quality of the programs they produce.

Quality of a program is a real number. We use it to compute the probability for trying out the program once more, e.g. to describe a new input or to generate new programs.

6 The Memory

Programs and data sets are stored in a memory, which has a structure given by the code edges E_c used by call instructions. This allows a recognition algorithm: When the input image changes, the system tries to describe the new image by adapting some already known programs. The syytem tries to adapt those programs which turned out to be the best until now (maybe the input has only changed a little). In the next step, the memory is scanned along the code edges. This is done by using an algorithm which can be seen as a simple ,kind of association. If some programs $P_1,...,P_k$ call a program P and P is a good partial description of the input, the system tries the programs $P_1,...,P_k$ by trying some subprograms not being shared by all $P_1,...,P_k$ (if it is possible). For instance suppose that the system observes a square in the input image (by noting that a program P which describes a square is a good description for a part of the input) and the memory contains two programs P_1 and P_2 which call P as a subprogram (maybe P_1 describes a house, P_2 a cabinet). Then the system tries to adapt P_1 and P_2. Assume the system has not seen too modern cabinets and only one-family houses which are no bungalows, it probably tries to adapt a triangle.

Further extension of the memory structure (edges symbolizing semantical relations, history of program generation, ...) are considered.

7 Ideas and the size af ideas

The actual state of the system is given by the set of programs in the memory. A change of the state is made by adding a new program being generated by a strategy S. An idea for a description of an input image I can be defined as a sequence $S_1,...,S_k$ of strategies producing a program P describing I. Each strategy S_i has a probability $p(S_i)$ to be used. We use the logarithm of the inverse of the probability as a measure of the conceptual size of an idea. The size of an idea is defined as

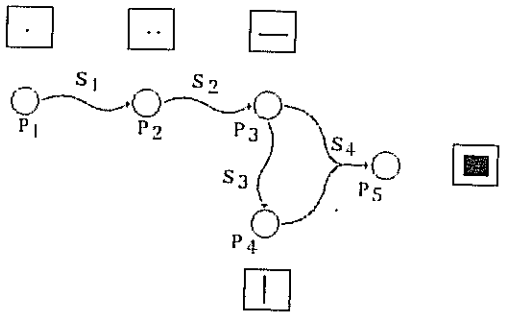
$$\sum_{i=1}^{k} \log(1/p(S_i))$$

The jumpsize of an idea $S_1,...S_k$ is defined as

$$\max_{i=1...k} \log(1/p(S_i))$$

Furthermore the jumpsize of the description P can be defined as the minimal jumpsize of all ideas leading to P and the jumpsize of 'describing I' as the minimal jumpsize of all P describing I. All jumpsizes are related to the actual state of the system. While the jumpsize of an idea might be easy to calculate, jumpsizes of descriptions or of input images do not seem to be easy to calculate. The jumpsize of an idea can be calculated for examples:

Example: Jumpsize of an Idea from 'one point' to ' a rectangle'



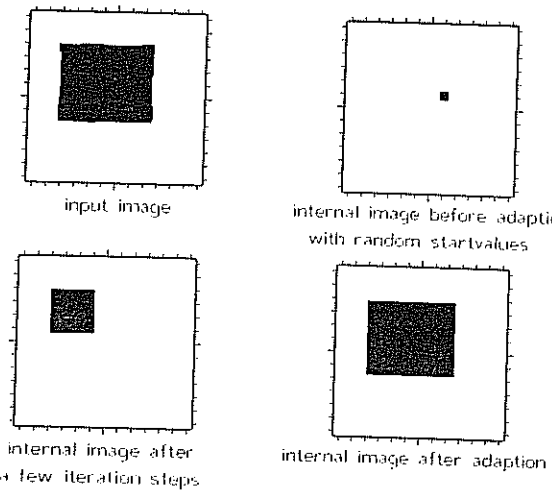
P_1 is a description of one point. P_1 is in the memory. $P_2,...,P_5$ are descriptions for the shown images. $S_1,...,S_4$ are strategies generating the programs $P_2,...,P_5$.

Jumpsize of the Idea:	using L	using a PASCAL-like language
$\log(1/p(S_1))$	21	21
$\log(1/p(S_2))$	16	31
$\log(1/p(S_3))$	10	10
$\log(1/p(S_4))$	24	26
jumpsize	24	31

8 State of the Implementation

A simpler system was implemented in 1988 [3], [4]. This system did not contain a structured memory and the syntax of the description language was not based on graphs. The strategies for program generation/modification were fixed and could not be changed by the system. We mainly examined the adaption process and the strategies mutation and recombination, which are well known from biology. The system was able to recognize simple geometric objects like lines, triangles and squares as well as compositions of some of the objects. The system presented in this paper is partially implemented [5], [6]. It will be completed soon.

An example for the work of the adaptor [5] is:



9 Conclusions

The presented system has two important properties: it learns descriptions of input pictures and organizes its knowledge by itself. The descriptions are programs in a simple programming language. By the quality function the system is able to evaluate a program and to decide wheter it should be added to the memory. The structure of the memory is determined only by the input images and their order. No external programming is made. We plan the extension of our approach to other perceptory channels.

References

[1] Kolmogorow, A. N., "Three Approaches to the quantitative definition of information problems", Information transmission, Vol. 1, No. 1, 1965, pp. 1-7

[2] Solomonoff, R. J., "A formal theory of inducture inference Part 1 & 2", Information and Control, Vol. 7, 1964, pp. 1-22, 224-254

[3] Bergmann, P., Paul, W. J., Thiele, L., "An Information Theoretic Approach to Computer Vision", Dynamical Networks, Berlin, 1989, pp. 52-58

[4] Bergmann, P., et al., "Implementierung eines informations-theoretischen Ansatzes zur Bilderkennung", Proc. Innovative Informations-Infrastrukturen, III-Forum, Saarbrücken, 1988, pp. 187-197

[5] J. Keller, "Ein Algorithmus zu Lösung eines speziellen Optimierungsproblems", Diplomarbeit, FB Informatik, Universität des Saarlandes, Saarbrücken, 1989

[6] S. Evangelidis, "Ein effizienter Aufzählalgorithmus für eine spezielle Sprache für ein selbstorganisierendes System zur Bilderkennung", Manuskript, Universität des Saarlandes, Saarbrücken 1990