

AVOIDING EXPLOSIVE SEARCH IN AUTOMATIC SELECTION OF SIMPLEST PATTERN CODES*

P. A. VAN DER HELM and E. L. J. LEEUWENBERG

University of Nijmegen, Montessorilaan 3, P.O. Box 9104, 6500 HE Nijmegen, The Netherlands

(Received 17 January 1985; in revised form 25 July 1985)

Abstract—According to the coding theory of pattern perception, the preferred organization of a pattern is reflected by the simplest code that represents this pattern. The number of codes, out of which the simplest one has to be selected, grows exponentially with the complexity of the pattern. So in computer simulation it would not be wise to generate all codes and then select the simplest one. This would take a lot of computing time. The present study proposes a procedure which avoids this explosion of code generation and yet obtains the simplest code. The central part of this procedure consists of translating the search for a simplest code to a shortest route problem.

Structural information Coding theory of pattern perception
Hierarchical representation of linear codes Minimization Automatic code reduction
Shortest route problem

INTRODUCTION

Patterns can be interpreted in different ways, but usually one interpretation is preferred. By several scientists this preferred interpretation is regarded as the most regular organization of the pattern.⁽¹⁻³⁾ In most models of pattern coding this preferred organization is specified as the one that is attained with a minimum of procedural steps.⁽⁴⁻⁶⁾ However, there is evidence from perception research that in perception this preference is not based on the process itself, but on the final outcome of this process. Independent of the way in which a description is determined, the visual system tends towards the most probable pattern description,⁽⁷⁻⁹⁾ or towards the simplest pattern description.⁽¹⁰⁻¹³⁾ The concept of "simplicity" pertains to the structural information approach⁽¹⁴⁾ and is used here as the starting point. On the one hand, this concept might be perceptually relevant, but on the other hand, it seems to require a very laborious process to find the simplest pattern description. According to a naive process model, all codes of a pattern have to be constructed before the simplest one can be selected. As will be shown later, the amount of all such codes increases dramatically with the pattern complexity. The aim of this paper is to show that this explosion can be largely reduced to a manageable size and that therefore "representational simplicity" is a realistic criterion for the preferred pattern organization.

The structural information approach provides a model which is the starting point of the present study.

First we will give an overview of this model as far as it is relevant to our present work. The model only concerns the perceptual representation of patterns, not the process that leads to such a representation. This process is considered as a black box in the total perceptual system. In the present work, we want to fill in this black box by proposing a procedure for the selection of a simplest code for a given pattern. This does not imply we think the real process works like this procedure, which only tries to simulate the final outcome of the process. But if the outcomes of the simulation are attained in an acceptable way (by avoiding explosive searches) and if they are in agreement with the outcomes of the real process, the simulation will be a support to the assumption about the perceptual system on which the structural information approach is based.

STRUCTURAL INFORMATION

The structural information approach deals with the perceptual organization of patterns.⁽¹⁵⁾ The organization of a pattern is reflected by a description or code of this pattern. However, a pattern can be described by several codes. Although it ought to be possible to reconstruct the pattern from each code, these codes are not equivalent: some codes express more regularity in the pattern than other codes. In other words, some codes reflect a "simpler" organization than others. The core of the structural information model is that the code which reduces the pattern organization to its "simplest" form, reflects the preferred perceptual organization. This tendency is called the "minimum principle".^(16, 17) This is the general idea. In order to be more specific about the description of patterns we need

* This research was supported by the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

to discuss two levels of coding: the semantic and the syntactic level.

Semantics: a linear sequence of symbols, say 'k a k a k a k a', may represent a musical structure such as 'do, re, do, re, do, re, do, re'. It also may represent a visual pattern such as a square (see Fig. 1). In the latter case the symbol sequence can be obtained by tracing the contour of the pattern while the symbols represent the subsequent angles and line lengths.

The symbol sequence is called a "primitive code" and the symbols are called "primitive elements". In a primitive code like the one above, one can clearly see certain regularities. But these regularities are not yet described in any way. This is where we enter the syntactic level, which is the main concern of this paper.

Syntax: the structural information model provides syntactical rules to "simplify" primitive codes. For instance, consider the code 'a a a a a'; it consists of five equal elements. Intuitively, one tends to reduce this code by describing it by means of an iteration code such as '5 * (a)'. But what does "reduction" or "simplification" actually mean here? Considered as symbol sequences, both codes consist of five symbols, so in this sense the second code does not lead to reduction. However, there is a clear distinction between the two codes; for this we have to look at what the codes say themselves. The first code consists of five primitive elements, but says nothing about any relation between these elements. They could just as well have been five different elements and only a bystander can see that they are equal. The second code however refers explicitly to five equal elements and only needs one primitive element to say this. So the reduction takes place on the number of primitive elements. But again, does this reduction have any perceptual meaning? Its meaning is a consequence of the fact that the code explicitly expresses identity relations between primitive elements and thereby reflects regularities in the original pattern. An alternative for the second code could be 'a 3 * (a) a', but this code expresses less regularity than '5 * (a)'. So the "simplest" or "most regular" organization of a pattern is reflected by a code which expresses the most identity relations between primitive elements. As we have seen above, expressing such a relation leads to a reduction of the number of primitive elements in the code, which number is called the parameter load P .^(1,2) Therefore the preferred organization of a pattern is reflected by a code with

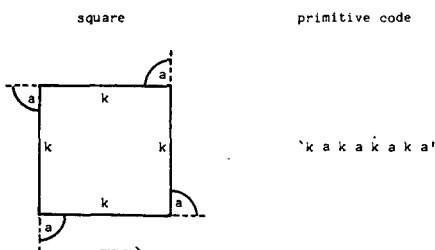


Fig. 1. Semantic coding.

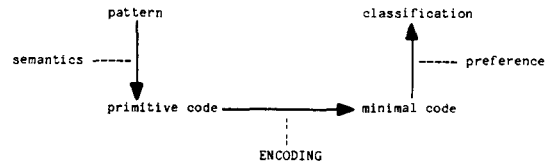


Fig. 2. Structural information model.

minimal parameter load. In Fig. 2 an overview is given of the entire concept; the "encoding" will be the main topic of this paper.

In the structural information model, identity relations are expressed by means of "ISA-forms", where "ISA" stands for Iteration, Symmetry and Alternation. With these ISA-forms we can formulate reduction rules.

Examples:

$$\text{I-form: } 4 * (a) \quad \Leftrightarrow a a a a$$

$$\text{S-form: } S[(a)(b)] \quad \Leftrightarrow ab ba$$

$$\text{A-form: } \langle (a) \rangle / \langle (b)(c) \rangle \quad \Leftrightarrow ab ac.$$

The use of parentheses seems to be superfluous, but beside identities between single elements we also want to be able to express identities between groups of elements. Parentheses are used to separate these groups and provide an unambiguous evaluation of the ISA-forms. Furthermore, Symmetry and Alternation can occur in two different forms.

Examples:

$$\text{I-form: } 3 * (ab) \quad \Leftrightarrow ab ab ab$$

$$\text{S-forms: } S[(a)(bc)(d)] \quad \Leftrightarrow a bc d d bc a$$

$$S[(a)(b), (d)] \quad \Leftrightarrow ab d ba$$

$$\text{A-forms: } \langle (a) \rangle / \langle (b)(cd)(e) \rangle \quad \Leftrightarrow ab acd ae$$

$$\langle (a)(b) \rangle / \langle (cd) \rangle \quad \Leftrightarrow acd bcd.$$

Note the difference between the two S-forms: in the second form 'd' does not belong to the symmetry part of the form. An A-form expresses the fact that the code can be divided into several subcodes which all begin or all end with the same element(s), dependent of the direction in which the code is scanned.

An ISA-form not only generates a code, but also operates on a code. Consider for instance the S-form 'S[(ab)(c)(ab)(c)]'. The argument of this form looks like a code 'x y x y' where $x = (ab)$ and $y = (c)$. Now the hierarchical structure of the code becomes visible: if the S-form itself represents the "lower" level of this structure, the argument code represents a higher hierarchical level. On this higher level the reduction rules from above can be applied anew, which yields a reduction of the argument code to '2 * (xy)'. So the total structure can be represented by 'S[2 * ((ab)(c))]'. A code which allows no further reduction is called an "end code". An end code covers the whole primitive code, but a primitive code itself may correspond to several end codes, whose parameter loads may be

different. Therefore an end code with minimal load P is called a "minimal end code".

Example:

Consider the primitive code 'a b b a b b'. One end code is 'S[(a) (b)] 2*(b)', which has three remaining parameters, so the parameter load is $P = 3$. Another end code is '2*(a 2*(b))', for which $P = 2$. This last one clearly is a minimal end code, since there is no other way to get more reduction.

There is one point of discussion left. Consider the code 'a b a c'. Two possible end codes are '<(a)>/<(b) (c)>' and 'S[(a), (b)] c'. Although these end codes seem to be different, they both express the identity relation between the first and third element of the primitive code. In general, if two end codes (like the ones above) express the same identity structure, we will call these end codes "equivalent". To express this equivalence we use the concept of an "abstract code".^(1,2) An abstract code represents the identity structure as expressed by an end code. It is obtained as follows: firstly, replace all parameters in the end code by arbitrary but different symbols. This way, the two end codes above will transform to '<(x)>/<(y) (z)>' and to 'S[(p), (q)] r'. Secondly, evaluate these end codes, which yields the codes 'x y x z' and 'p q p r'; these are the so called abstract codes. They both have the same identity structure (first element equals third element). Hence, the two original end codes are equivalent. In this example the abstract codes have the same identity structure as the primitive code has, but there are many primitive codes whose total identity structure cannot be expressed by one end code. The number of possible abstract codes is only a small fraction of the number of possible primitive codes.^(1,8) So the ISA-forms suggest a constraint in the perception of identity structures. This constraint is relevant: if every identity structure could be expressed by a code, the codes would not classify patterns in a meaningful way, since each pattern would form its own class.

Example:

Consider the code 'a b a b a a'. One possible end code is '<(a)>/<2*(b) (a)>', which yields the abstract code 'x y x y x z'. The end code 'S[(a) (ba)]' yields 'p q r q r p', which has a different identity structure. Both end codes disregard one identity relation. In fact, there is no other end code which could express the complete identity structure of the original code.

We would like to consider the structural information model as a black box, where only the input (primitive code) and the output (abstract code) are relevant. So we do not necessarily want to differentiate between end codes, but rather between their abstract codes.

A minimal end code was said to have a minimal parameter load. This load is equal to the number of different elements in the corresponding abstract code, so it is correlated to the identity structure of abstract

codes. Therefore it also provides a good criterion to differentiate between abstract codes. So an abstract code we are looking for, corresponds to a minimal end code.

From the above it will be clear that the main goal of the structural information approach is to relate minimal end codes to a primitive code. This search for end codes is called the "encoding", on which we will concentrate in the remaining part of this paper.

THE ENCODING

If we want to build a computer simulation of the encoding process, it is important to avoid an explosive computing time in finding a minimal end code. Otherwise it would not support the minimum principle as a realistic criterion for the preferred pattern organization. Explosive computing time means the computing time grows exponentially as a function of the number of elements in the primitive code. The structural information approach gives rise to two explosive searches. The first one concerns the A-form, on which we will go into detail when discussing the construction of ISA-forms. The main topic of this paper however is the other explosion. It concerns the number of possible end codes even if that first explosive search concerning the A-form has been avoided.

A naive way to search for a minimal end code would be to look for an ISA-form that describes a part of the primitive code, to substitute that form into the primitive code and to proceed the encoding with the resulting code. Figure 3 might make this point clear.

Every edge in the graph of Fig. 3 corresponds to one subcode. This implies every route from vertex 0 to vertex 5 represents a partitioning of the primitive code into subcodes. In general, every route could yield an end code after the subcodes of that route have been encoded. The number of routes in the graph of Fig. 3 is 2^4 , while in general for a code with length N (number of elements), this number is 2^{N-1} . This implies that if we generate all end codes and then select the best one, the number of operations to be executed lies in the order of 2^N operations. Each operation takes a fixed amount of computing time, so the computing time of such a process lies in the order of 2^N ms. Using the notation "O(.)" to indicate the order of the computing time of a

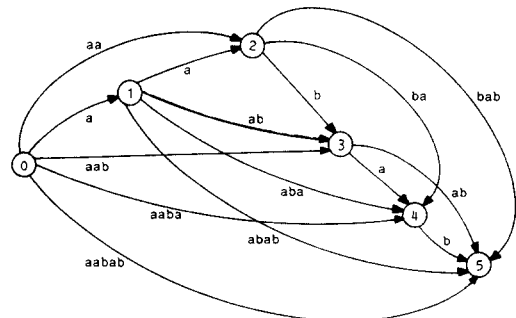


Fig. 3. Graph representing partitioning of a code into subcodes.

process, it means the computing time of such a naive process is $O(2^N)$. At the start of this section we argued that such a computing time is not acceptable. Therefore we have to find a method that only takes a polynomial computing time of $O(N^p)$ with 'p' a fixed number.

Figure 3 gives a suitable framework to realize an acceptable encoding process. It involves the translation of the search for a minimal end code to a shortest route problem, by which it is not necessary to generate completely all possible end codes while it is still possible to select a minimal one. In short, the translation goes as follows. If we would know the minimal parameter load of each encoded subcode in Fig. 3, we could consider this load as the "length" of the corresponding edge. Then the "length" of a route from vertex 0 to vertex 5 would correspond to the load of a code that describes the whole primitive code. So a "shortest" route would correspond to a code with minimal load, i.e. a minimal end code. The solution of a shortest route problem only requires a polynomial computing time.

In general, several routes will correspond to the same minimal end code. However, only one of these routes will consist of edges, each of which represents just one element or just one ISA-form. Therefore, in order to make the translation possible, it suffices to find for each subcode the best way in which it can be described by just one ISA-form. A code of length N yields $N * (N + 1) / 2$ subcodes varying from length 1 to length N . This is a convenient polynomial number to investigate each subcode separately. As we will see, this also requires just a polynomial computing time.

The translation to a shortest route problem is crucial in avoiding an explosive search. Therefore we will deal with it in detail. But first we will discuss two topics concerning the algorithm. The first topic is the internal representation of codes, which will be a hierarchical representation. The second topic is the data structure for the storage of information about encoded structures and for a reference scheme to store identity relations between subcodes. These two aspects are not crucial in avoiding explosive searches, but they are important in that they constitute a formal framework in which it is easy to deal with codes and identity relations.

REPRESENTATION OF CODES

In the encoding algorithm, we need an operational representation of codes. Until now we have used linear representations. For a human being these are easily readable, but a computer rather deals with hierarchical structures. As was observed before, codes do have a hierarchical structure. We will elaborate this concept in this section.

In order to attain a well defined hierarchical representation of codes, we have to describe their formal character. Therefore we will consider the following four aspects.

(1) A code like ' $a * (b) c$ ' is a code consisting of three elements: two primitive elements and one I-form. To avoid confusion, we reserve the name "symbol" for primitive elements and use the name "atom" for code elements in general. So an atom can consist of a symbol, an ISA-form or a chunk (see below).

(2) Consider the code $C = 'p q r r p q'$. This code C can be reduced to ' $S[(pq)(r)]$ '. The argument of this S-form is a code, consisting of two elements: ' (pq) ' and ' (r) '. But beside being elements of this argument code, these atoms also contain subcodes of the code C . Such atoms we will call "chunks". In simple words: every part of a code that is placed between parentheses is a chunk.

(3) We made a distinction between a chunk as a code element and the subcode which is contained in this chunk. This subcode has its own code structure, which means it can consist of a code, or of only one symbol, or of one ISA-form, or even of one chunk. For instance, the I-form in ' $S[(a) * 2 * (b)]$ ' has the argument ' $((b))$ ', which is a chunk that contains an element ' (b) ' and this element itself is a chunk that contains a symbol.

(4) If we consider an I-form, like ' $3 * (fg)$ ', we observe that the argument of such a form always consists of only one chunk. The argument of an S-form like ' $S[(ab)(c)(d)]$ ' is a code consisting of several chunks. An S-form like ' $S[(a)(bc), (def)]$ ' contains two arguments: the first again is a code consisting of several chunks, while the second always is just one chunk. An A-form, like ' $\langle (a) \rangle / \langle (bc) (d) \rangle$ ' or ' $\langle (f) (g) \rangle / \langle (de) \rangle$ ', also has two arguments, one of which always is just one chunk, while the other again is a code consisting of several chunks.

Summarizing, we can describe the structure of codes by means of five types of structure elements, which are:

- | | |
|---------------|--|
| (1) CODE: | ATOM sequence. |
| (2) ATOM: | CODE element of type SYMBOL, CHUNK or ISA-form. |
| (3) SYMBOL: | primitive element. |
| (4) CHUNK: | element of an argument code; its content is of type CODE, SYMBOL, CHUNK or ISA-form. |
| (5) ISA-form: | one of the following forms: I-form $\rightarrow 'n * \text{CHUNK}'$ S-form $\rightarrow 'S[\text{CODE}]$ or ' $S[\text{CODE}, \text{CHUNK}]$ ' A-form $\rightarrow '\langle \text{CHUNK} \rangle / \langle \text{CODE} \rangle'$ or ' $\langle \text{CODE} \rangle / \langle \text{CHUNK} \rangle'$. |

With these structure elements we can describe hierarchical representations of codes (see Fig. 4).

In the linear representation, the use of parentheses provides an unambiguous evaluation of codes; in the hierarchical representation this is provided by the use of CHUNKS. So the mapping from the hierarchical to the linear representation is straightforward.

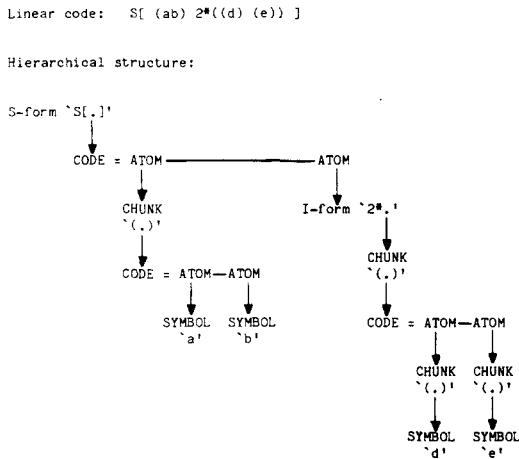


Fig. 4. Hierarchical representation of a linear code.

DATA STRUCTURE

Before the encoding starts, all possible identity relations between subcodes are gathered in a reference scheme. This facilitates the search for ISA-forms and prevents the double encoding of equal subcodes. In setting up the scheme, the subcodes are compared in order of growing length. This way the relations between smaller subcodes can be used while comparing larger subcodes. It implies that the computing time for the reference scheme is $O(N^3)$.

Note: if a subcode has the same elements as another subcode, but exactly in the reversed order, these two subcodes can be described by the same encoding rules; therefore such a reversal relation too is stored in the reference scheme, though we will not mention it any further.

In Fig. 5 an overview is given of the data structure as used in the algorithm.

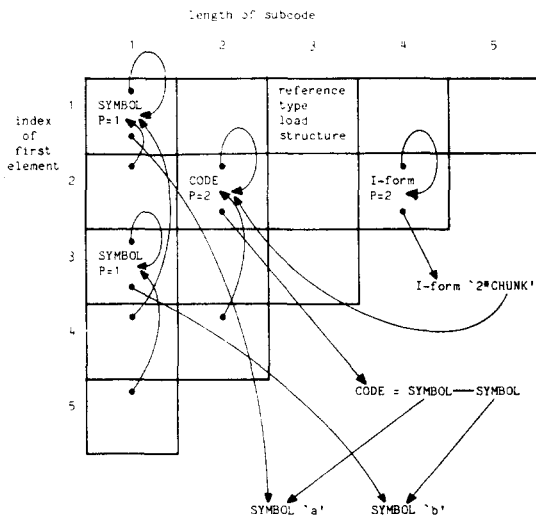


Fig. 5. Data structure with reference scheme for code 'a a b a b'.

Each matrix cell represents one subcode; the indices are the code index of the first element of the subcode and the length of the subcode. Firstly, each cell contains a reference pointer which is part of the reference scheme. By means of this pointer, each subcode gets a reference to the first equal subcode. A subcode that has no previous equal subcode, gets a reference to itself. As a consequence of this reference scheme, the following principle is possible throughout the entire algorithm: equal parts of a code have a reference to physically the same place. This implies that identity relations are transferred almost by themselves to other hierarchical levels. Suppose an argument code of an S- or A-form contains two equal chunks, i.e. both chunks contain an equal subcode. Because of the reference scheme, both subcodes have a reference pointer to the same place. So, to determine whether the two chunks are equal, it suffices to look whether those reference pointers point to the same place and one does not have to compare the entire structure of both subcodes. Moreover, this structure only has to be stored once.

Furthermore, each matrix cell contains data fields 'type' and 'load' to store information about the structure of the subcode after it has been investigated.

Finally, each cell contains a pointer to that encoded structure. For instance, in Fig. 5, investigation of subcode 'abab' yields it can be described by '2*(ab)'. The matrix cell corresponding to this subcode (first index 2, length 4) gets: 'type = I-form' and 'load = 2', while its structure pointer points to the place where the I-form is stored. The argument of the I-form is a chunk, which contains the subcode 'ab'. Therefore we can refer to the corresponding matrix cell (first index 2, length 2), so the structure of 'ab' need not be stored again.

Now that we have established a formal framework to deal with codes and identity relations, we can focus on the translation of the search for a minimal end code to a shortest route problem. First we will discuss the investigation of each subcode separately and then we will explain the shortest route method.

THE INVESTIGATION OF THE SUBCODES

The investigation of each subcode separately proceeds as follows. As argued before, it suffices to look for a best covering ISA-form, i.e. one form that describes the whole subcode with a minimal number of parameters (we will deal with this in a moment). However, we also look at whether the subcode might be described better by a composition of smaller subcodes, which already have been encoded for we investigate the subcodes in order of growing length. If such a better composition exists, it is chosen instead of the best covering ISA-form. Though this is not essential to the shortest route method, it is necessary, for the subcode might appear in a chunk on a higher hierarchical level and then we have to know its minimal load (the encoding of argument codes on higher levels is executed the same way as the primitive code is; we will

return to this at the end of this section). The method to obtain a minimal composition for a subcode is the same as used for the selection of a minimal end code for the whole code, namely by applying shortest route techniques. This will be explained in the next section. As we will see the shortest route method takes a computing time of $O(k^2)$ for a code or subcode of length 'k'.

In searching for a best covering ISA-form, we observe the following aspect. If a subcode can be described by a covering I-form, it can be described too by an S- or A-form.

Example:

- | | |
|---|--|
| (1) Subcode $ababab$ | (2) Subcode $ababab$ |
| I-form $3*(ab)$ | I-form $2*(S[(a), (b)])$ |
| S-form $S[(ab), (ab)]$ | S-form $S[S[(a), (b)]]$ |
| A-form $\langle(a)\rangle/\langle 3*((b))\rangle$ | A-form $\langle(a)\rangle/\langle 2*((ba))\rangle$ |

What we see in this example, holds in general: such an S- or A-form might express the same number of identities as the I-form does, but it never expresses more identities. So a covering S- or A-form can never be better than a covering I-form. For the same reason, a minimal composition too can never be better. Therefore, if a subcode can be described by a covering I-form, we do not need to investigate it any further. This suits well, for such subcodes are the most difficult ones with respect to the S- and A-form. Again for the same reason, we can take the I-form with the smallest chunk as argument. We investigate the subcodes in order of growing length. Therefore, when a covering I-form is found, the content of the argument chunk already has been encoded. This information is used directly to determine the total structure and parameter load of the subcode.

The following procedure is used to find a covering I-form. If a subcode is of length 'k', we look for $p = 1, 2, \dots, k/2$ whether 'p' is a divisor of 'k' and if so, whether the subcode can be described by ' $q*(e_1 e_2 \dots e_p)$ ' with $q = k/p$. For each divisor of 'k', there have to be made ' $q - 1$ ' comparisons between subcodes of length 'p' (using the information stored in the reference scheme). This implies the computing time of the search for a covering I-form is $O(k * \ln(k))$.

If no covering I-form is found, the best covering S- or A-form is selected, several of which in general exist. To satisfy the minimum principle, all possible S- and A-forms have to be investigated before the best one can be selected.

Example:

Consider the subcode 'abbb a'.

When looking for an S-form, it seems to be a good idea to look for the most symmetrical form, i.e. with the largest symmetry part.

Here we would find ' $S[(a) (b), (b)]$ ' with $P = 3$. However, the other possible S-form is better: it is ' $S[(a), (3*(b))]$ ' with $P = 2$.

For a subcode of length 'k', the argument code of the

most symmetrical S-form maximally contains $k/2$ elements. The argument codes of all other possible S-forms (maximally $k/2$) are part of this largest argument code. So after we have encoded the latter one, the encoding of the others is known also, for each subcode of a code is encoded completely. To find the most symmetrical S-form, $k/2$ comparisons have to be made (using the reference scheme) and after its argument code has been encoded, the best one has to be selected out of the $k/2$ possible S-forms (using the information found for the most symmetrical S-form). This implies the computing time to find a best covering S-form is $O(k)$.

Note: for the moment we disregard the computing time to encode the argument code; we will return to it at the end of this section.

I- and S-forms are invariant to the direction in which the code is scanned (left to right or right to left). A-forms however, have to be determined for both directions.

Example:

Consider the subcode 'afcfa f'.

Covering A-forms are:

- | | |
|---|--------------|
| (1) $\langle(a)\rangle/\langle(S[(f), (c)]) (f)\rangle$ | with $P = 4$ |
| (2) $\langle S[(a), ((c))] \rangle/\langle(f)\rangle$ | with $P = 3$ |
- of which clearly the last one has to be selected.

At the start of this section we said the structural information approach gives rise to two explosive searches. Beside the over all explosion of the number of possible end codes, the A-forms also give rise to an explosive search.

Example:

Consider the subcode 'abaa da d'.

It has two minimal covering A-forms:

- | | |
|--|---|
| (1) ' $\langle(a)\rangle/\langle(b) (2*(ad))\rangle$ ' | so the first and second 'a' are extracted; |
| (2) ' $\langle(a)\rangle/\langle(ba) 2*((d))\rangle$ ' | so the first, third and fourth 'a' are extracted. |

This example shows it is not always clear which elements 'a' have to be extracted to get a minimal A-form. The number of possible A-forms is equal to number of ways in which (beside the first 'a') one or more elements 'a' can be extracted from the subcode. This number grows exponentially as a function of the number of elements 'a'. So the investigation of all possible A-forms would give an explosive search. This explosion is smaller than the over all explosion for the end codes, but for the same reasons we cannot disregard it.

We have strong indications the explosion for the A-form too can be avoided by using shortest route techniques. However, this is not yet realized and will take intensive study for it seems to need rather sophisticated programming. It will be a subject of future research. In the meantime we will use a heuristic method, which in most cases leads to a minimum. We are well aware of the fact that using heuristics does not

fully satisfy the minimum principle, but as we will see, the damage due to heuristics should not be overestimated. So we will accept it as a temporary solution.

The heuristic we will use, is the following quite simple method. Consider the subcode 'a b a a c a d'. While scanning from left to right, the first, second and fourth 'a' are extracted to construct the covering A-form ' $\langle(a)\rangle/\langle(b)(ac)(d)\rangle$ '. So the first 'a' is extracted, then the next possible one and so on until a maximum number of them is extracted. It is a straightforward method without any other selection than the one just mentioned. Although this heuristic just takes one A-form out of all the possible ones, we said the damage should not be overestimated. The next example will make this clear for the code we showed before.

Example:

Consider the code 'a b a a d a d'.

Using the heuristic, we find the following covering A-form: ' $\langle(a)\rangle/\langle(b)(ad)(d)\rangle$ ' with $P = 5$, while a better one would be: ' $\langle(a)\rangle/\langle(ba)2*((d))\rangle$ ' with $P = 4$. But this minimum is reached too by:

'S[(a), (b)] 2*(ad)'.

So there are other ways to reach a minimum.

The following example shows it really is a heuristic, by which the algorithm can fail to find a minimal end code.

Example:

Consider the code 'a b a d e a e d a c'.

The only minimal end code is:

' $\langle(a)\rangle/\langle(b)(S[(d)(e), (a)]) (c)\rangle$ ' with $P = 6$.

This minimum will not be reached by using the heuristic, nor by using I- or S-forms instead of the A-form.

Until now we only have dealt with A-forms, where the chunk to be extracted contains one element. For a code like 'a b f a b g a b f', we also have to investigate A-forms where that chunk contains more than one element, e.g. ' $\langle(ab)\rangle/\langle(f)(g)(f)\rangle$ '. Furthermore, for a code like 'a f g c f g', A-forms with one chunk in the right-hand argument have to be investigated too, e.g. ' $\langle(a)(c)\rangle/\langle(fg)\rangle$ '. This way, for a subcode of length 'k', the heuristic method maximally yields $k - 2$ covering A-forms, out of which the best one has to be selected. To construct one A-form using the heuristic takes $O(k)$ computing time, so the time to find the best one takes $O(k^2)$ computing time.

Note: again, for the moment we disregard the computing time for the encoding of the argument codes.

Summarizing the complete investigation of a subcode of length 'k', we find the following computing times:

- (1) for a covering I-form: $O(k * \ln(k))$;
- (2) for a best covering S-form: $O(k)$;
- (3) for a "best" covering A-form: $O(k^2)$;
- (4) to find a minimal composition of smaller subcodes: $O(k^2)$.

So the complete investigation takes $O(k^2)$ comput-

ing time. There are $N * (N + 1) / 2$ subcodes for a code of length N , which implies the total encoding of a code takes $O(N^4)$ computing time. This means we have established a process that takes just a polynomial time instead of an exponential time. This was our main goal.

In this section we disregarded the computing time for the encoding of the argument codes of S- and A-forms. The reference scheme (see previous section) allows us to deal with identity relations on higher levels in exactly the same way as on the primitive level. Therefore, these argument codes can be encoded the same way as the primitive code. So, again it would take a polynomial computing time to encode each of these codes. However we can disregard this time, for it has little influence. The hierarchical depth of an encoded subcode of length 'k' maximally is $^2\log(k)$: at each level a new argument code appears. The length of these codes decreases each level by at least a factor 2, so their influence on the computing time diminishes very rapidly compared to the primitive code. Furthermore, there is no real code where all subcodes have this maximal hierarchical depth (mostly even none of them). Another argument might be the following. After having tested the algorithm on a PDP-11/44 computer with a large number of codes with length up to $N = 85$, the best fit for the actual computing time appeared to be $0.0025 * N^4$ ms. This is in agreement with the theoretical time of $O(N^4)$.

Now that we have investigated each subcode, we can concentrate on the translation to and on the solution of the shortest route problem. This will be discussed in the next section.

SHORTEST ROUTE METHOD

In the previous section we discussed the investigation of each subcode separately. We observed (see Fig. 3) that the total code can be composed out of the subcodes in 2^{N-1} different ways. But because now the encoding of all subcodes is known, the search for a minimal composition (i.e. minimal end code) can be translated to a shortest route problem. Shortest route problems are well known in mathematics and have a solution that only requires a polynomial computing time. This way, we can avoid the exponential process of investigating all possible compositions. As we saw in the previous section, this method has to be applied to every subcode as well. This is possible because the subcodes are investigated in order of growing length, so if a minimal composition has to be determined for a subcode of length 'k', all subcodes of smaller length already have been dealt with. So then we have exactly the same situation as for the whole code after all subcodes have been investigated. How the translation to and the solution of the shortest route problem are established, will be illustrated by an extended example in which we use the code and graph of Fig. 3 as a starting point. As we just argued, exactly the same procedure would be necessary if that code would be a subcode of a larger code.

In Fig. 3, a directed graph on six vertices is related to the code 'a a b a b'. Every route from vertex 0 to vertex 5 represents a partitioning of the code into several subcodes. An edge $e(i, j)$, with $i < j$, from vertex i to vertex j , represents subcode $e [i + 1, j]$. Here, subcode $[i + 1, j]$ refers to the subcode consisting of the code elements numbered from $i + 1$ to j inclusive.

We can define $w(i, j)$ as the "length" of edge $e(i, j)$ by:

$$w(i, j) = \text{load of subcode } [i + 1, j].$$

Now, we use the loads as found after the encoding of each subcode separately. In this case, six subcodes are covered by one ISA-form each, which yields:

- $w(0, 2) = 1$ from $2 * (a)$
- $w(1, 4) = 2$ from $S[(a), (b)]$
- $w(2, 5) = 2$ from $S[(b), (a)]$
- $w(0, 4) = 3$ from $S[(a), (ab)]$
- $w(1, 5) = 2$ from $2 * (a b)$
- $w(0, 5) = 3$ from $\langle (2 * (a)) (a) \rangle / \langle (b) \rangle$.

Subcodes of length greater than 1, which are not covered by one ISA-form, can be disregarded because they always can be described by the six forms above and the single elements. For instance, subcode 'a a b' only can be described by '2 * (a) b' which corresponds to the edges $e(0, 2)$ and $e(2, 3)$. So, subcode 'a a b' itself can be disregarded, which means edge $e(0, 3)$ can be left out of the graph. The resulting graph is shown in Fig. 6.

Going along $e(i, j)$ is equivalent to taking (the encoded form of) the related subcode as part of a code. Each possible end code is represented in the graph by a route from vertex 0 to vertex 5. Though not every route necessarily represents an end code, it will be clear that a shortest route by definition does represent a minimal end code, for the "length" of a route is equivalent to the "load" of the corresponding code.

Now that we have established the translation of the original problem to a shortest route problem, we have to solve this new problem. We will discuss its solution in terms of the graph of Fig. 6 and will not bother the reader with the actual implementation in our algorithm, which is rather easy to realize once you know how the method works.

The solution of a shortest route problem falls apart

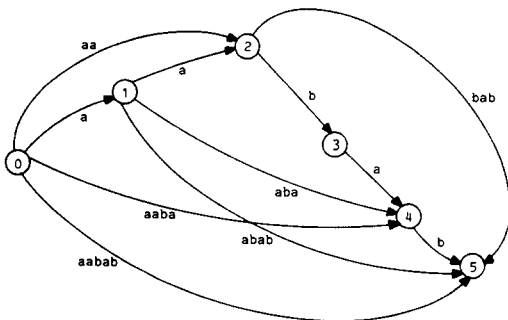


Fig. 6. Graph representing single elements and covering ISA-forms.

into two steps. First we try to determine the minimal distance between vertex 0 and vertex 5, without bothering about an actual route with that distance. This minimal distance is determined in such a way that afterwards a route with that distance is obtained quite easily.

At each vertex k , we determine the minimal distance $w_{\min}(k, 5)$ to vertex 5. We will start at vertex 5 and work backwards to vertex 0. The first value clearly will be:

$$w_{\min}(5, 5) = 0$$

Between vertex 4 and vertex 5, there only is edge $e(4, 5)$. So $w_{\min}(4, 5) = w(4, 5) = 1$. This last equation can be given more general by:

$$w_{\min}(4, 5) = w(4, 5) + w_{\min}(5, 5) = 1.$$

The extension seems rather superfluous here, but it reflects the general idea of using the values already found.

From vertex 3, we only can go to vertex 4, so:

$$w_{\min}(3, 5) = w(3, 4) + w_{\min}(4, 5) = 2.$$

Note we do not bother about an actual route from vertex 4 to vertex 5, we just use the minimal distance already found.

At vertex 2, a new situation occurs, for there are two possibilities to leave from vertex 2. Therefore we have to determine $w_{\min}(2, 5)$ out of two values:

$$w_{\min}(2, 5) \text{ is } w(2, 3) + w_{\min}(3, 5) = 3$$

$$\text{or } w(2, 5) + w_{\min}(5, 5) = 2.$$

So we will find: $w_{\min}(2, 5) = 2$. Now, also another aspect becomes important. If we were "walking" through the graph from vertex 0 to vertex 5 and if we would pass by vertex 2, we now know the best continuation would be edge $e(2, 5)$ and not $e(2, 3)$ for this last one would result in a longer distance. Therefore, already now we can remove edge $e(2, 3)$ from the graph, for it will never be part of a shortest route.

At vertex 1 a similar situation occurs with three possibilities:

$$w_{\min}(1, 5) \text{ is } w(1, 2) + w_{\min}(2, 5) = 3$$

$$\text{or } w(1, 4) + w_{\min}(4, 5) = 3$$

$$\text{or } w(1, 5) + w_{\min}(5, 5) = 2.$$

So here we find $w_{\min}(1, 5) = 2$. Furthermore, the edges $e(1, 2)$ and $e(1, 4)$ will never be part of a shortest route, so they can be removed. Finally, at vertex 0 we have to determine $w_{\min}(0, 5)$ out of four possibilities:

$$w_{\min}(0, 5) \text{ is } w(0, 1) + w_{\min}(1, 5) = 3$$

$$\text{or } w(0, 2) + w_{\min}(2, 5) = 3$$

$$\text{or } w(0, 4) + w_{\min}(4, 5) = 4$$

$$\text{or } w(0, 5) + w_{\min}(5, 5) = 3.$$

It will be clear that $w_{\min}(0, 5) = 3$ and that edge $e(0, 4)$ can be removed from the graph. The resulting graph is shown in Fig. 7.

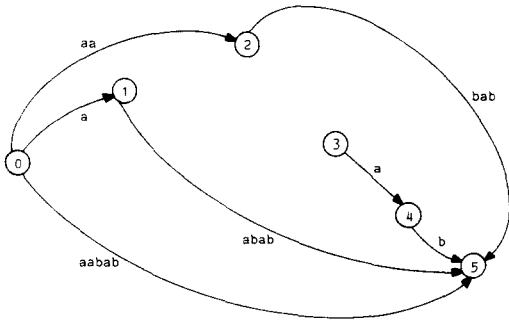


Fig. 7. A shortest route represents a minimal end code.

By working backwards from vertex 5 to vertex 0 and by determining at each vertex the minimal distance to vertex 5, in the end we know the minimal distance from vertex 0 to vertex 5 is $w_{\min}(0, 5) = 3$. As said before, the “length” of a route is equivalent to the “load” of the corresponding code. So we already know the minimal load of an end code is $P = 3$, though we do not know such an end code yet. But because at each vertex we removed the edges that could not be part of a shortest route, every route from vertex 0 to vertex 5 now will be a shortest route and therefore represents a minimal end code. In this example there are three minimal end codes with load $P = 3$:

- (1) $2 * (a) S[(b), (a)]$
- (2) $\langle (2 * (a) (a)) \rangle \langle (b) \rangle$
- (3) $a 2 * (a b)$.

In fact the first two are equivalent, because both yield the same abstract code: ‘ $x x y z y$ ’, while the third one yields: ‘ $x y z y z$ ’. So, according to the structural information approach, the original pattern ‘ $a a b a b$ ’ will be ambiguous, for there are two different but equally preferred organizations. Here we will end with the example.

If we apply the shortest route method to a code of length N , we observe the following. After the translation has been established, we have to work backwards from vertex N to vertex 0. At each vertex k , we have to determine:

$$w_{\min}(k, N) = \text{MIN} \{w(k, p) + w_{\min}(p, N),$$

for all $e(k, p)\}$.

The number of edges $e(k, p)$ maximally is $N - k$, which implies the computing time to find the minimal distance from vertex 0 to vertex N is $O(N^2)$.

The last question is which minimal end code has to be selected out of the possible minimal end codes. The structural information approach yields no preference for one of them. However, we would like to make a justifiable choice. Therefore we observe the following. The minimum principle is considered to be a perceptual tendency, which can be influenced by several non-structural aspects of perception. Some of these aspects are color and brightness⁽¹⁹⁾ or proximity and locality.⁽²⁰⁾ These factors are independent of structural

information and they play their own role in perception. Therefore the perceptually preferred code can be something else than just the structural minimum. Consider the code ‘ $a b c a b c b c$ ’. The perceptually preferred organization seems to be ‘ $2 * (abc) b c$ ’ or ‘ $S[(a), (bc)] 2 * (bc)$ ’, both with $P = 5$. However, the structural minimum is ‘ $\langle 2 * ((ab) (b)) \rangle \langle (c) \rangle$ ’ with $P = 4$. In this case, locality interferes with structural aspects. Under the influence of locality, subjects tend to minimize parts of the code, disregarding a possible global minimum. Mostly this implies a partitioning of the code into several independent subcodes. Therefore, locality is also related to the question whether a pattern is perceived as just one pattern or falls apart into several subpatterns, each of which is considered independent of the others.

In view of the observations above, we select among the possible minimal end codes the one with a maximal number of atoms. This incorporates a tendency to locality. Not that locality as such is dealt with, for this strategy leads to a solution which, in the first place, reflects a global structural minimum. But it also leads to locally oriented organizations of the pattern. In the shortest route example above this implies either the first or the third minimal end code is selected, each consisting of two atoms. Not the second one which consists of only one atom. This selection involves a longest route problem, the solution of which in this case (a special kind of directed graph) is easily added to the process. It is quite similar to the solution of the shortest route problem and also takes $O(N^2)$ computing time. So, after all subcodes have been investigated separately, the computing time to obtain a minimal end code with a maximal number of atoms is $O(N^2)$.

MEMORY SPACE

Beside the computing time, another interesting aspect of the algorithm is the memory space, needed during the encoding. A primitive code representing a simple visual pattern, quite easily consists of about 30 elements. So it is relevant whether the algorithm can deal with codes of such lengths. In Fig. 5, we observed that type and load are stored for each investigated subcode. To find a minimal composition for a subcode or a minimal end code, we only need the loads of the encoded subcodes, not their structures, for only the load is the criterion on which a structure is selected. Furthermore, only a few structures will be selected for a minimal end code. Therefore we do not store the structures themselves, only their type and load. After we have selected the structures for the end code on the basis of their loads, we can easily reconstruct these few structures, for we already know their type. This implies that beside some space to store structures temporarily while they are being investigated, we only have to store the matrix of Fig. 5. Using the same notation we used to indicate computing times, the storage space needed during the total encoding is $O(N^2)$. This holds even if we take into account the

storage space for similar matrices during the encoding of argument codes on higher hierarchical levels. The practical implication of this can be illustrated by the following. Without storage into files, on a PDP-11/44 (16-bit computer, available space 55 Kbytes), all possible codes of length up to $N = 85$ can be encoded. On a VAX-11/750 (32-bit computer, available space 6 Mbytes) there is space enough to encode all codes of length up to $N = 640$.

The algorithm has been implemented and tested on the two computer systems just mentioned, both under the UNIX operating system. It is written in the C programming language, which has a close relationship with UNIX; together they provide a convenient use of pointers to represent and store hierarchical structures.

CONCLUSION

In this paper we presented an encoding algorithm, which computes minimal end codes of patterns on the basis of a well defined and operational model for the structural information approach. This approach supports the minimum principle, which reflects a strong perceptual tendency. But it is not the only factor relevant to perception. How other perceptual aspects like locality interfere with structural information, will be an interesting subject for future research.

Our main goal was to avoid an exponential growth of computing time in finding a minimal end code. The structural information approach gives rise to two explosive searches. One concerns the number of possible covering A-forms for a subcode. To avoid this explosion we used heuristics. Whether these heuristics are well chosen or whether they are really necessary, has to be decided by further perception research. The main explosion concerns the number of possible end codes. To avoid this explosion, the problem of finding a minimal end code has been translated to a shortest route problem, which only takes a polynomial computing time to solve it. In order to make this translation possible, all subcodes of the original primitive code had to be investigated separately. The investigation of the subcodes also takes just a polynomial computing time. This way, instead of an explosive computing time of $O(2^N)$, the algorithm just takes an acceptable time of $O(N^4)$.

The structural information approach states, in agreement with the minimum principle, that a simplest pattern code reflects the preferred pattern organization. This assumption would not be very realistic if it would imply explosive searches to attain such a simplest code. Therefore the algorithm can be considered as a support to the structural information approach and to the minimum principle.

SUMMARY

In the coding theory of pattern perception, there are several points of view regarding the question which organization of a pattern reflects the preferred organization.

Some consider the most regular organization as the preferred one, others the most probable organization or the one which is attained with a minimum of procedural steps. The structural information approach⁽¹⁴⁾ states that the simplest organization is the preferred one, which is in correspondence with the minimum principle.⁽¹⁶⁾ So not the process itself is decisive, but only the final outcome of the process.

The structural information approach provides a set of reduction rules to describe regularities in a pattern. These rules explicitly express identity relations in a pattern code. In general not all identity relations can be expressed at the same time in one code. The preferred organization of a pattern is reflected by a code which expresses a maximum number of identity relations. However, the number of possible codes grows exponentially with the complexity of the pattern. So in computer simulation, it would take an exponential computing time to generate all codes and then select the simplest one.

One attempt to avoid the explosion of code generation and computing time has been made by Collard and Buffart.⁽¹²⁾

In this paper another method is proposed. The central part of this method consists of translating the search for a simplest code to a shortest route problem. After applying the reduction rules to every part of the pattern code separately, the simplest code can be found without generating all possible codes. This way the search for a simplest code only takes a polynomial computing time.

The structural information approach describes patterns in terms of linear codes. In the computer simulation, a hierarchical representation of these codes and a data structure are developed. These are not crucial in avoiding explosive searches, but they are important in that they constitute a formal framework in which it is easy to deal with codes and identity relations.

In one part of the encoding algorithm, heuristics were used to avoid a second explosive search (beside the one for the end codes). Whether these heuristics were well chosen or whether they are necessary at all, has to be decided by further perception research. Also further research has to be done concerning the question how other non-structural aspects of a pattern interfere in attaining a preferred pattern organization.

REFERENCES

1. K. Koffka, *Principles of Gestalt Psychology*. Harcourt, Brace & World, New York (1935).
2. F. Attneave, Some informational aspects of visual perception, *Psychol. Rev.* **61**, 183-193 (1954).
3. W. R. Garner and D. E. Clement, Goodness of patterns and pattern uncertainty, *J. verb. Learn. verb. Behav.* **2**, 446-452 (1963).
4. L. Uhr, Pattern recognition computers as models for form perception, *Psychol. Bull.* **60**, 40-73 (1963).

5. H. A. Simon and K. Kotovsky, Human acquisition of concepts for sequential patterns. *Psychol. Rev.* **70**, 534–546 (1963).
6. C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, Urbana (1949).
7. H. von Helmholtz. *Handbuch der physiologischen Optik*, 3. *Die Lehre von den Gesichtswahrnehmungen*. Voss, Leipzig (1866).
8. I. Rock. *Logic and Perception*. MIT-Bradford, Cambridge (1983).
9. R. L. Gregory and J. P. Harris. Illusory contours and stereo depth. *Percept. Psychophys.* **15**, 411–416 (1974).
10. F. Restle. Coding theory of the perception of motion configurations. *Psychol. Rev.* **86**, 1–24 (1979).
11. H. van Tuijl. Perceptual interpretation of complex line patterns. *J. exp. Psychol. Hum. Percept. Perform.* **6**, 197–221 (1980).
12. R. Collard and H. Buffart. Minimization of structural information: a set-theoretical approach. *Pattern Recognition* **16**, 231–242 (1983).
13. H. Buffart & H.-G. Geissler. Task-dependent representation of categories and memory-guided inference during classification. *Trends in Mathematical Psychology*. E. Degreef and J. van Buggenhaut, eds. North-Holland, Amsterdam (1984).
14. E. Leeuwenberg. A perceptual coding language for visual and auditory patterns. *Am. J. Psychol.* **84**, 307–349 (1971).
15. H. Buffart, E. Leeuwenberg & F. Restle. Coding theory of visual pattern completion. *J. exp. Psychol. Hum. Percept. Perform.* **7**, 241–274 (1981).
16. J. E. Hochberg and E. McAllister. A quantitative approach to figural "goodness". *J. exp. Psychol.* **46**, 361–364 (1953).
17. G. C. Hatfield and W. Epstein. The status of the Minimum Principle in the theoretical analysis of visual perception. *Psychol. Bull.* **97**, 155–186.
18. H. Mellink and H. Buffart. Abstract code network as a model of perceptual memory. *Pattern Recognition* (in press).
19. J. Beck. Textural segmentation. *Organization and Representation in Perception*. J. Beck, ed. Lawrence Erlbaum, Hillsdale, NJ (1982).
20. G. Kanizsa. Subjective contours. *Scient. Am.* **234**, 48–52 (1976).

About the Author—PETER VAN DER HELM was born in Nieuwlande, The Netherlands, in 1956. He received his MS degree in Applied Mathematics from the Twente University of Technology in 1981.

Since 1983 he has been a Z.W.O. research fellow of the Structural Information Theory Group at the Department of Experimental Psychology of the University of Nijmegen.

About the Author—EMANUEL LEEUWENBERG was born in Heemstede, The Netherlands, in 1939. He received his MS degree in Mathematical Psychology in 1964 and his Ph.D. in Social Sciences from the University of Nijmegen in 1967.

After a year at Carnegie-Mellon University, U.S.A., he became head of a special research project on "Structural Information Theory" at the University of Nijmegen.