MDLChunker: A MDL-Based Cognitive Model of Inductive Learning

Vivien Robinet, Benoît Lemaire, Mirta B. Gordon

To cite this version:

Vivien Robinet, Benoît Lemaire, Mirta B. Gordon. MDLChunker: A MDL-Based Cognitive Model of Inductive Learning. Cognitive Science, 2011, 35 (7), pp.1352-1389. 10.1111/j.1551-6709.2011.01188.x . hal-00624819

HAL Id: hal-00624819

https://hal.science/hal-00624819

Submitted on 19 Sep 2011

MDLChunker: a MDL-based Cognitive Model of Inductive Learning

Vivien Robinet[1], Benoît Lemaire[1], and Mirta B. Gordon[1]

[1]University of Grenoble

Running Head: Cognitive Model of Inductive Learning

Address correspondence to:

Vivien Robinet

INRIA – Equipe NeuroMathComp

2004, route des Lucioles

06902 Sophia-Antipolis Cedex 2

France

vivien.robinet@inria.fr

Abstract

This paper presents a computational model of the way humans inductively identify and aggregate concepts from the low-level stimuli they are exposed to. Based on the idea that humans tend to select the simplest structures, it implements a dynamic hierarchical chunking mechanism in which the decision to create or not a new chunk is based on an information-theoretic criterion, the Minimum Description Length (MDL) principle. We present theoretical justifications for this approach together with results of an experiment in which participants, exposed to meaningless symbols, have been implicitly encouraged to create high-level concepts by grouping them. Results show that the designed model, called hereafter MDLChunker, makes precise quantitative predictions both on the kind of chunks created by the participants and also on the moment at which these creations occur. They suggest that the simplicity principle used to design MDLChunker is particularly efficient to model chunking mechanisms. The main interest of this model over existing ones is that it does not require any adjustable parameter.

1. Introduction

Humans are able to associate together stimuli that they consider similar because they have adequate mental schemata to represent them. The ability to generate new mental schemata endows individuals with enhanced capabilities in the world where they evolved. When faced with sequences of low-level stimuli, they are able to organize them dynamically according to their latent structure in a way that helps prediction. This clustering mechanism mostly known under the name of "chunking" seems to be a general information-processing tool in perception and learning (Gobet et al, 2001). It is also studied in the literature of implicit learning (Cleeremans, 1997). The challenge of inductive concept learning is to understand how and when a collection of stimuli showing internal regularities are bound together to create a new concept or class.

There are infinitely many ways of organizing the information grasped from the environment, as well as there are infinitely many curves that fit a finite set of data, or an infinity of grammars which cover a set of sentences, etc. As has been thoroughly discussed in the theory of Machine Learning (see Bishop, 2006) the induction problem of finding the best model underlying or explaining a set of data has no trivial solution. The adequate complexity of a model is related to the available amount of data: too complex models may in principle better fit the data, but at the expense of a poorer predictive power. The trade-off between goodness-of-fit and predictability is known as the bias-variance dilemma. We believe that humans put into practice general mechanisms to construct and select concepts, which efficiently represent the input stimuli. Our hypothesis, following Chater & Vitanyi (2003), is that humans put a general principle of simplicity into practice to select representations among several possible ones. A normative justification for considering simplicity as a principle underlying cognitive tasks (Chater, 1999) is that among all the possible representations consistent with external stimuli, a cognitive agent should adopt the one with highest predictive power. Simplicity is closely related to compression: the simplest representation of a set of stimuli is the shortest one, with no internal redundancy. A short representation helps comprehension and interpretation of the external world because it contains all the relevant regularities.

Both statistical learning theory (Vapnik, 1998) and Kolmogorov algorithmic information theory (Kolmogorov, 1968) show that predictability is highly dependent on the complexity of data representation. In particular, within the framework of Kolmogorov's theory, the shortest representation (which is the simplest one) has the highest predictive power. The same principle is used in scientific theories: the best model of empirical findings has the lowest number of parameters. From a cognitive point of view, representations that satisfy the principle of simplicity are plausible because they require less resources, provide correct predictions and easy recall. Since algorithmic information theory gives a theoretical justification for selecting short representations, we recast the search of representations with optimal prediction properties into the problem of looking for the shortest representations.

We assume that humans learn new concepts inductively from low-level stimuli through generation and selection of new lexicons or concepts that represent these stimuli in the shortest possible way. These concepts give rise to representations of the external world which are optimal because they use a minimum of resources and have the highest predictive power. Since stimuli are perceived and represented with some initial lexicon, creating new concepts amounts to enhance the lexicon by including new elements that allow for shorter encoding of the old ones. Actually, these new concepts are useful only if they help to better represent new stimuli. In our model, the criterion defining the "goodness" of a concept among competing alternatives is the length of the stimuli representations when that concept is used. Applying this criterion is a very complex task, not only because defining a new concept requires a combinatorial search in the space of all possible combinations of lexicon elements but also because there is a multiplicity of alternative transcriptions of a stimuli in terms of a lexicon.

Our main concern in the present paper is to test the pertinence of creating chunks in accordance with the simplicity principle. In our approach, concepts are hierarchies of chunks, and concept selection is performed using the Minimum Description Length (MDL) principle (Rissanen, 1978). These choices have strong theoretical justifications that we discuss in the section 2.

We use an algorithm borrowed from the information theory literature: the two-part MDL coding. The two corresponding basic processes, generating chunks and selecting the best ones, are implemented in their simplest version, called MDLChunker, discussed in section 3. This implementation does not integrate specific cognitive constraints that are nonetheless highly relevant to human cognition modeling in general, although not to our problem in particular.

Special efforts have been made to keep the model as simple as possible, and especially to avoid free parameters. This is very important to ensure that results depend on the information carried by the model (Sun, 2004) and not on ad-hoc posterior adjustments. Said differently, we are not interested in the ultimate performance that may be achieved through specific refinements of our approach. Our goal is to test the relevance of the association of simplicity and chunking, not the performance of a specific implementation. The MDLChunker produces the quantitative predictions necessary to validate the model through comparison with experimental data (section 4). We compare it with other models, theoretically in section 5 and experimentally in section 6.

In section 7 we present a completely different implementation of the same principles, MDLChunker-cog, that is more plausible from the cognitive point of view, in order to show that the studied principles may be parallelized in a brain-like architecture. The cognitively plausible representations are shown to be compatible with cognitively plausible computations. The performance of MDLChunker-cog was found to be similar to that of the bare MDLChunker on a new experiment. Memory capacity is of central importance in concept learning. Learning of word segmentation, another cognitive task treated in the literature, is shown to be highly dependant on the size of the short term memory. The ability of our model to deal with memory size constraints is investigated in section 7. We discuss why memory size should be measured in terms of concepts' description lengths rather than in numbers of items. In section 8 we summarize our conclusions and propose some lines to guide future investigations.

2. Theoretical background

Our main hypothesis is that humans perform induction, i.e. construct and select representations, by implementing a general simplicity principle (Chater & Vitanyi, 2003). This section presents the theoretical justifications for choosing the simplicity principle and the chunking mechanism as a basis for the design of MDLChunker, presented in section 3. Its validation is described in section 4.

2.1. From the induction problem to the Kolmogorov complexity

Within the Machine Learning framework, the induction problem may be rephrased in terms of the “bias-variance dilemma” (Bishop, 2006): sufficiently complex models can in principle fit every finite dataset (small bias). The counterpart is a poor ability to account for new data (huge variance). Designing a new model brings a solution to this trade-off between predictive power and data fitting. For example, a cyclic phenomenon may be well modelled (i.e. provide good predictions) by a sinusoid even if there are fitting errors which may be attributed to noise in the data. This simple curve may provide better predictions than one without fitting errors. The VC-dimension (Vapnik, 1998) of a model is a theoretical concept that allows quantifying the probability of prediction error in terms of the number of parameters and the amount of data used to determine their values. Unfortunately the VC-dimension is not easy to calculate and in particular it is unknown for the model considered in this paper.

The simplicity principle provides a solution to the induction problem (Chater & Vitanyi, 2003) by preferring the shortest representation among all possible ones. This principle is of particular interest for cognitive modeling because relevant properties can be demonstrated within the Kolmogorov complexity framework. Kolmogorov complexity (Solomonoff, 1960; Chaitin, 1966; Kolmogorov, 1968) gives a formal definition of the intuitive notion of simplicity, without making assumptions about the process that generated the data. The complexity of a given dataset x is defined as the size of the shortest computer program p able to generate x on a Turing machine (Turing, 1936) and then halt. In other words, p is the best compression without loss that can be found for x.

The generality of this measure is guaranteed by the invariance theorem (Solomonoff, 1964): the programming language itself (i.e. the particular Turing machine) does not matter, since all programming languages produce programs of the same size up to an additive constant. The latter only depends on the Turing machine but not on the dataset x. This additive constant represents the size of the emulator able to translate instructions from one programming language to another. The Kolmogorov complexity is a theoretically useful measure of the information carried by a given set of data but, even worse than the VC-dimension (which has been determined for some learning machines, like neural networks), cannot be calculated (Li & Vitanyi, 1997).

One of the most important results in Kolmogorov complexity theory is that the shortest program p for a given dataset x has also the highest probability to make correct predictions about new data issued from the same process as x (Vereshchagin & Vitanyi, 2004). Intuitively, the shortest program contains all the information and nothing but the information necessary to generate the dataset x. Any longer program adds extra-information not supported by the dataset x. This extra-information is, on average, not better than random information which, included in a longer program, would decrease its probability of making correct predictions (Vitanyi, 2005).

Thus, without additional information, the shortest program p for x is the best model for x. This result gives a theoretical solution to the bias-variance dilemma mentioned before. As an example, consider the data {1, 1, 2, 3, 5, 8, 13}. The following representations are both compatible with the data:

- $x_{i+2} = x_{i+1} + x_i \ \ \forall i > 0$ and $x_1 = x_2 = 1$

- $x_i = \frac{1}{144} i^6 - \frac{41}{240} i^5 + \frac{241}{144} i^4 - \frac{395}{48} i^3 + \frac{1535}{72} i^2 - \frac{133}{5} i + 13$

However the first one is preferable because it is shorter. The second one is more constrained because it includes the implicit assumption that the data are generated by a polynomial process: this information is not contained in the data. Considering that {1, 1, 2, 3, 5, 8, 13} are the first seven Fibonacci numbers is the most probable interpretation, making the continuation 21, 34, ... (first case) more probable than 29, 85, ... (second case). Even if the exact Kolmogorov complexity of this dataset is unknown and the two representations are not actual programs for a universal Turing machine, it is easy to see that the first representation is shorter and thus more plausible than the second.

2.2. The chunking mechanism

Chunking may be considered a key mechanism in cognition. It was initially introduced by Miller (1956) to describe the short term memory capacity, that he found to be about seven plus or minus two chunks. In fact, there is no precise definition of what constitutes a chunk. According to Gobet et al. (2001) a chunk is "a collection of elements having strong associations with one another, but weak associations with elements within other chunks". Thus, a chunk clusters together strongly associated elements into a single unit. A chunk may be considered as a concept facilitating recall but it does not correspond to a fixed quantity of information. Chunks allow the cognitive system to represents complex information as collections of elementary units, which may be chunks themselves. The process of chunking has to solve two underlying problems: what elementary units should be clustered together into a chunk, and when a new chunk is worthy to be created.

For instance, when exposed to the flow of phonemes of their language, toddlers group phonemes that tend to occur together, forming syllables and then words. Words can thus be seen as chunks of syllables which are themselves chunks of phonemes (see Fig. 1). Sequences of words that often occur jointly can also be clustered to form higher-level chunks.

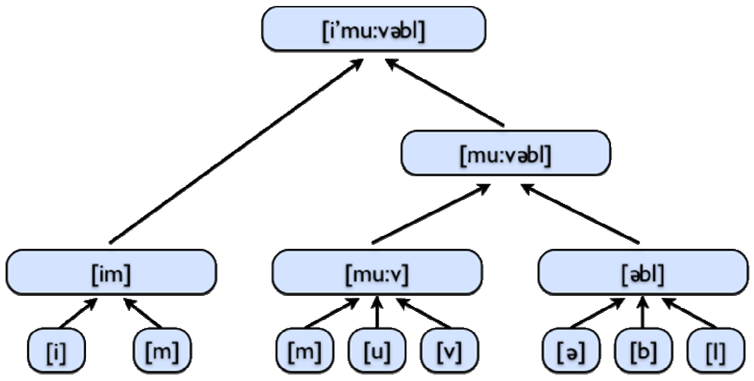


Fig. 1. Example of phoneme-based chunks that are hierarchically organized using prefixes and suffixes to form the syllables, and then the word "immovable".

In MDLChunker, concepts are represented by chunks and learning new concepts means creating new chunks. The simplicity principle may be applied to any kind of regularities. Because the regularities captured depend on the representation language, it is important to choose those representations carefully, and we have chosen representations that have strong cognitive justification. Languages having the expressiveness of a Turing machine are able to represent all the regularities expressible through an algorithmic procedure (Turing, 1936). In terms of language representation, chunks may be considered as a "minimal expressiveness" language. They are simple conjunctions of elements, such as:

- chunk ABC = A and B and C

However, a hierarchy of chunks can still represent a large variety of concepts. Chunks are broadly used and studied in cognitive psychology. They serve as a basis of several cognitive computational models such as EPAM (Feigenbaum & Simon, 1984), CHREST (Gobet, 1993), Competitive Chunker (Servan-Schreiber & Anderson, 1990), PARSER (Perruchet & Vinter, 1998), etc. A comparison of MDLChunker with some of them is presented in section 5.

2.3. The Minimum Description Length principle

Kolmogorov complexity-based approaches rely on description languages that have the expressiveness of Turing machines. However, a cognitive agent is not able to extract all the kinds of regularities that may be represented with a Turing machine (van Rooij, 2008; Oaksford & Chater, 1993; Chater, 1996). For example, it is easy for a human to interpret the 30 bits 001001001001001001001001001001 as 10 repetitions of "001", whereas it is difficult to interpret 110010010000111111011010101000 as the first thirty bits of π. Clearly, in the first case, there is a chunking mechanism at work: the regularities in the data can be recognized because the chunks "001" may be identified. Instead of the initial binary lexicon used to encode the data, the new representation contains a single chunk denoted "001".

In the particular case we are interested in, where regularities are chunks, the minimal Shannon information measure of the data is a computable approximation of Kolmogorov complexity according to Information Theory (Leung-Yan-Cheong & Cover, 1978)). Information Theory assumes that the data are realizations of a random variable X that takes values in a lexicon according to some probability density $P(x_i)$. The amount of information $I(x_i)$ carried by a given value x_i (an element of the lexicon) of the random variable depends on the probability of its occurrence $P(x_i)$ through the following equation (Shannon, 1948).

$$I(x_i) = -\log_2 P(x_i) \quad (1)$$

According to Shannon information theory, the shortest code for the random variable X has a length given by (1): the most probable values are given the shortest codes. Thus, the minimal length (in bits) required in average to encode one realization of a random variable that take values x_i drawn with probability $P(x_i)$ is:

$$L(X) = -\sum_i P(x_i) \log_2 P(x_i) \quad (2)$$

This intensive quantity is called entropy. Approximating probabilities by the corresponding frequencies, the shortest encoding of any realization of N data has a length[1]:

$$L_{\min}(x) = -\sum_i n_i \log_2 \frac{n_i}{N} \quad (3)$$

where n_i is the number of occurrences of the value x_i in the dataset x. This is the Shannon-Fano coding of the data. Whenever a dataset x may be considered as a set of N independent and identically distributed realizations of a random variable X, the frequency of each value x_i may be used to approximate its probability. Then, the optimal (shortest) representation of the data is given by equation (3). Notice that this does not give us the representation code, but only its size. The optimal representation code uses words of length $I(x_i)$ to encode the realization x_i whose frequency is $\frac{n_i}{N}$.

Consider for example the set "A B A A C B A D" which needs 16 bits to be naively encoded (because at least 2 bits per symbol are needed to encode four different symbols). Notice that equation (3) applies to this set and gives $L_{\min}$ = 14 bits. Using the frequencies of each symbol, we may encode them differently with codelengths given by equation (1). The optimal codewords[2] are: A: 0, B: 10, C: 110 and D: 111. They allow the system to compress the original data to give the following 14 bits representation: "0 10 0 0 110 10 0 111"[3]. This is an example of the usual application of the Shannon-Fano encoding algorithm to data compression.

Consider now a dataset showing frequency regularities and chunks (such as "XYZ B XYZ XYZ C B XYZ D"). A naive encoding requires 2.6 bits per character for the six letters alphabet (in fact, with 3 bits we can encode up to 8

---

[1] This is an approximation to the extent that frequencies are approximations of probabilities.

[2] The notation "Z: z" indicates that z is the codeword for Z.

[3] The codewords being self-delimiting, we use spaces for understandability reasons.

characters, but 2 bits are not enough to encode distinctly 6 characters). Thus, we need 41.6 bits to encode the data using this naive encoding. The optimal Shannon-Fano code takes advantage of the frequency of each symbol to encode the more frequent ones with fewer bits. According to (3), the minimum length needed to encode these data is 38 bits. Now observing that symbols X, Y, Z are frequently associated, we may replace them by a chunk A: XYZ. Then the data are represented by "A B A A C B A D" whose minimum representation length is 14 bits, as shown before. Creating the chunk "A" has a cost, which we measure by the length of its definition: 8 bits (2 bits[4] per symbol). We add the chunk definition length to the representation length in order to obtain the total description length of the data which is thus 22 bits. We see that introducing new concepts (the chunks) allowed us to further compress the data. Transforming "XYZ B XYZ XYZ C B XYZ D" into "A B A A C B A D" and "A: XYZ" saves 16 bits. The hypothesis that chunks are the only regularities present in the dataset ensures that the string obtained after defining the chunks can be seen as independent realizations of a random variable. The optimality of the resulting compression only depends on the optimality of the chunking process.

The above example shows that when looking for the best representations we need to identify which symbols should be grouped together to form the chunks. The different representation sizes are evaluated using equation (3). Chunks are generated whenever they reduce the overall length of the stimuli description. The latter is composed of two parts: the chunk definition part which serves as a lexicon and the stimuli description given the chunk definitions.

Within this paradigm, the best description of a dataset is the shortest one, written in a programming language involving chunks. Moreover, the Minimum Description Length (MDL) principle (see Grünwald et al, 2005 for a general overview of MDL) is utilized to find in practice the shortest description of the data. Notice that the rationale to look for the shortest encoding of input stimuli is the search for good predictability and not that of saving memory, which is a (probably useful) side effect. The two-part coding version of the MDL principle (Rissanen, 1978) is based on the probability decomposition (Bayes rule):

[4] In this case, each symbol costs 2 bits because there are four different symbols in the chunk definition: A, X, Y and Z. These codelengths evolve as other chunks are created.

$$P(stimuli, chunks) = P(stimuli \mid chunks) \cdot P(chunks)$$

which, once introduced into (3) shows that the information carried by the data is split into two parts: the chunks information and the stimuli information given the chunks. The chunk information captures the regularities in the data while the data information given the chunks refers to the information carried by the data when the latter are encoded using the chunks. Finding the shortest size for the overall description (stimuli representations and chunks definitions) is equivalent to minimizing together the size of the two parts.

3. MDLChunker

MDLChunker described in this section is a raw implementation of the MDL principle applied to chunking mechanisms. While the MDL principle is a powerful method to solve the induction problem because it provides a way of comparing the lengths of different representations, it gives no hints on how to generate them. In the absence of other information, the only regularities that we consider to cluster data together into useful concepts (the chunks) are the frequencies of the elements: elements appearing often together may be grouped and replaced by a chunk with a more compact representation.

Clearly, looking for and selecting the best chunks is a combinatorial problem, because one needs to compare the overall description length of the stimuli for each possible group of symbols that may constitute chunks. MDLChunker assumes that only pairs of symbols may be grouped together to constitute new chunks. Although it is not possible to create n-ary chunks in general without specifying an upper bound for n because of the combinatorial problem, in principle it is possible to create n-ary chunks with any arbitrary value of n through hierarchical aggregation of binary chunks. Clearly, the short term memory capacity gives an upper bound for n. Without precise results about the arity of chunks that humans are able to create, we restrict MDLChunker to only consider binary chunks. There is some evidence (detailed in section 7) that humans proceed that way.

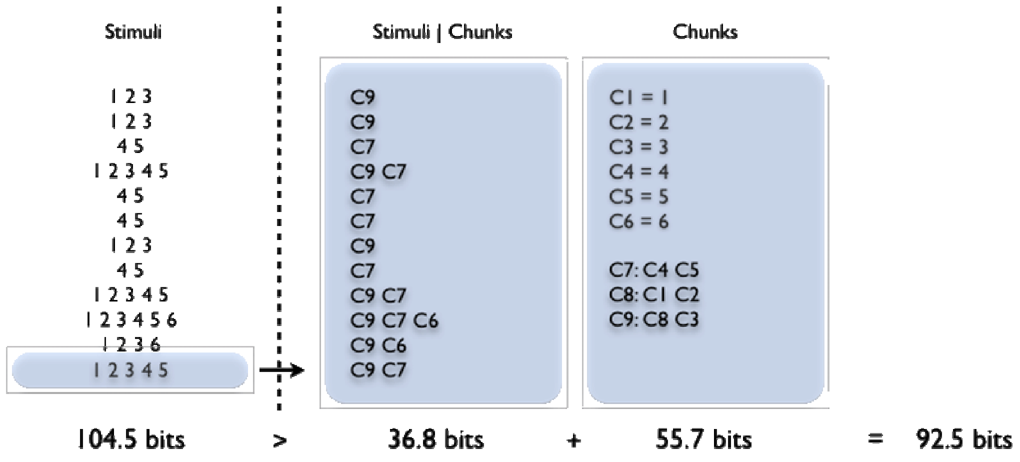
3.1. How MDLChunker works

This section describes the simplest version of MDLChunker. Some points may appear cognitively implausible to the reader because we have neglected cognitive constraints. In particular, it has no memory limits or time constraints.

The goal is to show how far we may go without introducing any specific hypothesis about human cognitive limits. More plausible versions from the cognitive point of view are presented in section 7 where we show that the results are similar.

MDLChunker generates a hierarchy of chunks through an exploration algorithm based on the MDL principle. It guides the search of the chunks that allow for the shortest representation of the perceived stimuli. The regularities found in the data are organized into chunks, which are then added to the lexicon used to encode the stimuli into a compressed representation. The MDL principle operates as a control mechanism of chunk creation: a new chunk can be accepted only if the resulting description length of the stimuli (including the chunks definitions) becomes shorter.

Inputs are sequences of stimuli. MDLChunker represents them using sets of initial symbols or features called hereafter canonical chunks (in the example of section 2, this is the initial alphabet used to encode the raw data), and creates higher level chunks by hierarchically merging lower level chunks. The higher level chunks are expressed in terms of unordered conjunctions of lower level chunks. Because our system is intended to be general and applicable to different domains, the initial features or canonical chunks obviously depend on the specific domain[5].



[5] This is a standard assumption required by many other models such as SP (Wolff, 2006), PARSER (Perruchet & Vinter, 1998), Competitive Chunker (Servan-Schreiber & Anderson, 1990), etc.

Fig. 2. Example of the way extracting chunks can compress information brought by the incoming stimuli. Chunks C1 to C6 are the canonical chunks allowing a raw description of the stimuli. Chunks C7 to C9 are used to compress redundant information because C4, C5 and C1, C2, C3 appear frequently together. The codelengths of the different parts are given at the bottom.

Through the learning process, information contained in each new stimulus is re-expressed in terms of the above mentioned two parts: definition of the already learned chunks and representation of the stimuli given the chunks' definitions, hereafter denoted "Stimuli|Chunks" (see Fig. 2). As a result, the information shared by a large enough number of stimuli is stored as Chunks while the Stimuli|Chunks part contains the information encoded through new chunks. According to the formalism described section 2, the chunks contain compressed information[6] which we interpret as the induced concepts learned from the stimuli, while the Stimuli|Chunks part contains the stimuli representations in terms of these concepts. This learning procedure changes the representation of the original data, but neither adds nor removes information. The two parts are incrementally updated taking into account each new stimulus, which is processed by applying a three-step procedure:

(1) Updating codelengths: each chunk is assigned a codelength defined by equation (1), according to the frequency of its use in the Chunks and in the Stimuli|Chunks parts.

(2) Factorization: the current stimulus is factorized to find its shortest encoding using the existing chunks. Its representation is then included in the Stimuli|Chunks part.

(3) Optimization: new chunks are created in the Chunk part whenever they allow to decreases the overall description length, i.e. the Chunk plus the Stimuli|Chunks lengths.

3.2. Example

Initially, before any stimulus has been perceived, the Stimuli|Chunks part is empty and the Chunk part contains the canonical chunks which are the features used to encode the arriving stimuli. Fig. 3 illustrates how MDLChunker

[6] Information that can be compressed in terms of chunks. The model is blind to other regularities.

works on the simple dataset of Fig. 2, where the canonical chunks are C1, C2, C3, C4, C5 and C6 corresponding to features 1, 2, 3, 4, 5 and 6. The first nine stimuli are trivially encoded using the corresponding canonical chunks C1 to C6 (first pane). At the 9th step, the conjunction of C4 and C5 is frequent enough to be merged into one single chunk: this is the optimization process. Adding C7: C4 C5 to the Chunk part increases the description length by 12.7 bits. However, once the corresponding co-occurrences of C4 and C5 are replaced by C7 in all the existing representations, the description length of the Stimuli|Chunks part decreases by 16.4 bits. The overall description length is decreased by 3.6 bits (second pane). When the next stimulus is presented, its encoding given the chunks is no more unique: it may be either C1 C2 C3 C4 C5 C6 or C1 C2 C3 C7 C6. The factorization process is used to find the shortest one (here C1 C2 C3 C7 C6 whose length is 13.1 bits) (last pane).

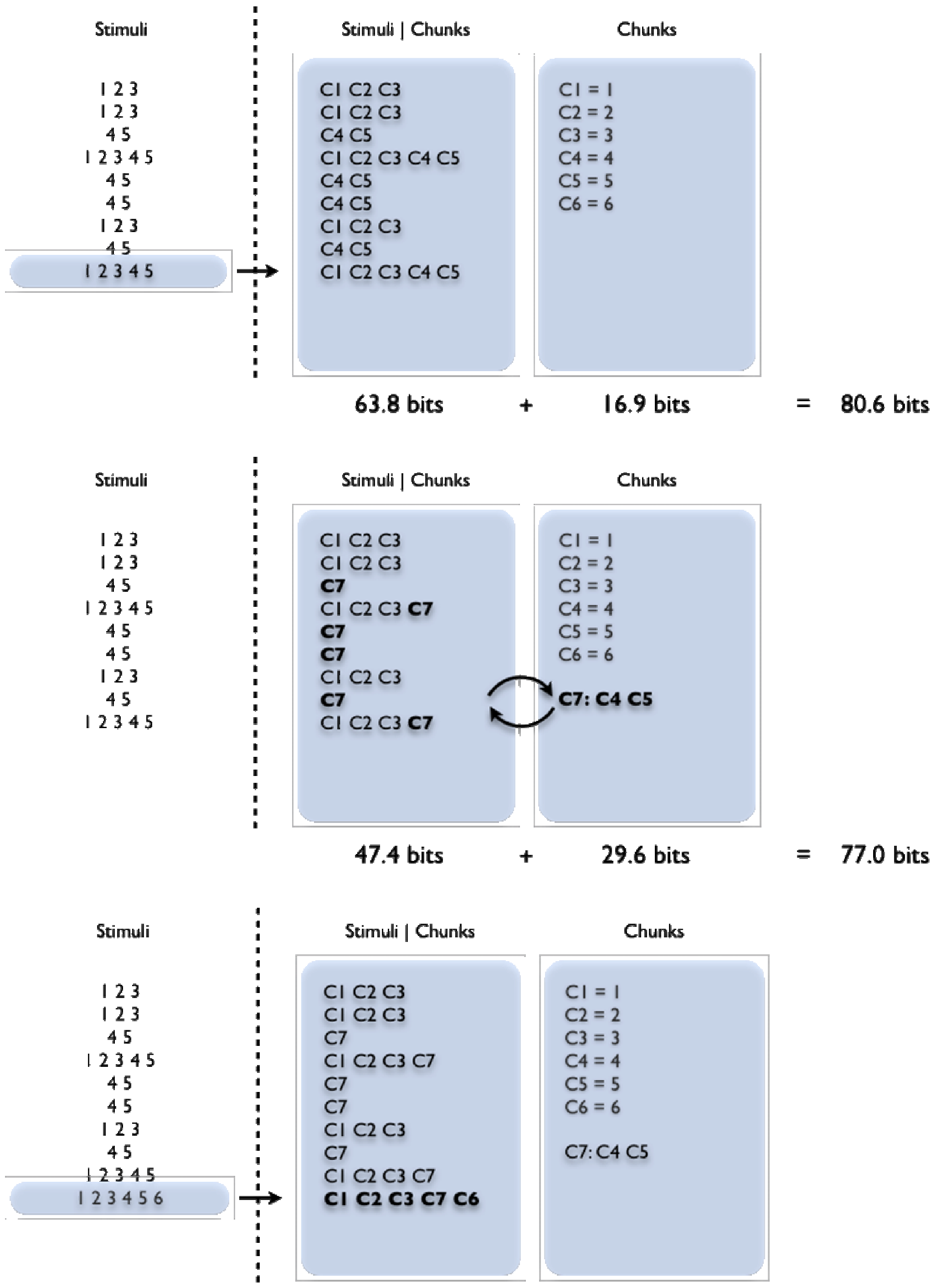


Fig. 3. The first pane shows the codelengths of the two parts after the 9th stimulus (1 2 3 4 5) is perceived. The second one presents the corresponding optimization phase that creates chunk C7: C4 C5 in the Chunk part

and updates the Stimuli|Chunks part accordingly. The third pane illustrates the factorization phase of the 10th stimulus (1 2 3 4 5 6) that gives raise to the representation C1 C2 C3 C7 C6.

We now describe the three processes involved in MDLChunker learning process: the updating of codelengths, the factorization and the optimization processes.

3.3. Updating codelengths

The optimal codelength of each chunk is computed using equation (1), where $P(C_i)$ is the probability that chunk Ci appears in the two parts, Chunks and Stimuli|Chunks. This is equivalent to the probability of Ci in a string composed of the concatenation of the two parts. As opposed to the example of section 2, the contents of the two parts are supposed to be generated by the same probability distribution. In Fig. 2, chunk C9 appears 8 times in the Stimuli|Chunks part and 1 time in the Chunk part. The optimal codelength corresponding to this probability of 9/33 is $I(C9)$ =1.9 bits[7].

3.4. Factorization

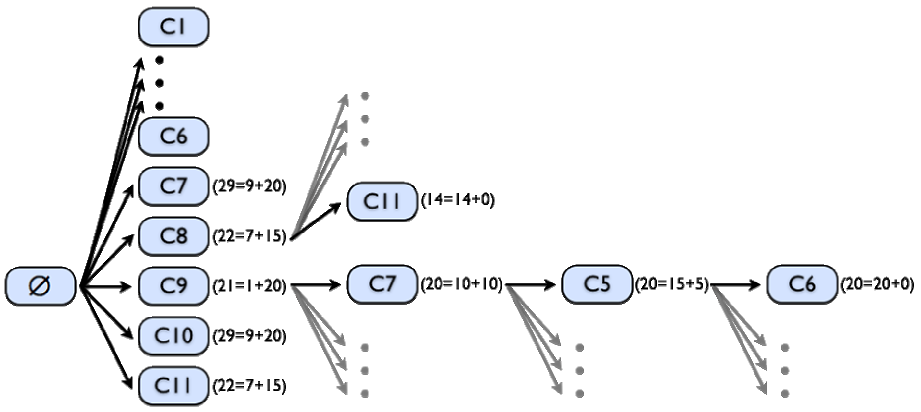
This process corresponds to an encoding of the input stimuli described using only the canonical chunks into a new representation in terms of the extended alphabet containing all the Chunks. Among all possible encodings, the aim is to find the shortest one. The factorization of each stimulus is generally not unique, and finding the shortest encoding is not trivial when chunks overlap. This problem is akin to splitting a sequence of phonemes into words: sometimes several solutions exist, especially if the signal is noisy. For example, "grapes and blackberry starts" and "grape sand black bear restarts" are two ways of chunking the same sequence of phonemes.

In MDLChunker factorization is implemented by means of a heuristic search in the space of all possible combinations of chunks. The algorithm starts with an empty node and progressively adds chunks until it fully constructs the stimulus representation. The search is guided by the distance (in the codelength sense) between the

[7] MDL is used as a selection criterion: it is not necessary to find codewords having this optimal codelength, and using real numbers is not problematic.

current sequence of chunks and the stimulus representation in terms of canonical chunks. In complicated cases the algorithm may fall into a local optimum and miss the global shortest encoding. The algorithm described here is used to factorize a bounded stimulus. In the case where there is no natural way to split the input data into bounded stimuli, it may be extended (see section 6.3) to segment a continuous stream.

Fig. 4 illustrates the factorization of the stimulus C1 C2 C3 C4 C5 C6 when the system contains already 11 chunks: 6 canonical chunks C1 to C6 of size 5 bits each and the chunks C7: C1 C2 (9 bits), C8: C7 C3 (7 bits), C9: C3 C4 (1 bit), C10: C4 C5 (9 bits) and C11: C10 C6 (7 bits). The current node is expanded, creating potential nodes with any chunks included in the stimulus but not yet applied in the current path (the one starting at the root and reaching the current node). Each potential node is associated a penalty (written in parentheses) equal to the length (in bits) of the path from the root to the node, plus the length necessary to complete the representation using only the canonical chunks. For example C9 is associated a penalty of 1 bit + 4 * 5 bits = 21 bits. The node having the smallest penalty becomes the current node[8] (here C9) and is in turn expanded. The first factorization found is C9 C7 C5 C6 (20 bits), and the second one is C8 C11 (14 bits). The latter is returned as the best representation for this stimulus.



[8] This algorithm is similar to A* (Pearl, 1984). Its optimality is guaranteed only if the chosen penalty (i.e. the heuristic used) underestimates the distance between the current node and the goal. This is not the case here because the penalty has no obvious lower bound.

Fig. 4. The first two factorizations of the stimulus C1 C2 C3 C4 C5 C6 where not all expanded nodes are represented. The penalty used to guide the search is indicated in parentheses. The first term is the cost of the path (in bits) from the root, and the second term is an estimation of the cost necessary to reach the goal.

3.5. Optimization

During the optimization process MDLChunker looks for new binary chunks that decrease the overall description length and adds them to the Chunk part. Given the number of co-occurrences $\#CiCj$ of each pair of chunks Ci and Cj, it is possible to calculate the new description length of the system if the corresponding chunk Ck: Ci Cj was created:

$$DescriptionLengthIncrease(Ci, Cj) = L1 + L2 + L3 + L4 \quad (4)$$

with

$$L1 = \#CiCj \cdot \left[\log_2\left(\frac{\#N+3-\#CiCj}{\#CiCj+1}\right) - \log_2\left(\frac{\#N}{\#Ci}\right) - \log_2\left(\frac{\#N}{\#Cj}\right)\right]$$

$$L2 = \#Ci\bar{C}j \cdot \left[\log_2\left(\frac{\#N+3-\#CiCj}{\#Ci\bar{C}j+1}\right) - \log_2\left(\frac{\#N}{\#Ci}\right)\right] + \#\bar{C}iCj \cdot \left[\log_2\left(\frac{\#N+3-\#CiCj}{\#\bar{C}iCj+1}\right) - \log_2\left(\frac{\#N}{\#Cj}\right)\right]$$

$$L3 = \log_2\left(\frac{(\#N+3-\#CiCj)^3}{(\#CiCj+1)\cdot(\#Ci\bar{C}j+1)\cdot(\#\bar{C}iCj+1)}\right)$$

$$L4 = (\#N-\#Ci-\#Cj)\cdot\log_2\left(\frac{\#N+3-\#CiCj}{\#N}\right)$$

where $\#Ci$ (resp. $\#Cj$) is the number of occurrences of Ci (resp. Cj), $\#Ci\bar{C}j$ (resp $\#\bar{C}iCj$) is the number of occurrences of Ci without Cj (resp. Cj without Ci) and $\#N$ is the total number of occurrences. $L1$ represents the codelength saved by replacing co-occurrences of Ci and Cj with Ck. $L2$ is the increase of codelength for Ci and Cj, due to their lower frequency. $L3$ corresponds to the codelength necessary to define the chunk Ck: Ci Cj, and $L4$ is the decrease of codelength for all the chunks, due to the smaller $\# N$.

All the new chunks that help decreasing the total description length are created, beginning by the one leading to the highest decrease. This implementation is sub-optimal because creating the best chunks first does not ensure global optimality. Another search algorithm, like simulated annealing (Kirkpatrick et al, 1983), may in principle improve the performance of the model. However, the benefit would be null because the overall cognitive plausibility would not improve. Here we would like to make a point about the distinction between plausibility of representations and plausibility of processes. For instance, a plausible representation can be obtained by means of a cognitively implausible search. There is a trade-off between the two: improving the plausibility of representations often means more computation and therefore less plausibility of processes. With MDLChunker, we would like to play it both ways but for the moment we only concentrate on plausibility of representations. Our search process was then selected on a criterion of simplicity. Therefore, any search algorithm that would increase plausibility of representations at a higher expense of implausibility of processes is not worth implementing. In front of two algorithms identically implausible, we keep the most simple.

Only co-occurrences of chunk pairs are considered to make MDLChunker as simple as possible. Consequently, new chunks merge always two old chunks. Since this process is iteratively performed, n-ary chunks are created by simple aggregation of binary chunks. An example can be found Fig. 2 where a ternary chunk C1 C2 C3 is created by merging chunk C8: C1 C2 and chunk C9: C8 C3. Notice that the intermediate chunk C8 serves only to define C9 and is not used in any representation. Its codelength is very high compared to that of C9 and its role is thus very limited. The distinction between binary and n-ary chunks can easily be established using the codelength. The chunks never used to represent the stimuli (i.e. the chunks having a great codelength), only serve to define the n-ary chunks.

Considering n-ary chunks as composed of smaller binary chunks has cognitive justifications. Evidence that humans process in the same way is detailed section 7. The vanishing sub-chunk effect described by Giroux & Rey (2009) favours the hypothesis that humans creates big chunks by aggregating smaller chunks that are then forgotten. This forgetting would correspond in MDLChunker to a codelength increase, as described above for C8 when it is replaced by the bigger chunk C9.

4. Experiment

4.1. Design

The main problem to compare MDLChunker with human performance on concept induction comes from the *a priori* knowledge humans may possess. If MDLChunker and humans do not start to learn on similar grounds, any comparison is flawed. There are two ways of overcoming this problem. The first one is to have infants as participants because of their limited *a priori* knowledge. French et al. (2004) for instance performed a categorization experiment with infants. The second one is to use materials that are unfamiliar to humans (pseudo-words, meaningless symbols, etc...). For instance, Fiser & Aslin (2001) create such an artificial environment composed of grids of unfamiliar symbols organized according to a predefined structure. They show that humans are partly able to learn that structure from repeated exposure to the grids. Our experiment belongs to this second approach with symbols similar to those used by Fiser & Aslin.

Basically, such experiments are based on stimuli generated by means of a grammar designed beforehand. The classical experimental paradigm used in inductive learning (Miller, 1958), called artificial grammar learning (AGL), is based on an artificial grammar producing short sequences of letters. After a learning phase on grammatical sentences, participants are given new sentences and are asked to determine whether they are grammatical or not. Using an artificial grammar is an attempt to limit the effect of participant prior knowledge. However, AGL usually provides very limited data to compare the model to. The various types of model validation that one may consider, sorted by increasing demands are:

- prediction of the percentage of participant correct answers;
- prediction of the precise sequence of participant responses;
- prediction of the chunks created by the participants;
- prediction of the time course of chunk creation.

Most of the time, only the first type is used, thus introducing some bias in the validation process (see Redington & Chater (1996) for a discussion). This is the case for the Competitive Chunker validation (Servan-Schreiber &

Anderson, 1990). However, the percentage of correct answers (or other statistics) is only one of the multiple consequences of the chunks but does not tell us much about the chunks themselves. By using such a limited amount of information from the experimental data (in the Kolmogorov sense), the risk is to validate a wrong model because many models may be compatible with these statistics. In other words, many models may reproduce a percentage of participants' correct answers but few of them are able to reproduce the dynamics of chunk creation. Since the classical AGL paradigm does not allow us to properly validate MDLChunker, we designed a new experiment in which participants have to make explicit the concepts they learn in order to compare them step by step with the chunks created by our system.

Because of the deterministic processes involved in MDLChunker, the model may be considered as a unique virtual participant. It cannot reproduce inter-participant variability. We expect MDLChunker to only capture consistent behavior among participants[9]. A deterministic model inferring both the consistent and the variable chunks would be a very bad predictor of the participant general behavior.

Its aim is to explain typical behaviors and not to predict the variance of such behavior in a population, which is an important information in cognitive modeling. One advantage of deterministic models is that all the information they bring about is extracted in one single run whereas stochastic models require theoretically an infinite number of run. The drawback is that deterministic models cannot capture the variability around this central behavior.

If the variance is due to differences in the environment (in our case, the particular data the individuals are exposed to in the experiment), the results of deterministic models will differ accordingly. One may then determine the variance of the model predictions by applying it to a large number of different trials, which is out of the scope of this paper. Notice however that in cognitive modeling both sources of variance (inter-individual and inter-trial) co-exist, and it would be difficult to disentangle them from each other. In fact, MDLChunker could easily be "stochastized" by introducing an additive Gaussian noise in equation 4. However, in the present paper we do not strive to model the

[9] Generally, results produced by non-deterministic models are averaged in order to capture regularities and make consistent predictions.

complete distribution of responses. Our aim is to determine to what extent the chunking mechanism may be explained using the simplicity principle.

4.2. Material

In our artificial environment participants are repetitively exposed to sets of artificial symbols. They are asked to drag and drop all of them into a given area, which is quite fastidious when done one by one. They are encouraged to group symbols together because their task is then lightened. This may be done at any time by a drag and drop operation. Each group is then represented by a colored circle on the screen. Although grouping takes some time, the benefit is that later these symbols may be dragged all together in a single operation, exactly like what offers the "group" command in a drawing software. There is no restriction on associations, and participants are free to create useful, incomplete or even completely useless groups. No feedback is given about the quality of the associations. Participants are successively presented 75 sets containing a variable quantity of symbols. There are 20 possible symbols which are organized through 11 chunks along a 3-level predefined structure that participants do not know (see Fig. 5). No hint is given about the number of chunks or the number of levels in this hierarchy. Participants are just told that they may create groups and that a group may be dragged and dropped like a single symbol. They are also told that associations of groups and symbols are allowed. Each of the 75 set is constructed in the following way: each of the 11 chunks has a 10 percent probability to be part of the current set, which selects about 10 symbols per set[10]. A random noise generator adds an extra symbol or deletes one of them, about once every 2.5 sets.

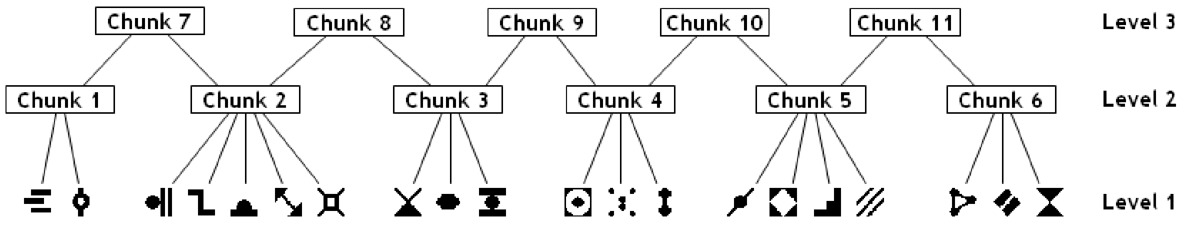


Fig. 5. Hierarchy of the 11 chunks used to generate the 75 sets of the experiment.

[10] Empty sets are removed.

All the participants have been presented the same dataset[11] without time limit to perform the task. However, in order to avoid any accidental association based on the morphology of symbols, they are randomly assigned to chunks at the beginning of each experiment. Fig. 6 shows the interface of our experimental setting in the course of one experiment. Symbols have to be dragged and dropped from the left pane to the empty area next to it. Other areas represent previously seen sets, so that participants do not need to memorize too much information. This is done to avoid differences between participants due to variability in memory capacity. To avoid position and proximity effects, symbols are randomly positioned in each set. The current participant has already created 3 groups (lower left). Each time the symbols occur together, they are marked with the corresponding color and they can be moved in a single operation. A symbol may be marked with different colors because it can belong to several groups.

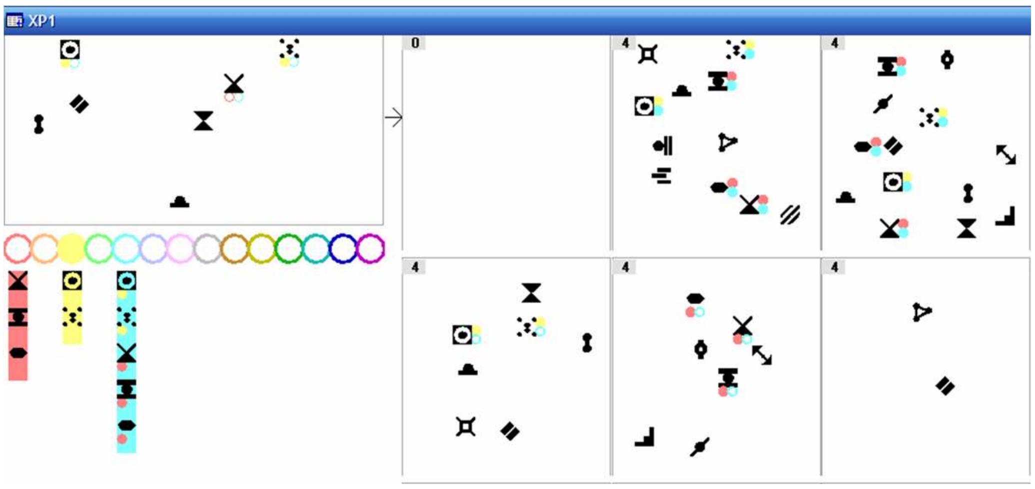


Fig. 6. Interface used for the experiment. The 7 symbols in the current pane (left side of the screen) have to be moved to the empty area. The 2 symbols at the top of the current pane are part of the same group and can be moved together in only one drag-and-drop operation.

4.3. Participants

18 participants were recruited. In order to motivate them, they were told that the appropriate creation of groups would give them points and that a ranking will be established but the grading scale was not provided.

[11] This is of central importance to compare the results.

4.4. Results

The experimental results exhibit some variability among participants: chunks are not exactly the same and are not created at the same iteration. However, the 82 different chunks created by the participants may be classified into two categories:

- 71 chunks created by a minority of participants (non-significant chunks). Most of them (54 chunks) are created by only one participant;

- 11 chunks created by a majority of participants (significant chunks).

The threshold between significant and non-significant chunks is arbitrarily set to one half of the participants. Results are not sensitive to this value and remain unchanged in the range one third to one half. Non-significant chunks are created in average by 1.25 participants, while significant chunks are created in average by 13.5 participants.

MDLChunker was run on the exact same data humans were exposed to. Chunks created by MDLChunker[12] are then compared to the groups created by the participants. Because MDLChunker is intended to reproduce the average human behavior, it is validated according to its ability to create the significant chunks only. The 71 non-significant chunks carry information about the variability of the results around this central behavior, due to participant idiosyncratic characteristics. As already discussed, we do not expect MDLChunker to reproduce this information.

[12] As explained section 3.5, chunks that are only used to define higher order n-ary chunks are not considered here.

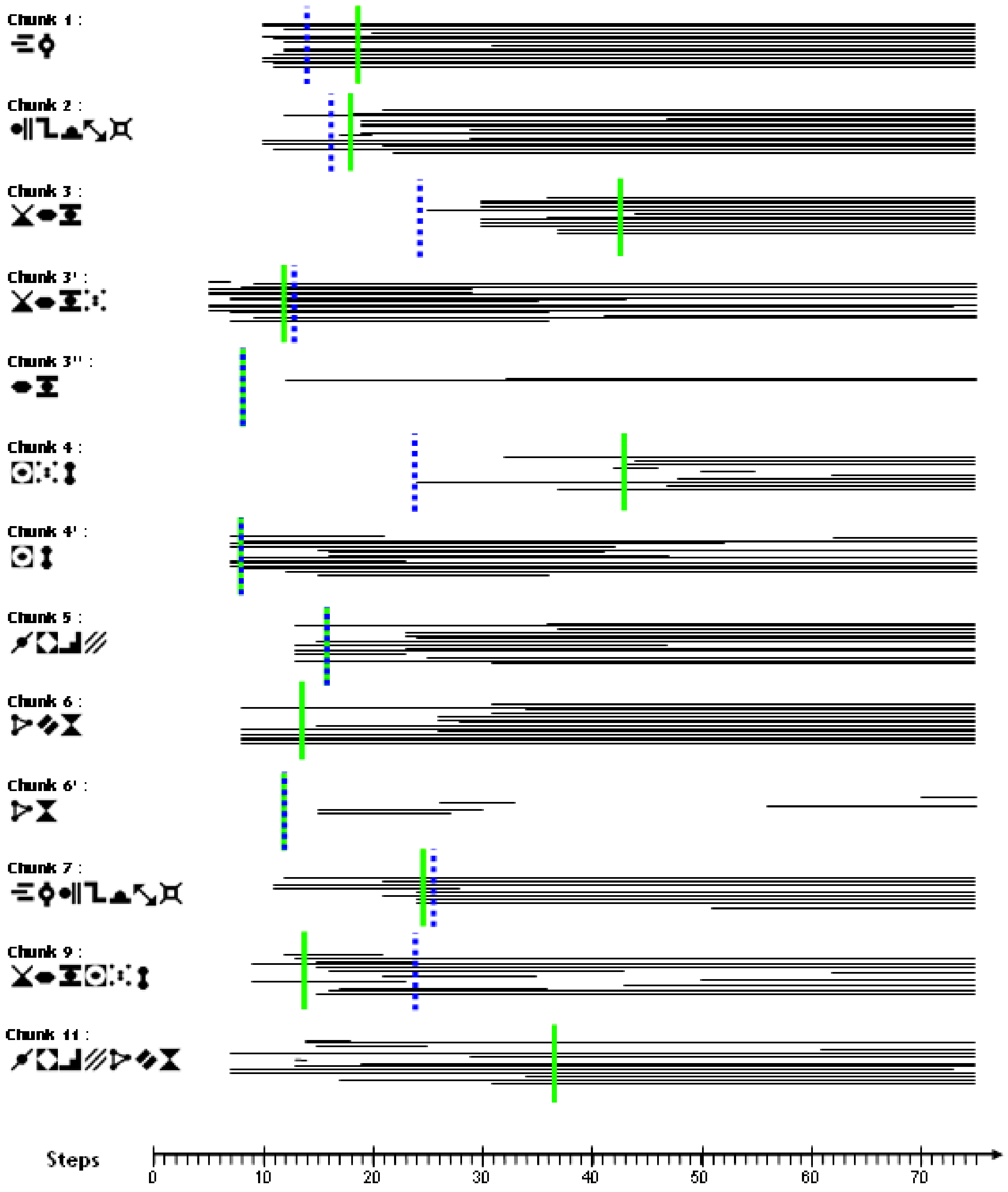


Fig. 7. List of the 13 chunks created by MDLChunker at the step indicated by a vertical line. All are significant excepted chunks 3'' and 6'. Each horizontal line corresponds to a participant: the left end indicates

the step at which the chunk was created, its length indicates its duration. Results for MDLChunker-cog presented section 7.1 are indicated by a vertical dotted line.

All the 11 significant chunks are successfully created by MDLChunker and are created approximately at the same iteration as the majority of the participants (in average two iterations later with a standard deviation of 7 iterations). MDLChunker only created 2 non-significant chunks, and each participant created in average 4.9 non-significant chunks. Thus, MDLChunker is a better predictor of typical participant behaviour than any randomly chosen participant. A quantitative evaluation of its predictability is given at the end of this section.

The 13 chunks created by MDLChunker are composed of the 11 significant chunks found by the majority of the participants and 2 non-significant chunks also found by 2 and 4 participants respectively. The time course of their creation over the 75 sets is presented on Fig. 7, together with the results of the participants.

The 11 significant chunks are not exactly the 11 chunks of the generative grammar. All the chunks of the second level (chunks 1 to 6) are significantly found by the participants as well as 3 of the 5 chunks of the third level (chunks 7, 9 and 11). The participants and MDLChunker also created 2 significant chunks not included in the original material, noted 3' and 4' in Fig. 7 because they are variations of the corresponding chunks 3 and 4. The ability of our system to create chunks 3' and 4', and not chunks 8 and 10, is of central importance because MDLChunker aims at predicting the human representations and not reproducing the original grammar. It is important to have a system able to make the same "errors"[13] than humans do.

Another representation of the relative distance between the model and the participants is obtained by plotting them as points in the chunk space. This space being of dimension 82, we performed a bi-dimensional projection of the chunk space in the highest inertia subspace (Fig. 8.), showing the position of MDLChunker (filled circle) and the participants (empty circles) at six different time steps equally distributed over the 75 steps.

[13] It is probable that these "errors" are highly dependant of the used dataset, and that the statistics provided by the experiment are not sufficient to discover the underlying grammar.

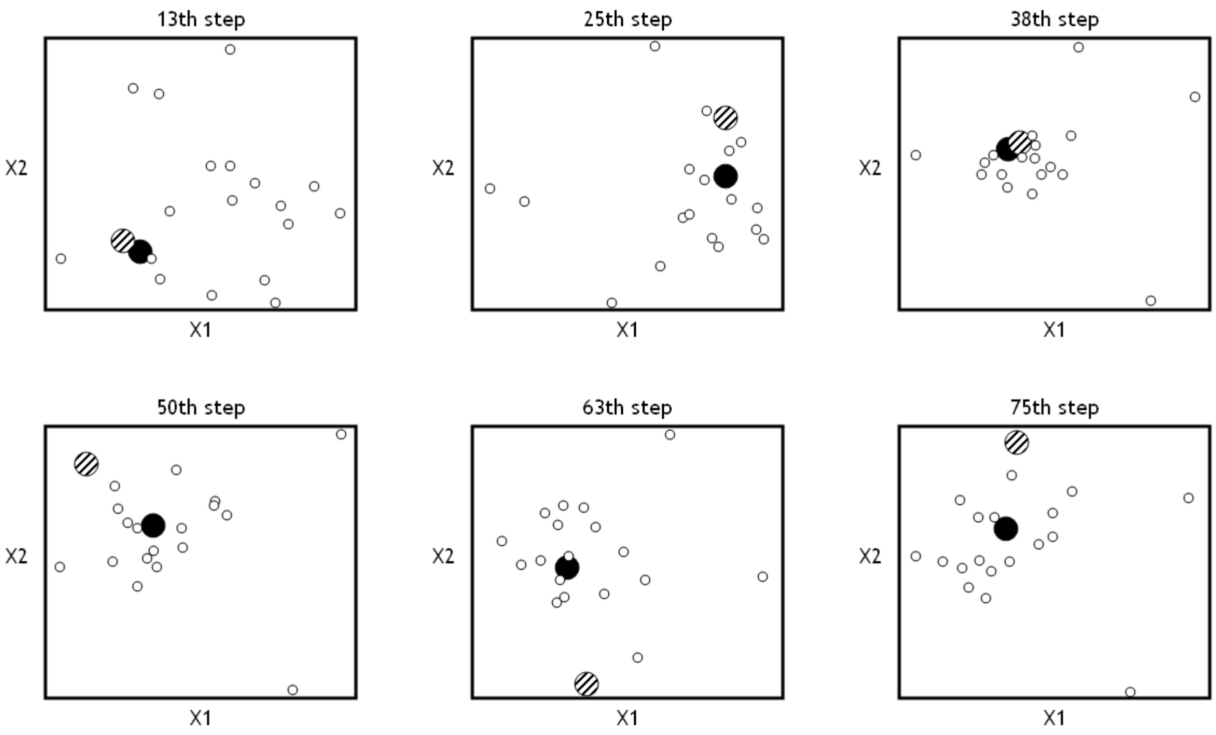


Fig. 8. Bi-dimensional scaling presenting the relative positions of MDLChunker (filled circle) and the 18 participants (empty circles) during the experiment. X1 and X2 are the highest inertia axis. Results are also given for MDLChunker-cog presented in section 7.1 (hatched circle).

The central position of the system may also be characterized in a more rigorous way (Fig. 9), by plotting the ranks over time. At a given step, the best representative of the participants' behavior, the participant having the most central position in the chunk space, is assigned the smallest rank.

By averaging the ranks over time, we obtain a general estimator of model predictability for the 75 steps of the experiment. We found that the system has the third smallest average rank among 19. Only two participants are better predictors. It is interesting to notice that although the predictive power of MDLChunker is not constant over steps, it converges to the most central position after 48 steps. This suggests that MDLChunker is a pretty good predictor of the time course of chunk creation and is also the best asymptotic predictor of the chunks created by the participants.

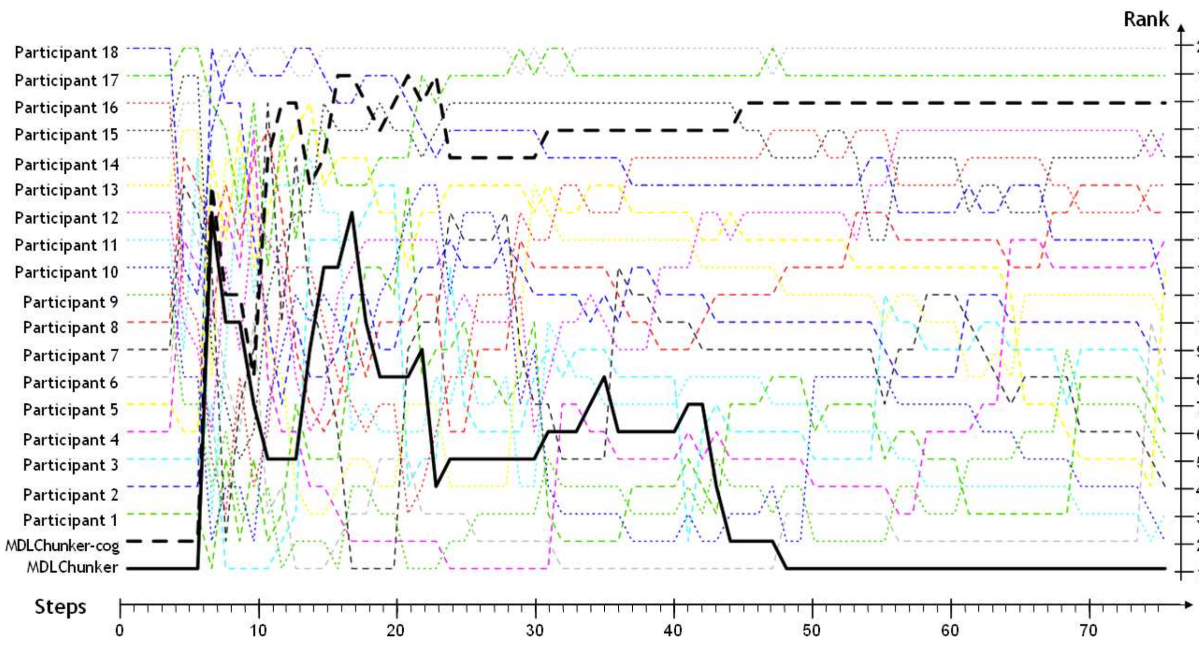


Fig. 9. Ranks of the participants over the 75 steps, the lowest rank corresponding to the most central position. MDLChunker (bold filled line) is considered as a virtual participant. Results for MDLChunker-cog presented in section 7.1 are indicated by a bold dotted line.

The best possible deterministic model should have the first rank all over the 75 iterations, but a model whose rank is close to the average rank[14] (the 9th rank in our case) is a good model. A model having a constant rank has a constant predictive power. Fluctuations are due to participants' behaviors. A perfectly constant rank is not possible, but a good model is expected to have a non-increasing rank. Between two models having the same average rank, the one having the most constant rank has the most constant predictive power over time, and thus is better than a model exhibiting huge variations. Having a good asymptotic predictive power (small rank for high iterations) is also important for the sake of comparison with other models. In particular, let us remark that models such as PARSER or Competitive Chunker are usually tested on their ability to reproduce asymptotically the behavior of participants.

[14] Such a model has the performance one may expect from an unknown participant, but it is not the best predictor of what an unknown participant would do. Models having a rank smaller than the last rank are better predictors than the last rank participant would predict.

Results show that among 18 participants, MDLChunker is the best asymptotic predictor, and is a good predictor at each time step.

5. Theoretical comparison with other computational models

A basic requirement for a cognitive model is its ability to incrementally process the stimuli, which is closely related to its capacity of using information already processed in order to guide the representation of new stimuli. This is the perception shaping effect (Perruchet et al., 2002). We present now different models in the light of this perception shaping effect, and compare them to MDLChunker.

The idea of applying the MDL principle to describe chunking mechanisms is not new. Brent & Cartwright (1996) used it in order to account for the ability of children to segment a stream of phonemes into words. As already discussed in section 2.2, word segmentation can be viewed as grouping into chunks phonemes which tend to occur together, and MDL is a way to select the most probable segmentation from a set of possible ones. In Brent & Cartwright's model, finding the shortest description is not incremental: it is done by testing all possible segmentations of the input stream considered as a whole. The authors themselves notice the cognitive implausibility of this exhaustive exploration of the chunk space and let open the possibility of using an incremental approach. In contrast, the optimization process of MDLChunker is iteratively performed during the learning process: at each stage new chunks are created whenever they help decreasing the overall description length. A second advantage of MDLChunker over Brent & Cartwright's model is its ability to reproduce the way existing chunks shape perception of new stimuli (see section 3.4), and in particular the vanishing sub-chunk effect described in section 7.

Some models are designed to account for this on-line perception shaping process in the field of speech segmentation. One of them is PARSER (Perruchet & Vinter, 1998). Without explicitly using the MDL principle, PARSER is based on three processes (chunk creation, forgetting and interference) leading to a general behavior similar to that of MDLChunker. At each time step, PARSER randomly selects between 1 and 3 units in the input to form a new candidate chunk or reactivate an existing chunk. Then a constant forgetting process is applied to existing chunks. As a consequence, the life span of low frequency chunks is very short, and only those having a high frequency remain after some iterations. These creation and forgetting processes are somehow equivalent to the optimization performed by MDLChunker to only create relevant chunks. An interference mechanism is used by PARSER to decrease the weight of small chunks nested in bigger ones. This is one of the side effects that we have observed through the factorization process performed by MDLChunker. When run on the same sequences of concatenated artificial words, PARSER and MDLChunker give similar results (see section 7). They are both able to extract the words of the language together with the time course of word extraction.

Since the percept shaper used by PARSER is partly random, its efficiency on extracting only relevant chunks depends on the forgetting process. The BootLex model (Batchelder, 2002) uses mechanisms very similar to those of PARSER, replacing the random size of the input by a deterministic parsing process. In this case, an estimation of the chunk performance is required, and the author uses a frequency-based heuristic to guide the search. In BootLex, the process of chunk creation is under the guidance of a Maximum Likelihood Estimator (MLE). Since this estimator imposes no constraint on the complexity of chunks, chunks tends to become longer as the algorithm processes the stimuli. To solve this problem, the author uses an external parameter (called OptLen) to fix the average chunk length. In MDLChunker, the estimation of chunk utility is also based on frequency, but the superiority of MDL over MLE is its ability to deal with model complexity (Vitanyi, 2005). MDLChunker does not suffer from this overfitting problem, thus avoiding the use of external parameters to constrain the chunk length.

MDLChunker can be qualitatively compared with other models of stream segmentation, using the classification proposed by Batchelder (2002) based on functional characteristics (Table 1). The characteristics involved are:

- Lexicon: specifies whether the model contains an explicit representation of the created chunks;

- Cluster or divide: distinguishes between the clustering strategy consisting of merging existing chunks (cluster) or inserting boundaries (divide);

- Cumulate: indicates the ability of the model to process the dataset incrementally;

- Feedback from output: indicates the ability of the model to use its own output to shape the perception of the current input;

- Constraints: indicates the free parameters.

Table 1. Table reproduced from Batchelder (2002) together with the corresponding characteristics for the MDLChunker and SP (Wolff, 2006).

Model	Build lexicon ?	Cluster or divide ?	Cumulate ?	Feedback from outputs	Constraints ?
BootLex	Lexicon	Cluster	Cumulate	Feedback	Optlen value
Networks[15]	No	Divide	Cumulate	No	Threshold value
MDL[16]	Lexicon	Cluster	No	No	Compute-intensive
MBDP[17]	Lexicon	Divide	Cumulate	Feedback	External parameters
MDLChunker	Lexicon	Cluster	Cumulate	Feedback	No
SP	Lexicon	Cluster	Cumulate	Feedback	External parameters

Other MDL-based chunking models such as Goldsmith (2001) and Argamon et al. (2004) are able to account for the segmentation of words into affixes. Such models are not described here because their architecture is very specialized for this specific task and cannot easily generalize to other domains.

Since it is very similar to MDLChunker, we now describe the Simplicity and Power (SP) model (Wolff, 2006). As it is shown in Table 1, this model is able to incrementally shape perception using existing chunks. A common point with MDLChunker is that SP can be applied to various fields: from word segmentation (Wolff, 2000) to probabilistic reasoning (Wolff, 1999) and medical diagnosis (Wolff, 2006). While not explicitly using the MDL principle, SP is also a simplicity-based chunking model. It uses a Shannon-Fano coding to search for the shortest

[15] The three connectionist networks reported are: Aslin et al. (1996), Christiansen et al. (1998) and Cairns et al. (1994).

[16] The two MDL-based models reported are: de Marcken (1995) and Brent & Cartwright (1996).

[17] The two Model Based Dynamic Programming models reported are: Brent (1999) and Venkataraman (2001).

encoding of the current stimulus. This "alignment process" of the new stimulus with old perceived chunks is the key mechanism in SP. It is similar to alignment of DNA sequences, and the heuristic involved is based on codelength minimization. The better the alignment, the higher the compression of the encoded stimulus with respect to the raw stimulus. In SP the way existing chunks may shape the perception of new stimuli is addressed through the alignment process; the simplicity principle is involved only for stimulus alignment but not for chunk creation. In MDLChunker, both the factorization and the optimization processes are under the guidance of the MDL principle.

6. Experimental comparison to competing models

The rank defined section 4.4 is a natural and objective indicator of model performance, but it cannot be used to compare the performances of different models on the datasets found in the literature. PARSER and Competitive Chunker explicit the chunks, but this information is not available from the participants due to the experimental setting. On the other hand, applying other models to our data is not straightforward because in most of them the order of symbols inside chunks is important[18].

In the following we compare the performance of MDLChunker to that of Competitive Chunker and PARSER, on the data of the well-known experiment by Miller (1958). Results of Competitive Chunker on these data have been published by Servan-Schreiber & Anderson (1990). We reimplemented PARSER based on Perruchet & Vinter (1998) and personal communication with the first author, and ran it on the same data.

Miller's data come from a typical artificial learning experiment in which participants are exposed to strings of letters either produced by a finite state automaton representing a formal grammar (L), or randomly generated (R). There are four sets of data, two of them grammatical (L1, L2) and two random (R1, R2). Each one contains 9 strings:

- L1: SSXG, NNXSG, SXSXG, SSXNSG, SXXXSG, NNSXNSG, SXSXNSG, SXXXSXG, SXXXXSG

[18] Most of them have been designed for studying chunking in the field of language acquisition where data are intrinsically ordered and considering unordered chunks is irrelevant.

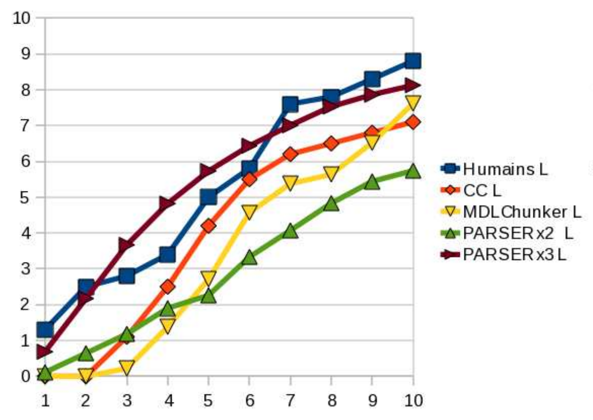
- L2: NNSG, NNSXG, SXXSG, NNXSXG, NNXXSG, NNXXSXG, NNXXXSG, SSXNSXG, SSXNXSG
- R1: GNSX, NSGXN, XGSSN, SXNNGN, XGSXXS, GSXXGNS, NSXXGSG, SGXGGNN, XXGNSGG
- R2: NXGS, GNXSG, SXNGG, GGSNXG, NSGNGX, NGSXXNS, NGXXGGN, SXGXGNS, XGSNGXG

A block of 9 strings is presented to participants, in a random order, one string at a time during 5 seconds each. Then participants are required to recall as many strings as they can. When they have finished, a new trial begins with the same 9 strings in another random order. The dependent variable studied is the mean number of strings correctly recalled after each of ten trials.

In order to assess which are the strings recalled by the models, we follow the criterion proposed by Servan-Schreiber & Anderson (1990): a string is considered recalled if the model creates the corresponding chunk. Competitive Chunker has been tested (Servan-Schreiber & Anderson, 1990) through 10 simulated users, and its two parameters, decay and competition, were adjusted in order to find the best fit to Millers' experimental results. In our implementation of PARSER we used the default parameters. MDLChunker has no free parameters. We made 100 runs both with PARSER and MDLChunker to have a good precision in the results in a reasonable computation time.

Both PARSER and MDLChunker did not learn much after being exposed to 10 trials: only about 1 or 2 strings were correctly recalled in L lists, although humans correctly recalled almost all strings. Actually, effects of stimuli exposure may vary according to the duration of presentation. Humans were presented each string during 5s, but the results would have been different with a shorter or a longer presentation. MDLChunker always processes stimuli in the same way: there is no mechanism to change the strength of a stimuli. With PARSER, we could have changed the weight which is given to a new stimuli (the default value is 1), but we preferred to use the default values rather than adjusting parameters.

In order to solve the problem, we doubled each stimuli, in order to mimic the fact that participants were exposed to stimuli during a long time compared to normal processing of letters. Since PARSER showed bad results, especially with the R list, we also ran it with tripled stimuli. Results are presented in Figure 10:



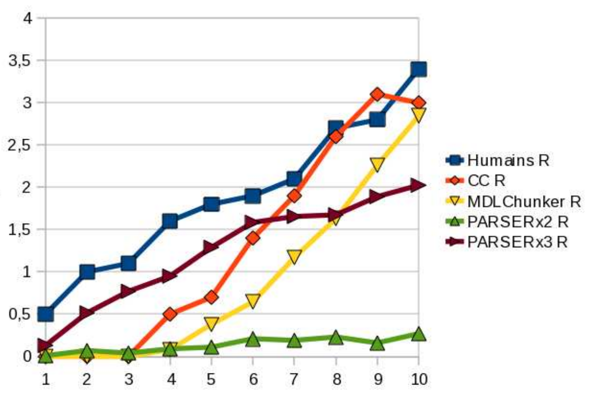


Figure 10: Mean number of string correctly recalled for humans, Competitive Chunker, MDLChunker and PARSER, for L and R lists.

Results for the L lists shows that MDLChunker compares well to other models. It is slower to learn but get similar results after 10 trials. These results obviously depend on the number of strings that are presented in a row. PARSER is much better when strings are tripled than when they are doubled (especially in lists R). The same apply for MDLChunker which even learns too fast when strings are tripled. As we said earlier, we could have adjusted PARSER parameters to have a better fit. We could also have introduced a new parameter in MDLChunker to better fit the data, but we believe that computational models should be tested on default parameters or, better, should contains no parameters. However, in the particular case of cognitive modeling, parameters are sometimes necessary to reproduce inter-participant variability or external constraints.

7. Model robustness and cognitive limitations

The version of the model discussed in the preceding sections does not take into account cognitive constraints such as memory limits. We have resigned cognitive plausibility to test the principle of "chunking by MDL" which proved to successfully account for the way humans tend to group items. In this section we investigate how MDLChunker can be extended to account for cognitive limits. To that end, we designed a version of MDLChunker that produces equally good results and is more plausible from a cognitive point of view.

7.1. MDLChunker-cog

There are two motivations for introducing MDLChunker-cog. The first is to test whether the results obtained in section 4 can be reproduced with an implementation[19] that takes into account the brain limited computational capacity (Oaksford & Chater, 1993; Van Rooij, 2008). The second is to test the robustness of the applied principles: results have to be consistent over different possible implementations of these principles. MDLChunker suffers from both requiring an infinite memory (to store representations), and using brute-force computations during the factorization phase. To overcome these two problems, MDLChunker-cog does not explicitly store representations, and computations are based on local information only, processed in parallel. Its architecture is a network composed of weighted oriented links and nodes hierarchically organized into layers (see Fig. 11). Describing precisely MDLChunker-cog is far beyond the scope of this paper. A precise description may be found in Robinet (2009) (in French).

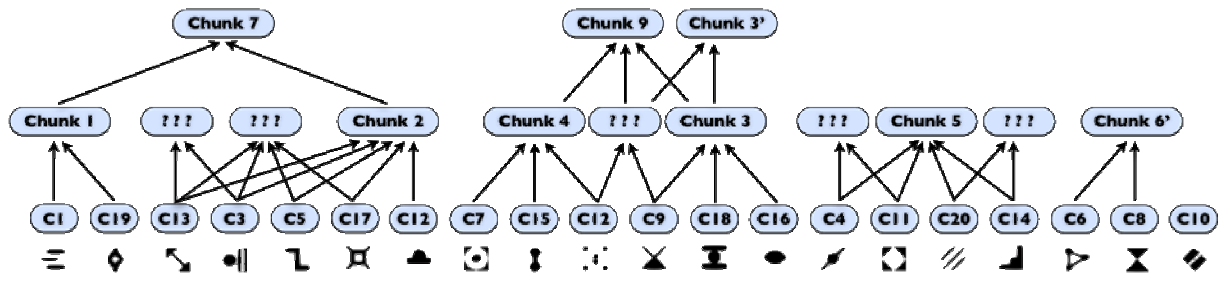


Fig. 11. State of MDLChunker-cog when trained on the dataset used in section 4. Nodes C1 to C20 are the canonical chunks whose corresponding symbols are represented below. Nodes of layers 2 and 3 are the chunks created during the learning process. To simplify the comparison with MDLChunker, the names are those of section 4.

MDLChunker and MDLChunker-cog are based on the same processes: the updating of codelengths, the factorization and the optimization. MDLChunker-cog starts with an empty fully connected N-layer feedforward network, N being

[19] The aim is to show that such an implementation exists, it is not to find the best one.

the highest hierarchy level of chunks that the model can reach. At the beginning, the k nodes of layer L are connected to nodes of layer $L+1$ through $k-1$ inactive links and one active link.

Stimuli are incrementally processed following the three steps procedure described in section 3.1. First, the canonical chunks involved in the stimulus are activated (bottom layer). Then activation propagates through the network from the bottom to the top. A node having all its children activated becomes active (i.e. represents the chunk grouping its children) and inhibits its children. The greater the difference between the children and the parent codelengths the faster the activation process. As a consequence, when chunks overlap, the one saving the largest codelength is activated first. Since its children are inhibited, other competing nodes (chunks) are not activated: this is the factorization process (see Fig. 12).

Codelengths are updated according to node activation using equation (3) where n_i corresponds to the number of activation of node i.

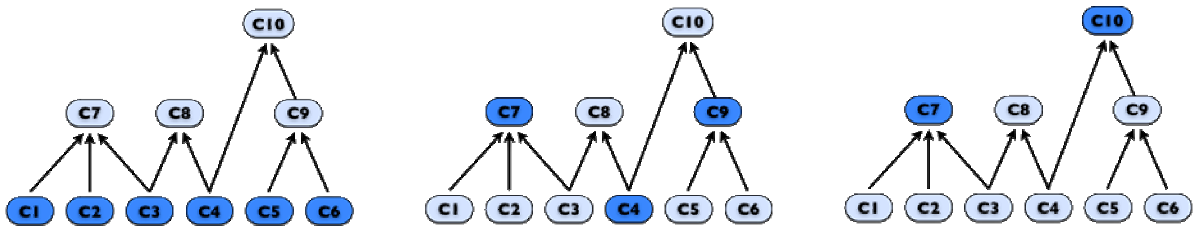


Fig. 12. Example of factorization for the stimulus 1 2 3 4 5 6. Canonical chunks C1 C2 C3 C4 C5 C6 are first activated. Then activation propagates to chunks C7 C4 C9 because codelength saving is more important using C7 than C8. The stimulus is finally factorized into C7 C10.

Links having both father and son activated are strengthened. This process, similar to a Hebbian mechanism (Hebb, 1949), allows the creation of new chunks (optimization process). The link strength is used to account for the number of co-occurrences between two nodes. The higher this value, the stronger the confidence that the child node has to be part of the parent chunk (see Fig. 13). A parent and a child may by merged by activating the link between them. Two nodes are merged if the codelength of the resulting chunk is smaller than the codelength of the two nodes considered separately. The criterion used to activate a link is the same as the one used by MDLChunker to create a chunk (equation (4)). Activating a link increases the chunk arity: unary chunks (simple nodes) become binary, ternary, etc.

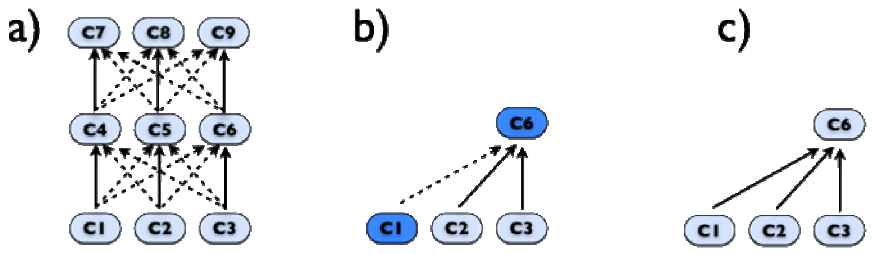


Fig. 13. Figure a) presents a 3-layer network with three canonical chunks at the beginning of the learning. Unary chunks and inactive links (dotted) are represented. Figures b) and c) presents how joined activations of C1 and C6 strengthens the link C1-C6, thus leading to its activation. The corresponding binary chunk C6 becomes ternary.

MDLChunker-cog uses links to extract the co-occurrences between nodes. It is an implicit way of storing the relevant information provided by the Stimuli|Chunks part of MDLChunker without requiring an infinite memory. Despite the sub-optimality of the optimization and factorization processes with respect to those described section 3, results obtained by MDLChunker-cog are quite similar to those of MDLChunker. Results of the first experiment are presented Fig. 7, 8 and 9. We now provide a deeper comparison of the two models.

7.2. Experimental validation

In order to compare our two models, we replicated our experiment with several changes. First of all, stimuli are created with a new grammar randomly generated (presented in Fig. 14). It exhibits several differences with the previous one of section 4.2 (Fig.5). The 20 symbols are separated into two non-overlapping clusters, it has one more level and the chunk arity is lightly higher (μ=4.0, σ=1.2 compared to μ=3.3, σ=1.0 in the first experiment). Moreover, it has a couple of identical chunks (chunk 9 and chunk 10) which may be seen as a way of artificially increasing the corresponding chunk probabilities when generating the experimental data. Secondly, the generated stimuli do not contain any noise: no extra symbol is added and no symbol is deleted. These changes are introduced to test the robustness of the results. The rest of the experiment follows exactly the same protocol as described in section 4.

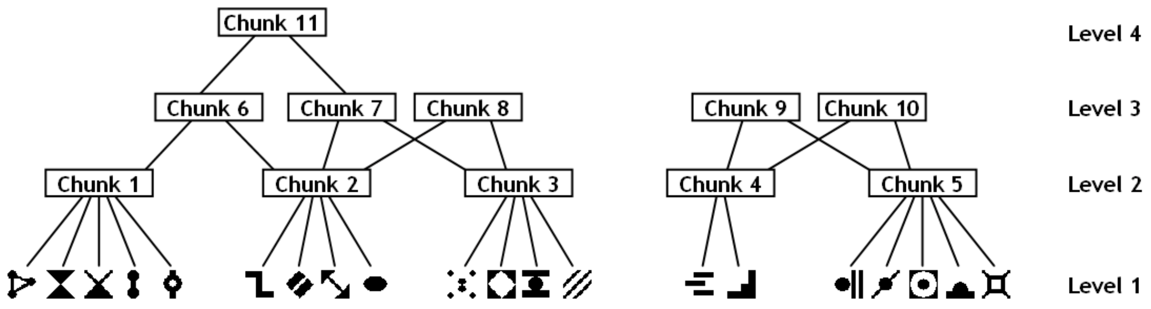


Fig. 14. The grammar used for the second experiment.

The performance of 18 recruited participants on the sequence of 75 sets of symbols is compared to the results of our two models on the same set of stimuli.

MDLChunker created 6 chunks: C1, C2, C4, C5, C6, C9 (or C10 which is the same as C9), that is all second-level chunks except C3, plus 2 third-level chunks. The fourth-level chunk C11 was not created. MDLChunker-cog created the same 6 chunks together with 4 incomplete chunks that are part of the second-level chunks C1, C2, C3 and C5 in which one or two symbols are missing.

A total of 52 different chunks were created by all participants, but their number of occurrences shows a clear distinction between 46 non-significant chunks (39 created by only one participant, 6 created by two participants and 1 created by 4 participants) and 6 significant chunks created by most participants. These 6 chunks are exactly those created by the two models. Their number of occurrences are presented in Table 2.

Table 2. Number of occurrences of the significant chunks

Chunks	C1	C2	C4	C5	C6	C9 / C10
Number of occurrences	17	18	18	17	10	18

Fig. 15 shows the time course of chunk creation for those 6 chunks. Vertical lines represent the moments when chunks are created by MDLChunker (solid green line) and MDLChunker-cog (dashed blue line). Models created chunks at roughly the same time as the majority of participants. Chunk 6 was generated quite late by MDLChunker, but only half of the participants had created the chunk at that time.

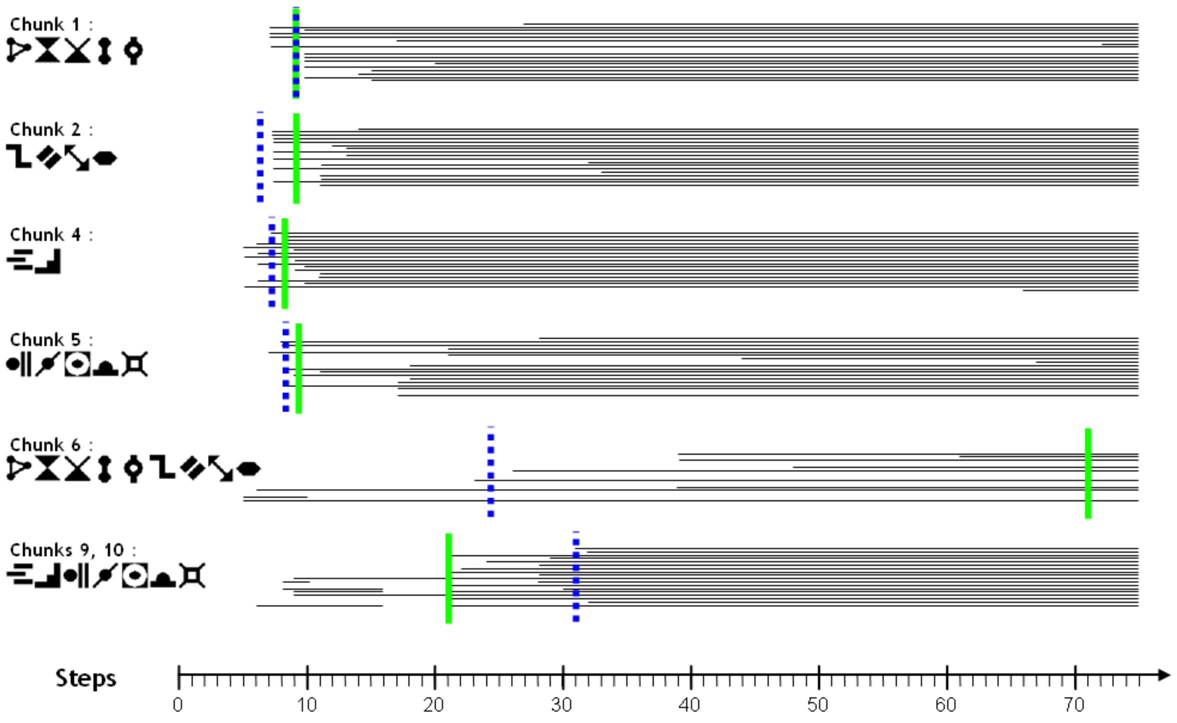


Fig. 15. Time course of chunk creation for the 6 significant chunks. Results are presented for the 18 participants, MDLChunker (solid green line) and MDLChunker-cog (dashed blue line).

It is worth noting that chunk C3 was neither created by any participant nor by the models, although the symbols of that chunk appear in 39 of the 75 stimuli. This should be compared with chunk C1 which was created although its symbols only appear in 29 stimuli. The symbols of chunk C3 nonetheless appear 5 times in a row in steps 22 to 26, but right after MDLChunker and most participants created their first second-level chunk: C9. It might be that participants were busy dealing with this new higher-level chunk and did not realize that some new symbols were worth to be grouped. Concerning the models, the explanation is the following: the creation of a new chunk increases the frequency of all the other chunks. For instance, A has a frequency of 20% in ABCBC but 33% after the chunk BC has been created. Therefore, their codelengths become shorter and new chunks are less likely to be created. This corresponds to the part P4 of equation (4).

A model which would only be based on stimulus frequency would have created chunk C3 before chunk C1, but this is not what happened. Both models are able to mimic the fact that creating a chunk is harder when other chunks have already been created.

Like previously, Fig. 16 shows the bi-dimensional position of participants at six different time steps. Both models appear quite central in the cloud of participants. Similarly to the previous experiment, results are quite consistent over time.

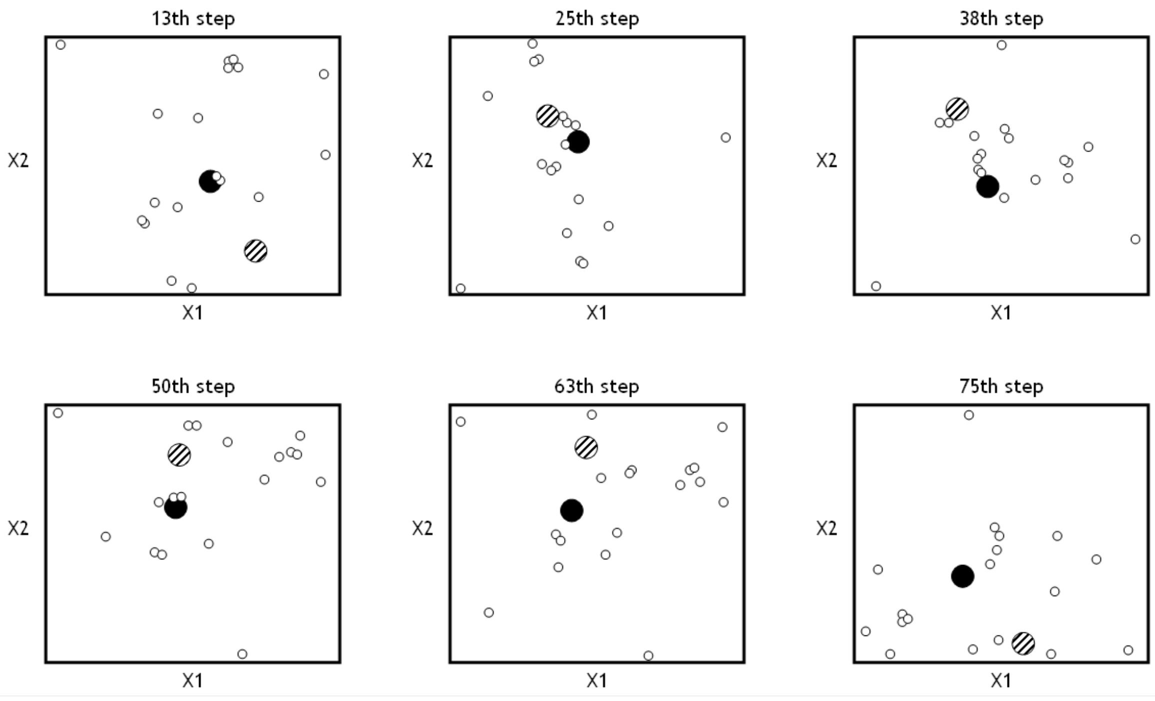


Fig. 16. Bi-dimensional representation of the relative positions of MDLChunker (filled circle), MDLChunker-cog (hatched circle) and the 18 participants (empty circles) at different time steps of the experiment. X1 and X2 are the highest inertia axis.

Like in the previous experiment, Fig. 17 displays ranks over time. MDLChunker has the best average rank at iteration 75 among 20 participants (including the two models). It is therefore the best prototype of the participant behavior. MDLChunker-cog ends up at rank 11. The difference between the rank curves of figures 9 and 17 is due to the absence of noise in the second experimental setting. Without that noise, both the models and the participants create less idiosyncratic chunks. The learning rate is higher with two third of the chunks created before iteration 10.

MDLChunker is a good asymptotical predictor. As a consequence, the convergence to the first rank is faster in this experiment where the learning rate is higher. Because MDLChunker-cog is sub-optimal compared to MDLChunker, the results provided by MDLChunker are necessarily better[20] than those of MDLChunker-cog.

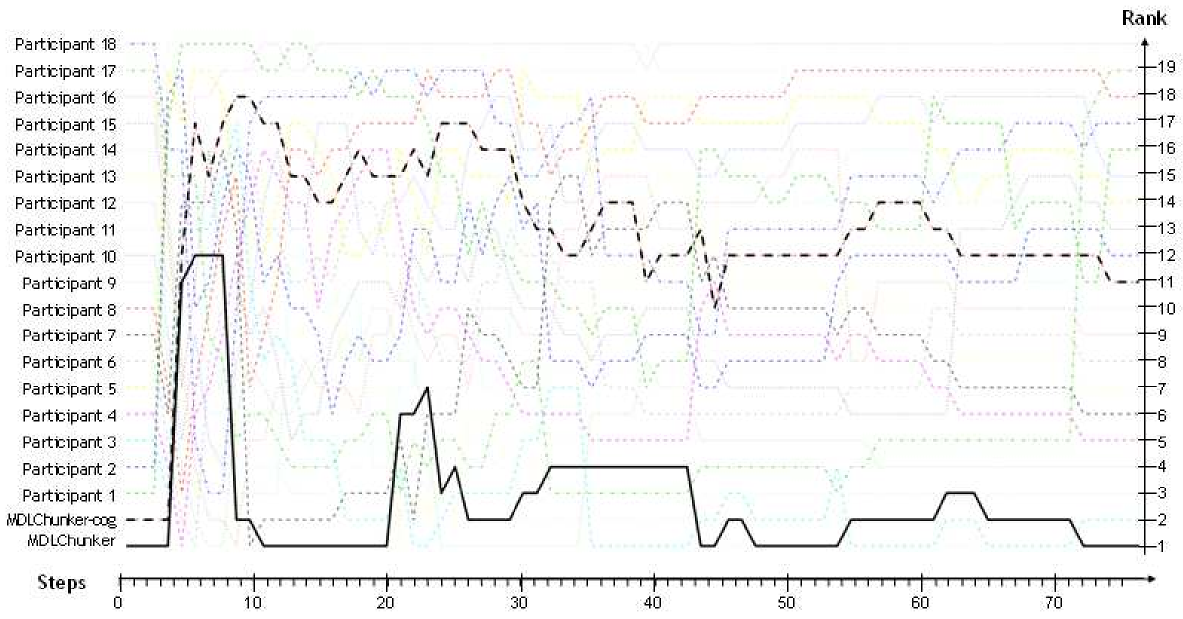


Fig. 17. Ranks of the participants over the 75 steps. MDLChunker (bold filled line) and MDLChunker-cog (bold dashed line) are considered as virtual participants.

Results are similar to those obtained in the previous experiment. A clear distinction exists between significant and non-significant chunks, and both MDLChunker and MDLChunker-cog are able to reproduce all the significant chunks which are not all the chunks of the initial grammar. The two models are thus efficient predictors of human behaviour and not simple grammar extractors. Changing some parameters of the grammar does not decrease the quality of the results, suggesting that the underlying principles of both models are quite robust. Testing the exact

[20] Each creation of a new chunk implies a loss of information about co-occurrences. For example in Fig. 12, the inactive link C1-C6 stores information about co-occurrences between C1 and C6. By creating the chunk C6: C1 C2 C3, it is impossible to recover the number of co-occurrences between other chunks Cx and C6: C1 C2 C3 from the co-occurrences between Cx and C6: C2 C3.

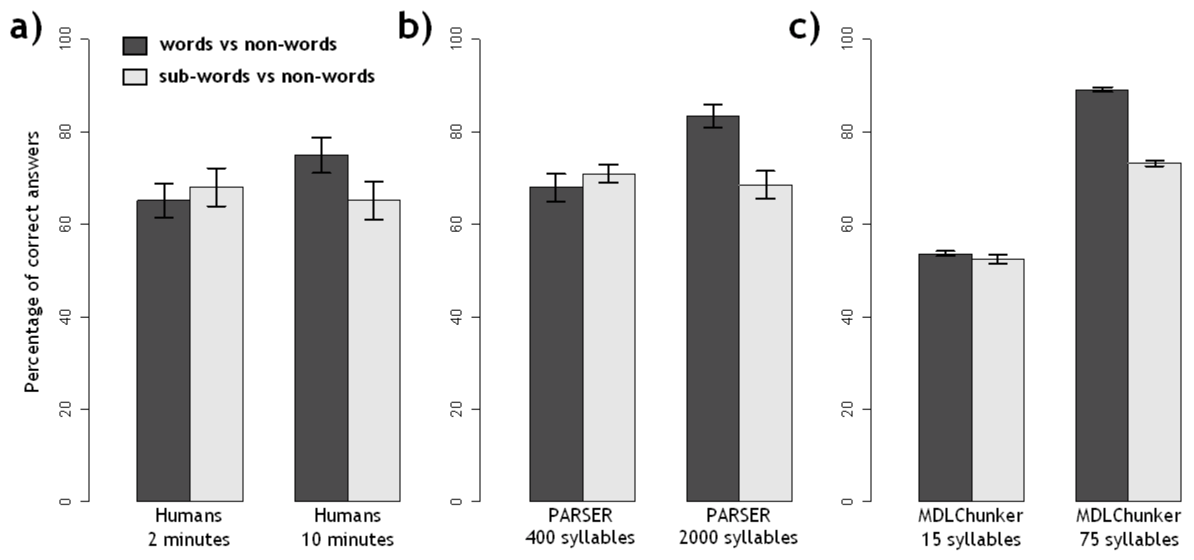
range of validity of the models, which requires to replicate the experiments varying different parameters of the stimuli generator (chunk arity, noise level, etc.) is not the purpose of this paper.

7.3. Application to word segmentation

In order to test the robustness of the model, we applied it to the domain of word segmentation. In a seminal paper, Saffran et al. (1996) showed that, when exposed to a stream of phonemes corresponding to a concatenation of artificial words, humans are able to segment correctly and learn the words from the sole transitional probabilities between syllables. According to Swingley's (2005) terminology, MDLChunker follows a clustering strategy, creating new units by grouping frequent ones, as opposed to a bracketing strategy which would attempt to insert boundaries in the flow of phonemes.

In contrast with the experiment described in section 4, the input is now a stream of units without any delimiter. We adapted the factorization process in order to split the current input into units. There are various ways of splitting the beginning of the current input according to the chunks already created. In MDLChunker, the best segmentation is the one which has the smallest codelength. The first two units are then candidates for forming a new chunk. As usually, the new chunk is created only if its creation decreases the description length.

We tested MDLChunker on data from Giroux & Rey (2009), who designed a new experiment to compare recognition performances of adults hearing either 2 or 10 min of an artificial spoken language. The language is composed of 6 words, formally represented here as ABC, DEF, GH, IJ, KL and MN. Participants just listen to a random concatenation of these words, uttered at the rate of 3.3 syllables per second by a speech synthesizer without any prosodic information. After 10 min training, recognition performances on dissyllabic words (such as "GH") are significantly higher than those obtained on dissyllabic sub-words (such as "BC"), while no difference occurs after 2 min training (Fig. 18 a)). This result credits the clustering strategy by suggesting that sub-word recognition is a necessary step in the word learning process.



Humans	2min	10min
p-value F-test	= 0.56	= 0.01

PARSER	400sy	2000sy
p-value F-test	*ns.*	< 0.01

MDLChunker	15sy	75sy
p-value F-test	= 0.41	$< 10^{-16}$

Fig. 18. Results from Giroux & Rey (2009) presenting the percentage of correct answers obtained for 32 human participants (a) and for 32 PARSER virtual participants (b). We added the corresponding results for 1000 MDLChunker virtual participants (c).

MDLChunker was run on the same artificial language to check whether it reproduces this vanishing sub-word effect (Robinet & Lemaire, 2009). 1000 simulations were performed with random input sequences. At the beginning, there is no difference, and both the dissyllabic words and sub-words are created, but after about 30 syllables the model behavior begins to change: non-words cost more and more to be represented, whereas actual words, both dissyllabic and trissyllabic, are efficiently coded.

We used the test designed by Giroux & Rey to compare MDLChunker to PARSER (Perruchet & Vinter, 1998) and to human production. The words GH, IJ, KL, MN are tested against the non-words CK, FM, CG, FI and the sub-words AB, DE, BC, EF are tested against the non-words HI, JK, LM, NG. In this test, the task is to choose the dissyllabic unit that best matches with the training language (in our case, it is the unit with the smallest codelength). The two tests (words *vs* non-words and sub-words *vs* non-words) are performed eight times for each participant (real or virtual). Averaged performances are presented on Fig. 18 c) together with the results obtained by PARSER b) (Perruchet & Vinter, 1998).

No significant difference between performance on words over sub-words was obtained with MDLChunker at 15 syllables (F(1, 999)=0.69 ; p=0.41), whereas a significant difference (F(1, 999)=465 ; $p<10^{-16}$) is observed at 75 syllables. MDLChunker was therefore able to reproduce the main effect. However, it learns much too fast since only 75 syllables are necessary to distinguish between disyllabic words and sub-words.

The main reason of this too fast learning rate is that MDLChunker has an infinite memory. We designed a new version of the model that uses a memory buffer which restricts the amount of previous data available for chunk construction. This buffer plays the role of a short-term memory (STM) whereas the set of existing chunks is analogous to a long-term memory. Chunks are therefore created if their creation decreases the size of the data in the buffer plus the existing chunks. Old data units exceeding memory size are removed from STM (see Fig. 19). Instead of modeling STM as limiting the number of items, our framework naturally considers it as restricting the quantity of information. We will discuss in the conclusion of this paper this novel way of considering STM.

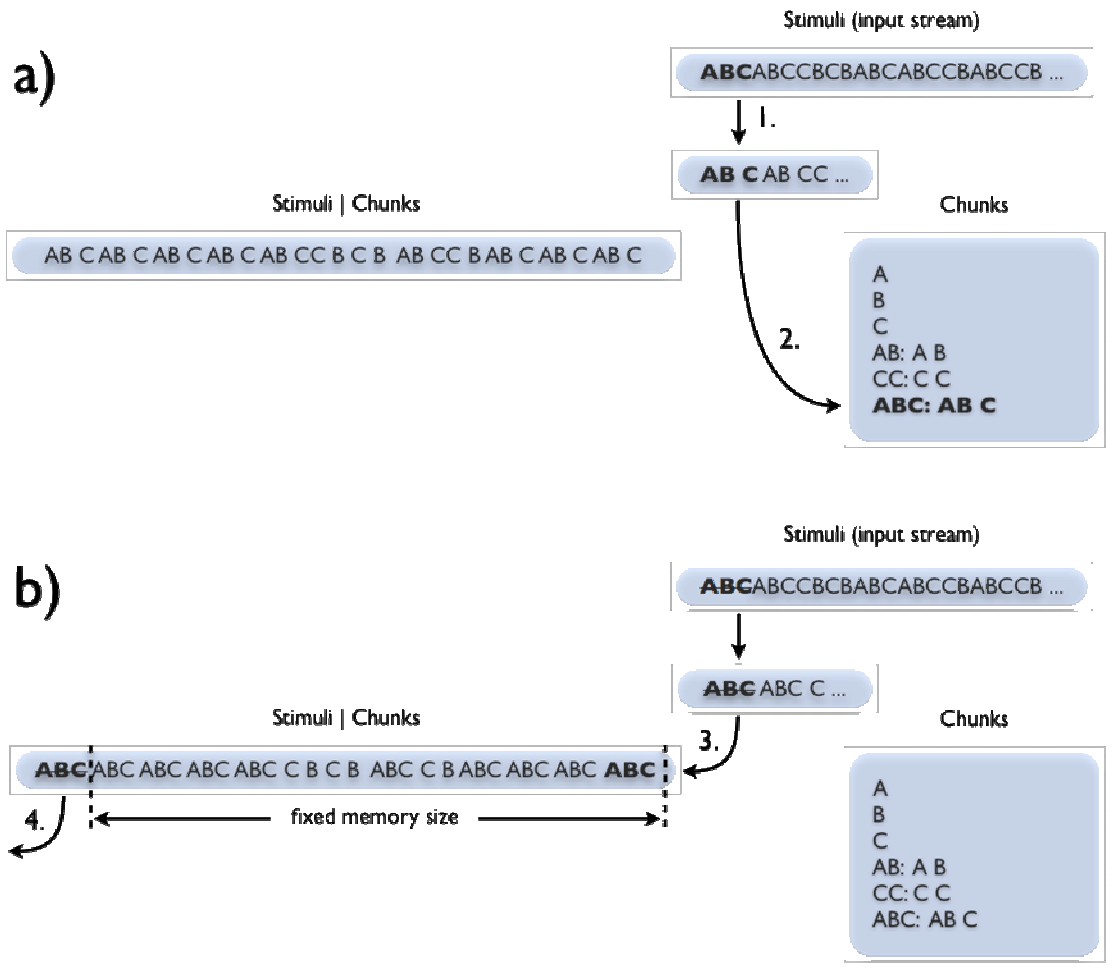


Fig. 19. Modifications introduced in order to apply MDLChunker to a stream segmentation task. Fig. a) presents the factorization (step 1) and the optimization (step 2). Because the input stream is ordered, only adjacent chunks are candidates to form a new chunk (here ABC). Fig. b) presents the corresponding updating of memory (step 3) and the deletion of items exceeding the memory size (step 4).

By supplementing MDLChunker with a finite memory, we added a parameter (the memory size) that needs to be adjusted. With a huge memory (1000 bits), the model learns at a very high rate, like in its previous version. With a too small memory (100 bits), no learning occurs at all, because there is not enough data available at the same time in memory to find regularities. With a buffer size of 150 bits, the model reproduces both the vanishing sub-words effect and the time course of learning observed experimentally in humans. After 2 minutes (400 syllables) there is no significant difference ($F(1, 999)=-0.21$; $p=0.64$) between words and sub-words. This difference becomes significant ($F(1, 999)=48.4$; $p<10^{-12}$) after a 10 minute (2000 syllables) training.

8. Conclusion

In this paper we presented MDLChunker, a model of inductive learning integrating two research areas in cognitive science: chunking and simplicity. Chunking mechanisms have been broadly described in the cognitive psychology literature. Simplicity provides a plausible explanation of the way humans extract regularities from highly patterned stimuli (Chater, 1999). Chunking models and simplicity-based models involving refinements to account for well-known cognitive limits (interferences, forgetting, etc.) have been proposed in various domains. Such models are of particular interest to analyze the mechanisms involved in the resolution of particular ecological tasks. Generally they correctly reproduce humans' productions but their results are consequences of interacting mechanisms difficult to disentangle. Our approach is different: we are not interested in modeling a specific task but rather to study a general cognitive principle, the association of simplicity and chunking.

According to algorithmic information theory, the best model of empirical data, i.e. the one with higher predictive power, has the shortest codelength. Models with longer codes necessarily include spurious information, not supported by the data, introducing biases in their predictions. This principle is at work in most scientific models. In this context it is called the Occam's razor principle, and states that among models explaining a given phenomenon, the best one is that with less parameters. In this paper we extend this idea to humans. In our view humans implement the same principle to induce knowledge from the environment, encoding the available data into chunks that minimize the description length of the stored information. Not only "short" models are more likely to make correct predictions, they also require fewer resources, which may be considered an ancillary benefit.

Paraphrasing simplicity as Minimum Description Length, we propose a computational cognitive model of the way humans perform induction: MDLChunker. Our model, not tailored for any specific task, integrates chunking and simplicity using a Minimum Description Length two-part coding criterion to guide the chunking process. The description language of both chunks and representations is homogeneous in order to fulfil the theoretical requirements of the MDL framework. The strength of MDLChunker over existing models is that it does not have any adjustable parameter, so that a posteriori fits to experimental results cannot be realized. This is of central importance for validation of the simplicity-chunking association.

In this paper we showed that MDLChunker behaves like humans in several experimental tasks that require building and selecting models of the environment. We believe that models of inductive learning should not be just validated through their ability to reproduce percentages of correct answers, as is often the case in the framework of artificial grammar learning. Creating chunks at the same time as humans is much more difficult to reproduce, and requires the design of specific tricky experiments in order to track the time course of chunk creation by human participants. Our model predicts the time of chunks creation, a further evidence of its appropriateness. Moreover, the performance of our parameter-free model on experimental data from the literature is comparable to that of models with adjustable parameters.

After introduction of memory limitation, a cognitive constraint not included in our first design, we tested the quality of our model with a second experiment. Not only memory limitation does not spoil its ability to reproduce human behaviors but the way we view the creation of chunks has repercussions on short-term memory modeling. Usually short-term memory is viewed as a buffer with a capacity limited to a fixed number of items. However, all the chunks do not convey the same amount of information, and should occupy different space in memory. For instance, a chunk of letters like IBM may need fewer resources to be maintained in short-term memory than GWIC (for Global Warming International Center), especially because the latter is less frequent. This is precisely the idea on which codelengths are based: short or frequent items have shorter representations, ceteris paribus. This is why we propose to limit the short-term memory capacity to a fixed number of bits (a quantity of information) instead of items. With this idea we reconsidered experiments of the literature on word segmentation, where memory has to be bounded in order to reproduce the participants' learning dynamic. Tuning the short-term memory capacity, i.e. the number of bits it may store, allows correct reproduction of the published results.

MDLChunker predicts which chunks are created, the relative time of their creation and their codelengths. The latter represent the usefulness of chunks because they are smaller the more frequent their use. In the future it would be interesting to capture the degree to which humans create and store chunks. A specific experiment would probably be necessary to measure, for example, reaction times to make decisions involving the use of particular chunks.

Our model can be viewed as a learning process based on a notion of cost, the description length. However, only the cost of representations has been considered in the present work, not the cost of their creation or identification. For instance, given the chunks ABC and DA, the model might prefer to view DABCDA as DA+B+C+DA rather than D+ABC+DA, based on the respective codelengths. The search over the possible encoding of a stimulus into chunks bears a cost. This cost was neglected in our model although it may be quite important. This cost probably plays an important role in the dynamics underlying new concepts' induction. Including it in the model should lead to sub-optimal decisions from the codelength point of view, much in the manner of MDLChunker-cog.

9. Bibliography

Argamon, S., Akiva, N., Amir, A., Kapah, O. (2004). Efficient unsupervised recursive word segmentation using minimum description length. In *Proceedings of the 20th International Conference on Computational Linguistics (Coling04),* Morristown: ACL.

Aslin, R. N., Woodward, J. Z., LaMendola, N. P., & Bever, T. G. (1996). Models of word segmentation in fluent maternal speech to infants. In J. L. Morgan & K. Demuth (Eds.), *Signal to syntax,* 117-134.

Batchelder, E.O. (2002). Bootstrapping the lexicon: a computational model of infant speech segmentation, *Cognition,* 83(2), 167-206.

Bishop, C.M. (2006). *Pattern recognition and machine learning*. Berlin: Springer.

Brent, M. R. (1999). An efficient, probabilistically sound algorithm for segmentation and word discovery. *Machine Learning*, 34, 71-105.

Brent, M.R., Cartwright, T.A. (1996), Distributional aregularity and phonotactic constraints are useful for segmentation, *Cognition,* 61(1-2), 93-125.

Cairns, P., Shillcock, R., Chater, N., & Levy, J. (1997). Bootstrapping word boundaries: a bottom-up corpus-based approach to speech segmentation. *Cognitive Psychology*, 33, 111-153.

Chaitin, G.J. (1966). On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13(4), 547-569.

Chater, N. (1996). Reconciling simplicity and likelihood principles in perceptual organization, *Psychological Review,* 103, 566-581.

Chater, N. (1999). The Search for Simplicity: A Fundamental Cognitive Principle? *The Quarterly Journal of Experimental Psychology, A* 52, 273-302.

Chater, N., Vitanyi, P.M. (2003). Simplicity: a unifying principle in cognitive science? *Trends in Cognitive Sciences*, 7(1), 19-22.

Christiansen, M. H., Allen, J., & Seidenberg, M. S. (1998). Learning to segment speech using multiple cues: a connectionist model. *Language and Cognitive Processes*, 13 (2/3), 221-268.

Cleeremans, A. (1997). Principles for implicit learning. In D. Berry (Ed.) *How Implicit is Implicit Learning?*, 195-234. Oxford: OUP.

de Marcken, C. (1995). The unsupervised acquisition of a lexicon from continuous speech. *A.I. Memo No. 1558*, MIT Artificial Intelligence Lab.

Feigenbaum, E.A., Simon, H.A. (1984). EPAM-like models of recognition and learning. *Cognitive Science*, 8(4), 305-336.

Fiser, J., Aslin, R.N. (2001). Unsupervised statistical learning of higher-order spatial structures from visual scenes, *Psychological Science*, 12(6), 499-504.

French, R.M., Mareschal, D., Mermillod, M., Quinn, P.C. (2004). The role of bottom-up processing in perceptual categorization by 3-to 4-month-old infants: Simulations and data. *Journal of Experimental Psychology: General*, 133(3), 382-397.

Giroux, I., Rey, A. (2009). Lexical and sublexical units in speech perception. *Cognitive Science*, 33(2), 260-272.

Gobet, F. (1993). A computer model of chess memory. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, 463-468, Lawrence Erlbaum Associates.

Gobet, F., Lane, P., Croker, S., Cheng, P., Jones, G., Oliver, I., Pine, J.M. (2001). Chunking mechanisms in human learning. *Trends in Cognitive Sciences*, 5(6), 236-243.

Goldsmith, J. (2001). Unsupervised learning of the morphology of a natural language, *Computational Linguistics*, 27(2), 153-198.

Grünwald, P., Myung, I.J., Pitt, M.A. (2005). Advances in Minimum Description Length: Theory and Applications, MIT Press.

Hebb, D.O. (1949). *The organization of behavior,*. New York: Wiley.

Kirkpatrick, S, Gelatt, C.D., Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598), 671–680.

Kolmogorov, A.N. (1968). Three approaches to the quantitative definition of information, *International Journal of Computer Mathematics*, 2(1), 157-168.

Leung-Yan-Cheong, S., Cover, T. (1978). Some equivalences between Shannon entropy and Kolmogorov complexity. *IEEE Transactions on Information Theory*, 24(3), 331-338.

Li, M., Vitanyi, P.M. (1997). An introduction to Kolmogorov complexity and its applications. Berlin: Springer.

Miller, G.A. (1956). The magical number seven, plus or minus two: Some limits on our capacity to process information, *Psychological Review,* 63(2), 81-97.

Miller, G.A. (1958). Free recall of redundant strings of letters, *Journal of Experimental Psychology,* 56, 485-491.

Oaksford, M., Chater, N. (1993). Reasoning theories and bounded rationality. In K.I. Manktelow and D.E. Over (Eds), *Rationality: Psychological and philosophical perspectives*, 31-60. International Library of Psychology.

Pearl, J. (1984). Heuristics: intelligent search strategies for computer problem solving, Boston: Addison-Wesley.

Perruchet, P., Vinter, A. (1998). PARSER: a model for word segmentation. *Journal of Memory and Language*, 39(2), 246-263.

Perruchet, P., Vinter, A., Pacteau, C., Gallego, J. (2002). The formation of structurally relevant units in artificial grammar learning, *The Quarterly Journal of Experimental Psychology Section A,* 55(2), 485-503.

Redington, M., Chater, N. (1996). Transfer in artificial grammar learning: a reevaluation, *Journal of Experimental Psychology: General*, 125, 123-138.

Rissanen, J. (1978). Modeling by shortest data description, *Automatica,* 14(5), 465-471.

Robinet, V. (2009). *Modélisation cognitive computationnelle de l'apprentissage inductif de chunks basée sur la théorie algorithmique de l'information,* Ph.D Thesis, Institut Polytechnique de Grenoble, Grenoble, France.

Robinet, V., & Lemaire, B. (2009). MDLChunker: a MDL-based model of word segmentation. In N. Taatgen & H. Van Rijn (Eds.), *Proceedings of the 31th Annual Conference of the Cognitive Science Society (CogSci 2009)*. Amsterdam, Netherland: Cognitive Science Society.

Saffran, J.R., Aslin, R.N., Newport, E.L. (1996). Statistical learning by 8-month-old infants. *Science,* 274(5294), 1926-1928.

Servan-Schreiber, E., Anderson, J.R. (1990). Learning Artificial Grammars With Competitive Chunking, *Journal of Experimental Psychology: Learning, Memory, and Cognition,* 16(4), 592-608.

Shannon, C. (1948). A mathematical theory of communication. *Bell System Tech. Journal*, 27(3), 379-423.

Solomonoff, R.J. (1960). A preliminary report on a general theory of inductive inference, Report V-131, Zator Co, Cambridge Mass.

Solomonoff, R.J. (1964). A formal theory of inductive inference. Parts I and II. *Information and Control*, 7(2), 224-254.

Sun, R. (2004). Desiderata for cognitive architectures, *Philosophical Psychology,* 17(3), 341-373.

Swingley, D. (2005). Statistical clustering and the contents of the infant vocabulary, *Cognitive Psychology,* 50(1), 86-132.

Turing, A.M. (1936). On computable numbers: With an application to the entscheidungsproblem. *Proceeding of the London Mathematical Society,* 2, 230-265.

van Rooij, I. (2008). The tractable cognition thesis. *Cognitive Science,* 32(6), 939-984.

Vapnik, V.N. (1998). The Nature of Statistical Learning Theory. Berlin: Springer.

Venkataraman, A. (2001). A statistical model for word discovery in transcribed speech. *Computational Linguistics*, 27(3), 352-372.

Vereshchagin, N.K., Vitanyi, P.M. (2004). Kolmogorov's structure functions and model selection, *IEEE Transactions on Information Theory,* 50(12), 3265-3290.

Vitanyi, P.M. (2005). Algorithmic statistics and Kolmogorov's structure function. In P.D. Grünwald, J. Myung, M.A. Pitt (Eds), *Advances in Minimum Description Length: Theory and Applications*, 151-174, MIT Press.

Wolff, J.G. (1999). Probabilistic reasoning as information compression by multiple alignment, unification and search: an introduction and overview, *Journal of Universal Computer Science,* 5(7), 418-462.

Wolff, J.G. (2000). Syntax, parsing and production of natural language in a framework of information compression by multiple alignment, unification and search, *Journal of Universal Computer Science,* 6(8), 781-829.

Wolff, J.G. (2006). Medical diagnosis as pattern recognition in a framework of information compression by multiple alignment, unification and search, *Decision Support Systems,* 42(2), 608-625.