An Information Theoretic Approach to Computer Vision

P. Bergmann, W. Paul and L. Thiele - University of Saarbrücken, West Germany

Abstract: Man tries to understand the real world by making theories. Two theories are competing, if they describe the same phenomena. The principle of Occam's Razor says that among competing theories the simpler one is to be preferred over the more complicated one. In the spirit of Kolmogorov and Solomonoff [K,S] a theory *is nothing but* a set of descriptions for observations. The theory is nontrivial if the descriptions are shorter than the observations. Formally observations are sequences of zeros and ones and descriptions are computer programs. In this paper we try to give evidence that with present technology systems can be constructed, which build such theories *in reasonable time*.

1. Introduction. This paper contains speculations about how to build artificial intelligent systems. We start with some observations about freshly made natural intelligent systems (babies).

1.1 babies are born with a brain. This brain is presumed to have computing power which is very large compared to that of traditional AI-systems.

1.2 in spite of all the computing power of their brain babies behave for many months not very intelligently. Suppose babies *do* compute during that time. What are they computing ? And has this extended startup phase something to do with the fact that eventually humans become smarter than traditional AI-systems. which have to be smart right away.

1.3 babies have a basic drive to understand and control the part of the world which they can observe. While lying in their cradle they learn to focus their eyes at things. particularly their hands. They learn (apparently by trial and error) how to control their hands, for example in order to hit at objects which are suspended within reach of their hands.

Suppose we attach a camera and a robot arm to a sufficiently powerful computer system. Is there a way to program the system *in a uniform way*. such that the following happens: the system learns to focus the camera and to control the robot arm. It starts to play with objects which are placed within reach of the robot arm and it eventually discovers. that when it lets an object fall, then in time t it falls down by a distance proportional to t^2.

Such a system might not be completely beyond reach. In this paper we will sketch a system for the particular AI problem of computer vision which has the following properties.

1.1a it benefits from large computing power

1.2a it spends considerable amounts of time with exponential search. particularly after a cold start.

1.3a it is driven by *one single principle* : it searches for short descriptions of large sets of input data.

We will argue that such a system will by itself find the concepts :

straight line
rectangle
parallelogram
neighborhood

and will identify these concepts as useful. It will describe figure 1 as

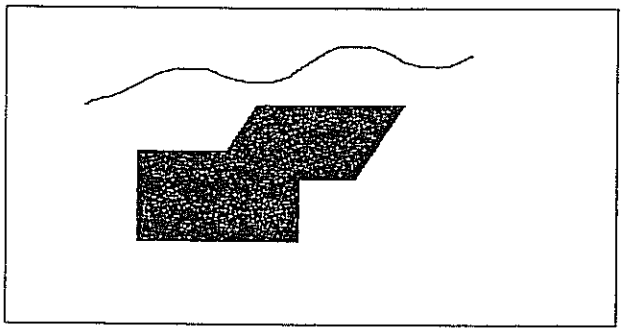

Fig . 1

a white rectangle, on top of it a black rectangle, a parallelogram and a line (not a straight one).

2. Information theoretic view of pattern recognition. Let P and Q be two dimensional patterns. In this section we will propose a very general definition of what it means to *recognice pattern* P *in pattern* Q.

As a first example consider figures 2 a to 2 e. A human will clearly recognize the A of figure 2a in figures 2b and 2d, probably in figure 2c and probably not in figure 2e. Observe that definitions based on Hamming distance will suggest that in fugure 2b the A is as absent as it can possibly be.

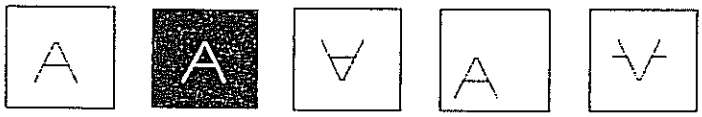

Fig. 2a - 2e

As a second example consider figures 3a to 3c. A human will recognize the camel in figure 3a in figure 3b but probably not in figure 3c, because the animal behind the palm tree migh as well be a cow.

Fig. 3a

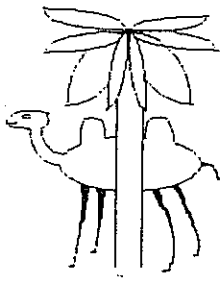

Fig. 3b

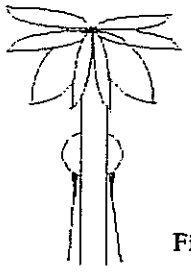

Fig. 3c

Our definition will be based on the concept of *relative Kolmogorov complexity* [K,S]. Fix a universal programming language L. The relative Kolmogorov complexity $K_L(P \mid Q)$ of pattern P given pattern Q relative to language L is the length (measured in bits) of the shortest program p in L which started on input Q produces output P and halts. Let ε be the empty pattern. Then $K_L(P) := K_L(P \mid \varepsilon)$ is called the Kolmogorov complexity of P relative to L.

If L' is a second universal language and s is the length of an interpreter written in language L for programs in L', then $K_L(P \mid Q) \leq K_{L'}(P \mid Q) + s$. In the usual applications of Kolmogorov complexity one is only interested in the asymptotic growth of the complexity of infinite sequences of patterns. In this case by the argument above the choice of language L contributes only an additive constant and is therefore of little consequence. Therefore the subscript L is usually dropped. We are here interested in complexities of particular fixed patterns like the ones in figures 2 and 3. Hence for us the choice of L *is* important. For the remainder of the paper it suffices to assume L to be a reasonable language somewhere between PASCAL and assembler language.

Now suppose $K(P \mid Q) = K(P)$. Then pattern Q is of no help whatsoever in reproducing pattern P. Hence this is a good formalization of the intuitive concept: Q does not in any sense contain P. This formalization was used in several proofs of lower bounds for the runtime of Turing machines[P,PSS].

If on the other hand $K(P \mid Q)$ is much smaller than K(P) then Q contains much information about P. Pattern P can be reproduced from pattern Q by a program which is much shorter than any program which has to produce P from scratch. It is tempting to use this as a formalization of the concept: P can be recognized in Q. But there are several drawbacks. First the function $K(\ \mid\ )$ is not computable. Thus for the purpose of constructions of any kind this would be of very limited use. Second no requirements are made on the computational requirements of the short program which produces P from Q. This program might have a very large runtime. Third there might be a short and fast program which would reproduce P from Q but in a sense the program is so far out that one would hardly try it. Take as an example a portrait of 1000 x 1000 pixels of a well know person. Align the pixels linearly row by row. Roll up the resulting bit string of 1 million bits. For most purposes the well known person cannot be recognized in the resulting round picture, although the decoding procedure is simple enough, *if you know it.*

This suggests the following "subjective" definition: pattern P can be recognized in pattern Q by system S if S knows a short and fast program p, which produces P from Q and which is much shorter than any program known to S, which produces P from scratch.

The reader is asked to check this definition against the examples in figures 2 and 3. It also covers the example in figure 4 which we recognize as a camel, although a whole hind leg is missing. The procedure to complete the camel is simply: copy the hind leg which is there and put it in the proper place.

Fig. 4

3. How to formally define a camel or a good set of programs which reproduce a prototypical camel from patterns Q is something we don't know how to do. Thus if we want to build a camel recognizer, then our only hope is to build a system which by itself produces the definition of the camel and the procedures for reconstructing same camel by itself.

We proceed to sketch a system which makes successive observations of the world (for our purposes an NxN array of black and white pixels), tries to identify *interesting* patterns in it, and in case it succeeds remembers that pattern. The system will find a pattern P interesting if it can find a program p, which is fast, has much less bits than P and which reproduces P. Pattern P is remembered by storing program p.

There should be three basic mechanisms for generating candidate programs p.

3.1 systematically enumerating all programs which have at most c instructions, where c is some small magic constant (like 7).

3.2 mutation of programs which are already stored: changing constants, deleting instructions, adding instructions or replacing instructions by one or few other instructions.

3.3 combination of programs: concatenation, replacing instructions by other programs.

If a system like that is confronted with something which is unlike anything it has seen before (like after a cold start), then it will have to rely heavily on 3.1 . During the systematic enumeration of candidate programs little progress can be observed superficially. Also progress comes like ideas: even immediately before the idea is there, there is no hint that it will come.

Programs p must be stored by the system in a nontrivial data structure. The underlying programming language L can be designed such that is has few types of instructions, thus the combinatorial explosion in 3.1 can be kept moderate. But the number of programs p which are remembered will become very large. This is bad for the runtime of the system if mechanisms 3.2 or 3.3 are used. Also if in a new pattern P a subpattern P' is found, which was previously successfully combined (by mechanism 3.3 say) to another pattern P", then chances are, that this combination will be successful again. Think of P as a face, P' as a pair of eyes and P" a nose. Therefore it would make sense to store programs as nodes of a weighted graph: if program p' was successfully combined to program p", then there should be an edge between p' and p". This weight should vary in time and represent the rate at which this combination was successfully applied.

One can try to teach a system like that by confronting it with an appropriately chosen sequence of patterns. A new pattern P can be taught in reasonable time, if the system can derive from the programs it has stored a good description for P with a reasonable number of tries, i.e. by modifiying existing programs in 5 or 7 places but not in 100 places. As long as progress can be made in small steps like that the system can continue to learn. Thus we begin to study the question:

4. How much progress is possible in small steps ?

This question turns out to be amenable to theoretical study. Suppose an all white pixel array A is presented to the system. Then by enumeration of all very short programs the system will find the program p_1:

```
j := j0;
for i = α to β do
A[i,j] := 0 od
```

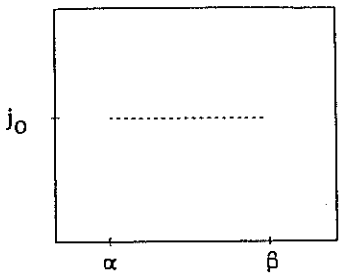

Fig. 5a

(see figure 5 a). This program is very fast and has a length of $\log \alpha + \log \beta + O(1)$ bits. It describes a pattern (a white horizontal line) of $\beta - \alpha + 1$ bits. Thus if $\beta - \alpha$ is large enough, this pattern will be classified as interesting and the program above (or a better one) for it will be stored.

By adding instructions to program p_1 one gets program p_2:

```
j := j0;
for j = γ to δ do
for i = α to β do
A[i,j] := 0 od od
```

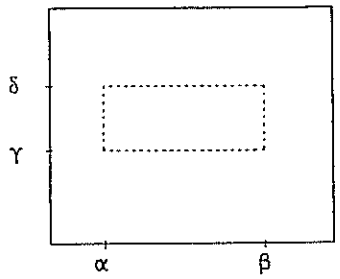

Fig. 5b

(see figure 5 b), which describes a white rectangle. The first instruction in p_2 can be deleted. This turns produces the even shorter program p_3:

```
for j = α to β do
for i = γ to δ do
A[i,j] := 0 od od.
```

Variation of the constants will produce in the case $\alpha = \gamma = 1$, $\beta = \delta = N$ an all white background.

Next if we present to the system figure 5 c we ge by variation of the constants in p_3 a program p_4 which produces the black rectangle. Concatenating a program for white background with this program produces figure 5 c.

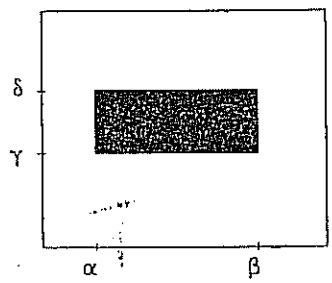

Fig. 5c

By a somewhat greater mutation of program p_4 we get p_5:

```
for j = γ to δ do
for i = α to β do
k := i + j - γ;
A[k,j] := 1 od od
```

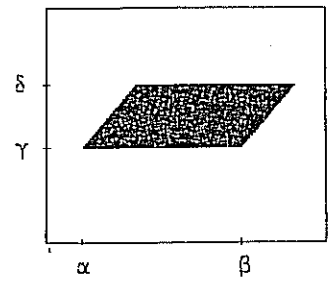

Fig 5d

which produces the parallelogram in figure 5d Except for the crooked line the system can now reproduce figure 1 by concatenation of programs it knows. Consider figure 6a . It has a white background and a single black dot P_0, which can be described by program p_6:

```
i := x0;
j := y0;
A[i,j] := 1.
```

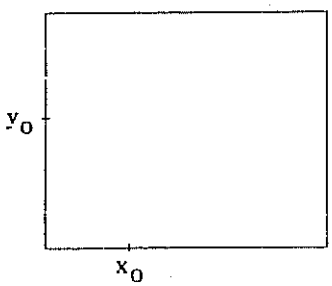

Fig. 6a

Two black dots like in figure 6b can be described by concatenating two copies of p_6 and chanching constants in the second copy. One gets program p_7:

```
i := x0;
j := y0;
A[i,j] := 1;
i := x1;
j := y1;
A[i,j] := 1.
```

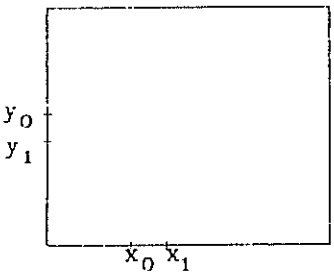

Fig. 6b

This program has length $\log x_0 + \log y_0 + \log x_1 + \log y_1 + O(1)$. This would be fine if the two dots were in generai position. But the dots in figure 6b are not. Indeed changing only two instructions in p_7 produces p_8:

```
i := x0;
j := y0;
A[i,j] := 1;
i := i + x1-x0;
j := j + y1-y0;
A[i,j] := 1.
```

This program is shorter than p_7 if $\log (x_1-x_0) + \log (y_1-y_0) < \log x_1 + \log y_1$. Repeating the same process k-2 more times produces a description of a crooked line of k points. This description reflects human intuition.

5. Plans for experimantal work. Installation of a system of the kind described above is planned for 1988 as a subproject of the I.I.I. (Innovative Informations Infrastrukturen) project. This is a joint project of the University of Saarbrücken and Siemens. In this projects hundreds of Siemens PC's are hooked together via the network CANTUS, which was developped at the computing center of the University of Saarbrücken. Further processing power will be available on DATIS-P, a massively parallel computer under construction in Saarbrücken as a project of SFB (Sonderforschungsbereich) 124 sponsored by the DFG.

6. References

[K] Kolmogrov, A. N.: Three approaches to the quantitative definition of information problems in information transmission vol. 1 No. 1 pp. 1-7, 1965.

[P] Paul W.: Kolmogorov complexitiy and lower bounds, Fundamentals of computation theory 79, Band 2, pp. 325-334, 1979

[PSS] Paul W. Seiferas J. I. Simon J.: An information-theoretic approach to time boungs for on-line computation, Journal of computer and system sciences Vol. 23, No. 2, 10/81, pp 108 - 126 , 1981

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