Regularity and Structure

Alexander Botta

New York University - Courant Institute of Mathematical Sciences
Author's current address: 1310 Dickerson Road, Teaneck, N.J. 07666

Abstract

We present an approach to unsupervised concept formation, based on accumulation of *partial regularities*. Using an algorithmic complexity framework, we define *regularity* as a model that achieves a compressed coding of data. We discuss induction of models. We present induction of finite automata models for regularities of strings, and induction of models based on vector translations for sets of points. The concepts we develop are particularly appropriate for *natural spaces* - structures that accept a decomposition into recurrent, recognizable parts. They are usually hierarchical and suggest that a vocabulary of basic constituents can be learned before focussing on how they are assembled. We define and illustrate: *Basic regularities* as algorithmically independent building blocks of structures. They are identifiable as local maxima of compression as a function of model complexity. *Stepwise induction* consists in finding a model, using it to compress the data, then applying the same procedure on the code. It is a way to induce, in polynomial time, structures whose basic components have bound complexity. *Libraries* are sets of partial regularities, a theoretical basis for clustering and concept formation. Finally, we use the above concepts to present a new perspective on *explanation based generalization*. We prove it to be a language independent method to specialize the background knowledge.

1. Introduction

Unsupervised learning aims at the discovery of laws that govern the structure of data. It succeeds only if the data is constrained by some regularities, that is, not totally random. In this case, the data can be encoded in a shorter representation. The algorithmic information theory [4,5,22], and the related concept of minimal length encoding [1,3,12,19,20] will provide the framework for defining regularity. Our main concern is learning the structures of those spaces that can be decomposed in recurrent, recognizable parts. Expressing the whole depends on being able to name the parts. For example, the syntax of a sentence is based on the intermediate concept of word. The concept of rectangle illustrated by vectors of pixels depends on the intermediate concepts of pixel adjacency and line segment. Such structures are usually hierarchical, involving several levels of vocabularies. They suggest learning at first a dictionary of building blocks, and only then concentrating on how they are assembled. We call them *natural spaces*. In [1] we looked at a robot that can move around a room and apply simple actions (push, rotate, etc.) on simple-shaped blocks. By capturing important regularities in the flow of its perceptions, the robot constructs an ontology of its universe. We shall investigate here ways to capture and combine regularities in natural spaces. This may allow the proverbial blind men to put together an accurate representation of their elephant.

2. Complexity, Information, Sophistication

In this section we review some definitions from [4,5,22]. Let s,t be strings. $P_U(s)$ is the probability that a universal Turing machine U will produce s, if it starts with any self-delimiting program string on its program tape. $K_U(s)$ is the length of the shortest such program. Since a fixed size routine enables another machine V to simulate U, we drop 'U', assuming that the equations involving K(s) are correct up to a constant term. K and P are related through $P(s) = 2^{-K(s)}$. K(s) measures the *algorithmic complexity*, or the *algorithmic information* of s. K(s,t), the *join complexity*, is a minimal length program for the string "s,t".

K(s:t) = K(s) + K(t) - K(s,t) is the *mutual information* of s and t. K(s|t) = K(s,t) - K(t), the *conditional complexity*, is the length of a minimal program that has to be added to a generator of t in order to obtain a program for s. An infinite string s is *k-random* iff, beyond a certain length, none of its prefixes can be compressed more than k bits.

Random data has no regularity, thus nothing to learn from it. What we expect to discover in an apparently random universe are constraints that prove that certain rational predictions are still possible. But, there is almost nothing to learn from trivially regular data either. Take, for example, two infinite strings: $s_1 \in (a+b)^\infty$ is random and $s_2 = a^\infty$; s_1 has no structure and $K(s_1)$ is infinite, s_2 has a trivial one and its $K(s_2)$ is almost null. They are equally uninteresting since they have simple explanations: s_1 is *random*, s_2 is *a string of a's*. They both lack *sophistication*, a concept formalized in [14]. He assumes that observations derived from a physical universe are produced by processes that have both a deterministic structure and a random component. Let us review some of his definitions.

D, S are sets of strings, s_n is the length n prefix of s, and $d_1 \subset d_2$ denotes that d_1 is a prefix of d_2. A function p:D->S is a *process* if $d_1 \subset d_2 \Rightarrow p(d_1) \subset p(d_2)$. Processes provide generative explanations for strings. If a minimum size process p, and an infinite random sequence d are found such that p(d) = s, then p contains the *structure of s* and d contains

the *random choices needed to instantiate p to s.* If p(d) = s, then (p,d) is a *description* of s. The algorithmic complexity, can be redefined as K(s) = min {|p|+|d| / (p,d) is a description of s}. If p(d) = s, p and (p,d) are *c-minimal* iff |p|+|d| < K(s)+c. A *compression program* is a process that is c-minimal for every prefix of s. *Sophistication*(s)=min{|p| / p is a compression program for s}

Let Λ be 0 length program. A string α is random iff (Λ,α) is a minimal description for it, thus its sophistication is 0. [14] shows that the definition of compression program is independent of the universal Turing machine. Therefore *if it exists, it does determine the structure of a string.* Consequently, a theory currently held valid need not be refuted by new data in order to be replaced. A more complex one may be a better explanation, if it achieves higher compression.

3. Regularities and Models

We shall use the above concepts for their expressive power, but, since K(s) is not computable, we shall operate with constrained versions of them. Here are two precedents. [16] introduced a definition for K restricted to a class of finite state machines instead of universal Turing machines. [12] defined a similar complexity for the elements of language L. A grammar G for L can be seen as a machine executing a string of decisions, that control the application of G's productions, and derives an element x∈ L. K(x) is approximated by Σlog(n_i) where the i^{th} step is a decision among n_i alternatives. Therefore, we accept a restricted class of computations as an implicit bias, and use **plex(x)** to denote an approximation of **K(x)**.

We shall claim the following points: **(a)** A short but non minimal description (p,d) of a string still captures a fragment of its structure. A residual of the string's structure should be inducible by finding compressing descriptions for p and d (*section 8*). **(b)** An agent executing a random walk through a well structured universe should be able to induce a minimal description of its infinite string of sensorial information. This description should separate the randomness introduced by the exploratory walk, from the fixed structure of that universe. [1] **(c)** A Bayesian argument shows that, the most promising generalization of a single string s is its *compression program* p. This will provide a new perspective on explanation-based generalization (*section 10*).

Intuitively, regularity can be used for prediction. Given partial knowledge, data can be reduced to a shorter code that embodies the missing knowledge. Let S, C be sets of strings. S contains descriptions of objects, and C contains codes. M is a class of processes m:C->S. We introduce now our main definitions:

(i) A *model* m for a string s is a process that is able to compress some fragments $\{w_i\}$ of the string s, into the code strings $\{c_i\}$. It captures a *regularity* of s:

$$m(c_i) = w_i \quad \& \quad \Sigma|w_i| > \text{plex}(m) + \Sigma|c_i|$$

(ii) *parses*(m,w) <=> ∃ c ∈ C such that m(c)=s and w⊂ s

(iii) *code*(m,w) = min { c / w⊂ m(c) }

(iv) $gain_w(m) = |w| - (|\text{code}(m,w)| + \text{plex}(m))$

(v) $press_w(m) = gain_w(m) / |w|$ (*compression factor*)

(vi) A model that parses an entire string is a *complete model* for that string.

(vii) A *partial model* is one that can account for only some fragments of the string. For example, the rule *aab is always followed by bbab or bbb.* Also, an agent exploring a partially known territory will alternate among confusion, recognition of a familiar space, and prediction within that space. A partial model can be easily extended to a complete one by using the identity function on unparsable fragments.

(viii) A complete model that achieves the maximum compression represents a *complete regularity.* It is a *compression program* and, consequently, it reduces the data s to a random code c. All other models are *partial regularities.*

Example 3.1 [1] Let s be a string in {0,1}* where the i^{th} position is 1 if *i* is prime and 0 otherwise. Here are three models: $\mathbf{m_1}$ enumerates the prime numbers. It can generate the entire string. It is a complete model, but also a complete regularity since it captures the entire structure of s. There is no random component to s, thus m_1 reduces s to a null code. $\mathbf{m_2}$ produces a 0 in position *i*, if *i* is even, and uses a code string in {0,1}* to determine what to produce in odd positions. This is a complete model that captures only a partial regularity. $\mathbf{m_3}$ only predicts that every 1 in s is followed by a 0, except for the first 2 positions. This is a partial model. It cannot parse the entire string but only the 10 sequences.

4. Induction of Regularities

We focus now on *incremental induction of complete models* for an infinite string s. A fixed order enumeration would discard a hypothesis as soon as it fails to parse a prefix of s, loosing information on initial successes. Therefore we start with a small set of simple models as hypotheses and *refine* them when they fail. Let m be a model that parsed the prefix w but failed on the next symbol a. Assume a function *refine* that takes m, a, and some information on how m failed, and returns a set of models m' that can parse wa. An *incremental refinement* procedure requires $press_{m'}(wa)$ to be computable from $press_m(w)$. In [1] we gave sufficient conditions for this approach. We also showed that,

(i) If h_n is the hypothesis with the highest $press_{sn}(h_n)$ after reading the prefix s_n (the best at step n), and s=m(c) where m is a bijective process of finite complexity, and c is k-random code, then $\lim_{n->\infty}[\ press_{sn}(h_n) - press_{sn}(m)\ ] = 0$

[1] presents an *incremental refinement* algorithm to induce finite automata models. It maintains a set H of hypotheses (A,q,g) where A is transition graph, q the current state, and g the current gain. Initially H={(A,q,0)} where A has one state and no transitions. Let ***a*** be the current symbol read from s. Parsing *fails* when there is no ***a***-transition from the current state. An ***a***-transition is added to the *failed hypothesis*, in all possible ways. Maintaining, for every state q, the number of times q was reached in parsing, allows us to com-

Symbols	Models discovered	Plex	Press
24	m_1: (aab+abbb)*	6	0.5
50	m_1	6	0.6
	m_2: (aab+abbb(aab+abbb))*	12	0.5
241	m_1	6	0.69
	m_2	12	0.67
	m_3: (aab(aab+abbbabbb)abbb)*	19	0.84

pute the gain of a refinement without running it. When H is too large, hypotheses with low gain are discarded. Theorem (i) insures that, if the right hypothesis is in H, it is likely to survive the pruning.

Example 4.1 [1] We applied this algorithm on a string generated by taking random decisions for the '+' in the regular expression E= (aab (aab + abbbabbb) abbb)*. Complexity of models was measured by the number of transitions. Two less performant complete models, m_1 and m_2, emerged before the one equivalent to E, m_3, was found. One of the runs is summarized in the above table.

5. Basic Regularity, Hierarchy, Vocabulary

Searching for partial models is particularly appropriate when the target structure is based on a finite set of smaller substructures. [6] presents a related idea. He defines the d-diameter complexity of a spatial structure X (a container of gas, a crystal, a live cell etc.) as:

$K_d(X) = \min [K(\alpha) + \Sigma_i K(X_i)]$ where $\{ X_i \mid \text{diameter}(X_i) \le d \}$ is a partition of X. It is a minimum length description of X as an assemblage of parts, without cross-references, each not greater than d. When d grows from 0 to diameter(X), $K_d(X)$ decreases since it can take advantage of more complex substructures and more distant correlations. Changes occur at the points where d exceeds the diameters of *important substructures* of X. Therefore $K_d(X)$ provides a kind of *spectrogram* of the object X.

An ideal vocabulary of building blocks would contain elements as different from each other as possible, since any substantial amount of shared information among blocks could support a smaller vocabulary. Therefore they should be *algorithmically independent*. This allows us to define a criterion for identification of good building blocks:

(i) *If a structure is composed from highly algorithmic-dissimilar constituents, then those constituents can be identified, during model refinement, by local maxima of compressibility.*

Here is an argument for our claim. Let w_1, w_2, w_3 be fragments of a string s. Let m be a model for w_1 but not for w_2 or w_3. We assume that *w_1 and w_2 are algorithmically independent* (dissimilar). Their mutual information is very small. Thus we can expand a minimal program for w_1, in order to produce w_2, but the addition will be almost as large as a minimal program for w_2 alone. Also, let *w_1 and w_3 have a similar structure*. Thus program for w_1 needs only a small addition to generate w_3, since the mutual information of w_1 and w_3 is almost the same as the total information of w_3. The two assumptions can be expressed as:

(a) $K(w_1 : w_2) < \varepsilon \iff K(w_2 \mid w_1) > K(w_2) - \varepsilon$

(b) $K(w_3 \mid w_1) < \varepsilon \iff K(w_3 : w_1) > K(w_3) - \varepsilon$

Let m' be a minimal refinement of m that parses w_1w_2. Unless plex(m') = plex(m) + K(w'|w), that is unless m' is substantially more complex that m, m' will not be able to maximally compress w_2. But (a) makes such a refinement unlikely. Let us look at one of the smallest additions that could take m out of impasse. This is the *copy* program, that prints its input to the output, coding every string by itself. Let m' be m endowed with *copy*, and let us estimate its performance on w_1w_2. If w_2 is coded by itself, then $|\text{code}(m',w_1w_2)| = |\text{code}(m,w_1)| + |w_2|$. It follows that:

$$\text{press}_{w1w2}(m') = (|w_1w_2| - |\text{code}(m',w_1w_2)|) / |w_1w_2| =$$
$$= (|w_1w_2| - |\text{code}(m,w_1)| - |w_2|) / |w_1w_2| =$$
$$= (|w_1| - |\text{code}(m,w_1)|) / |w_1w_2| = (|w_1| / |w_1w_2|)\text{press}_{w1}(m)$$
$$= \text{press}_{w1}(m) / |w_2| < \text{press}_{w1}(m)$$

Therefore m' can parse anything but can compress less. Its performance decreases with the length of w_2 and assumption (a) forces w_2 to be a relatively long string. (If it were a short one, its complexity would be small, hence it would have been derivable by a program for w_1 with a small addition).

To parse w_3, m has to be refined too. But (b) shows a small addition to m yields a high compression performance on w_1w_3. Certain classes of models may not allow expression of that small addition as a refinement. For example, a recursive invocation is a small addition not expressible in a finite automata class. Nevertheless this discussion shows that *there is a trade-off point beyond which models should not be refined any more.* If the encountered *novelty* w_2 is small (that is, similar to the model), then it is likely to be incorporated into the model without a serious loss in compressibility. If the novelty is large, all small additions to the model will degrade compressibility; looking for a different model is a better solution. ♦

This principle justifies the following definition of *basic regularity*. Let us consider a refinement that increases complexity by a small quantum. Let m << m' denote that m' is obtained from m through one or more refinement steps. Let $0<\lambda<1$.

(ii) We define a *λ_basic regularity* of s to be a model m with the property: **If** $m' >> m$ & $\text{press}_s(m') \ge \text{press}_s(m)$ **then** $\exists m''$ such that $m' >> m'' >> m$ & $\text{press}_s(m'') \le \text{press}_s(m) - \lambda$

The parameter λ affects the separation between regularities. A higher λ forces a higher separation. ♦

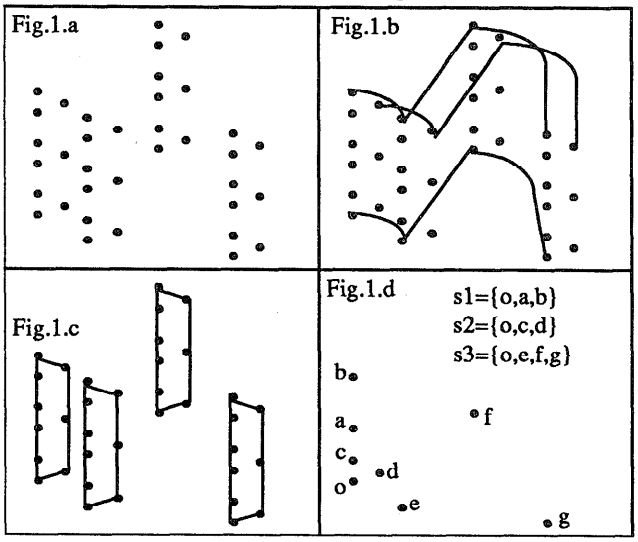
Example 5.1 A similar criterion is used in [2]. It describes an algorithm (Procrustes) that partitions a text into fragments. It starts with 1 letter fragments and refines them by concatenation with continuations encountered in the text.

The refinement stops when the average information necessary to specify the letter following a fragment approaches the average information necessary to specify that letter independently. The algorithm converges to a set of basic regularities. For example, without any background knowledge, an Italian text yields a set of proper Italian syllables; a text of Morse code yields the codes of the individual letters. ♦

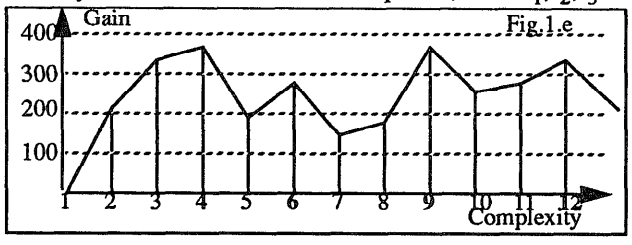
Example 5.2 [13] contains another empirical application of this principle. It searches for substructures in organic chemistry formulae (for example, benzene rings). Identification of proper substructures takes into account a compression factor, a substructure density, its connectivity to its neighbor atoms, and a coverage factor. These parameters determine a trade-off point between a candidate substructure and its possible extensions (refinements). ♦

Example 5.3 [1] Plotting compression as a function of complexity for the automata in example 4.1 shows that, after about 250 steps, the models m1,m2,m3 become well isolated local maxima. Indeed, they reflect the main components of the string: the sequences aab and abbb ♦

Example 5.4 [1] Let S be a set of 36 points in a two dimensional space (1.a), given as a set of vectors $[x_i,y_i]$ of 8 bit integers. A routine $r=\{p_1, ..., p_k\} \in R$ is a set of vectors. It has one argument p of type point. Let $r(p_0)=\{p_0+p_1, ..., p_0+p_k\}$. Also plex(r)=16k. A routine translates its points to different positions. We shall look at models that are set of routines of the same complexity (see next section). For this structure we shall study gain(m) by considering, in turn, routines with increasing numbers of points.



Every vector in S is the sum of 3 points, from s_1,s_2,s_3 respectively (fig,1.d). Look now at the local maxima of gain (fig.1.e). The routine with 4 points (fig.1.b) identifies s_3. The one with 9 points (fig.1.c) identifies s_1+s_2. ♦



Induction of basic regularities can assist a shift in representation bias. They provide a vocabulary for a new, more compact representation of data. The next section establishes a connection between such a vocabulary and the emergence of categories in conceptual clustering.

6. Libraries of Regularities

A *library of routines* can represent *a set of partial regularities*. Given M, a class of models, let $\mathcal{L}(M)$ be the power set of M. A model $L \in \mathcal{L}(M)$ can be seen as a library of routines each of which can generate, with appropriate input data, some fragments of the string s. If the regularities in L do not cover the entire string, we can always add to L the identity function, to turn the partial model into a complete one. The string s will be coded by the model into a sequence of invocations and routine parameters. The performance of L will take into account the size of reference, that is, information neccessary to distinguish among the routines. If s is a concatenation of fragments $w_{ij} = r_i(code_{ij})$, then

(i) $gain_s(L)=\Sigma_{ri \in L}(\Sigma_j (|w_{ij}|-|reference_i|-|code_{ij}|)-plex(r_i))$

Example 6.1 [3] defines two types of library models for finite strings: *sequences* and *classifications*. Classifications are partitions of the alphabet into classes of symbols. A library corresponding to a classification would have a routine for each class. Routine arguments would distinguish among symbols within a class. Sequences and classifications can be composed in both directions to generate more complex routines: classes of sequences and sequences of classes. ♦

Example 6.2 [1] Let $S=\{x_1, ... x_n\}$ be a set of vectors of natural numbers. Given a fix point x_0, S can be expressed as $S'=\{y_1, ... y_n\}$ where $y_i = x_i-x_0$. This is equivalent to defining a routine r_{x0} such that $x_i = r_{x0}(y_i)$. If $|x_i|$ is the sum of representation lengths for its numeric coordinates, the gain of representing S by r_{x0} is

$gain_S(r_{x0})= |S| - |S'| - |reference(r_{x0})|= \Sigma_i|x_i| - \Sigma_i|y_i| - |x_0|$

A partition of S into the sets C_j, j=1,...,k together with a set of points $\{x_{01}, ... x_{0k}\}$ such that $(\forall j=1,...,k)\ gain_{Cj}(r_{x0j}) > 0$ is in fact *a clustering* of S. This is not surprising, since finding any good clustering for S is equivalent to discovering a part of its structure. Let $H(x_i|C_j)$ be the information needed to retrieve $x_i \in C_j$, knowing C_j. A good clustering would minimize $\Sigma_{j=1,k}(|C_j| + \Sigma_{(xi \in Cj)}H(x_i|C_j))$ which is equivalent to maximizing the gain of representing S by the library of routines $\{C_1,... ,C_k\}$. Clusters defined by a central point require simple routines. Libraries allow more complex examples. Let $S=\{x_1,... x_n\}$ be a cloud of points arranged around a certain curve C. Each can be expressed as $x_i = y_i + z_i$ where z_i is a proximal point on C. But, z_i can be reduced to the equations for C and a scalar parameter ζ_i. Thus, a more complex routine can reduce each x_i to $r(z_i,\zeta_i)$ ♦

Assuming that we have a parsing procedure for each routine, we have to decide which routine in the library, if any, should parse the current string fragment. Some domains have well defined fragments. For example, attribute-value vectors are classified one at a time. Texts, visual images, speech, or handwriting, however, do not have well defined fragments. Routines represent regularities. *Fragmentation* is the problem of recognizing an instance of a regularity and decide its boundaries. A Bayesian approach (based on algorithmic probability) yields a minimal description length criterion: given the fragment w, choose r to minimize K(r) + K(w|r). Given a class of models, and incompressible codes, this amounts to $\min(\text{plex}(r)+|\text{code}(r,w)|) <=> \max \text{press}_w(r)$.

A library operates *a classification on fragments*. Every routine in a library defines a class (or cluster) that contains all fragments parsed by it. The routine captures a part of the common algorithmic information shared by those fragments, ideally, their structure. Therefore libraries are theoretical models for classification. A partial model for a finite amount of data may turn out not to be a basic regularity in view of further data. Research in conceptual clustering, (for example, the COBWEB and CLASSIT algorithms surveyed in [9]) encountered the problem and described an evolution of clusters through (a) *merging* or (b) *splitting*.

(a) Two clusters that were initially well separated may merge if new points are added in the space between them. Routines should adapt as well. For some representations (vectors, first order formulae, etc.) a merging can be defined as a minimal common generalization, for other representations this is an open problem. If the past fragments were available, a common model could be induced from them.

(b) A routine r maps all fragments $\{ x_i \mid i \in I \}$ parsed by it into the codes $\{c_i\}$. If a regularity allows the set $\{c_i\}$ itself to be compressed by a library $\{p,q\}$, that is, $c_i = p(d_i)$ for $i \in I_1$ and $c_i = q(e_i)$ for $i \in I_2$, then r can split into the pair $r_1 = p \otimes r$ and $r_2 = q \otimes r$ (where $\otimes$ stands for composition of functions). This operation can be interpreted as specialization since the classes defined by r_1 and r_2 are contained in the class defined by r. In many approaches of learning from examples, specialization of hypotheses is prompted by negative examples. Here the more specific r_1 and r_2 are the result of positive examples! (see this distinction in section 9).

7. Stepwise Induction

An experiment reported by [10] showed that human subjects try to recognize a regularity, and then focus on the residual parts of the data, that are not accounted for by that regularity. Our approach supports this view. Once the compressing description $s=m_0(s_0)$ was found, the residual information is contained both in the model m_0 and the code s_0. A second induction step can be attempted on each:

(a) find m_1, s_1 such that $s_0=m_1(s_1)$ & $\text{plex}(m_1)+|s_1| < |s_0|$

(b) find m^1, s^1 such that

$(\forall x)(m^1(s^1,x)=m_0(x))$ & $\text{plex}(m^1)+|s^1| < \text{plex}(m_0)$

While (a) is a further induction on strings (this time code strings), (b) depends on the availability of a model class appropriate for descriptions of object of class M. (For example, let M be the set of finite automata, and s be a string of balanced parentheses like (((())()()((((()))))(())()). M will never capture the balance of parentheses. That will be visible only to a class of models able to recognize similar parts in the transition graph of m_0 and code them as subroutines.)

Let *stepwise induction* denote this approach. It provides a natural way of learning hierarchical structures one level at a time. Moreover, the levels of a hierarchy might share a large amount of mutual information. (For example, structures that accept models like m⊗m⊗m⊗m). This suggests that the sequence of levels itself may become an object for induction. Many interesting learning problems are intractible. However, *for a k level hierarchical structure, where each level has complexity less that C, a stepwise induction algorithm will take* $O(k2^C)$ *steps.*

Example 7.1 [1] Let us stop the algorithm in example 4.1 at the first basic regularity: m_1. Let m_1 code *aab* by *a* and *abbb* by *b*. The same algorithm applied on the code will find m_1 again because the string s is a hierarchy: $m_3 = m_1 \otimes m_1$. ♦

Example 7.2 [1] The set of points in example 5.4 has a hierarchical structure too. The code produced by the 4 point routine (fig.1.b) is a set, c_1, of 9 points. By applying the same procedure to it, we find the s_2 triangle (fig.1.d) as a 2nd level basic regularity. This 2nd model will code c_1 into a 3 points set identical to s_1. ♦

[2,3] apply a similar procedure on texts.

8. Explanation Based Generalization

How is background knowledge used in learning? In [1] we defined *analysis* as the part of the learning process that relates the new datum (w) to the background knowledge (B) in order to separate two components: K(B:w) - the part of w that is *recognized as known*, and K(w|B) the part of w that is *really new*. Analysis is well defined when B is given as a formal system that generates the set of learner's expectations. Explanation based generalization (EBG), [8,17], is such a case. Let us see how it looks in our framework:

Take a formal system T with axioms A that generates a language L. For every object $e \in L$ there is at least one sequence of operations d that derives e from A. Some arbitrary choices (among applicable rewriting rules, etc.) are made in order to build d, rather than some other derivation. Let c be this set of choices. Then c *contains that part of the structure of* e *that is not determined by the fact that* e *is an element of* L. In our terms, T is a *model* of e and c=code(T,e). The proof c represents the novelty K(e|T). Assume now that examples $e_1 \ldots e_n$ from L illustrate an unknown concept $X \subseteq L$. How can we use in the induction of X the knowledge that we already have in T? We derive each e_i from T, that is, we use A as a model to parse them, and we extract their codes $\{c_i\}$. *If there are any common features among e's that distinguish them from other elements of L, they must be represented in*

the particular choices used to derive them. Therefore the induction should operate on $\{c_i\}$. However, the target of generalization should not be only the constants but the entire structure of the proof! [7,21] are first attempts to identify recurrent substructures that allow the compression of the derivation graph.

Example 8.1 [1] Let A= $\{A_1(X,Y)$: p(X,Y)=>p(f(X),Y), $A_2(S,U)$:p(S,U)=>q(S,U), $A_3(V,Z)$: q(f(V),Z)=>q(V,g(Z))$\}$ All variables are universally quantified. Consider the proof p(a,b) => p(f(a),b) => p(f(f(a)),b) => p(f(f(f(a))),b) => q(f(f(f(a))),b) => q(f(f(a)),g(b)) => q(f(a),g(g(b))) => q(a,g(g(g(b))))). A possible model for the proof structure is $h(X,Y,N) = A_1(X,Y)^N A_2(X,Y) A_3(X,Y)^N$ ♦

When A cannot derive **e**, *the failed attempts* contain the novelty. They can be used to refine the model [11,18].

Traditional approaches to learning need negative examples to specialize a theory. We show now how EBG can do this with positive examples. The theory is a model **m**:C->E. Endowing a learner with **m** as background knowledge is equivalent to constraining its expectations to **m**(C), the range of **m**. Let **e** be an example of a target concept. Therefore we assume that e is parsable and **e=m(c)**. EBG generalizes **c**. In absence of other constraints, the natural generalization of **c** is its compression program. Short of that, any model **f** would do. Let **c = f(c')**. Let C' be the range of **f**. The only constraint on generalizing **c** to **f** is $C' \subset C$, so that **m** can apply! The new model, **m'=f⊗m**, is a specialization of **m** since it constrains **m**(C) to **m**(C'). Our perspective shows EBG to be a valid method beyond the usual first order logic. [15] reached a similar conclusion by a different path.

Example 8.2 [1] Let G_0 be a grammar with the following productions: {(1) S -> ε (2) S -> aSa (3) S -> bSb}. $L(G_0)=\{w\overline{w} \mid w\in(a+b)^n$ & $\overline{w}$=reverse(w)$\}$ Production are numbered to allow G_0 to code a string in $L(G_0)$ by a string of production numbers, in the order of their applications. Let us examine the string

e = aabbaaaabbaabbbbaabbaaaabbaa = G_0(**c**),

where **c** = 22332222332233331

Assume we found in **c** the following regularity: *c is made up of pairs 22 and 33, except for the last character.* This enables us to compress it into **c** = G_1(**c'**) where **c'** = 232232331 and G_1= {(1) S -> 1 (2) S -> 22S (3) S -> 33S}

The structure of **e** is contained in the new theory $G_2=G_1 \otimes G_0$. We operate this composition by noting that G_1 is forcing productions 2 and 3 in G_0 to be applied twice in a row. Thus G_2={ (1) S -> ε (2) S -> aaSaa (3) S -> bbSbb}. In conclusion G_0 was specialized to G_2 while the set {**e**} was generalized to

$L(G_2) = \{ w\overline{w} \mid w\in(aa+bb)^n$ & $\overline{w}$=reverse(w)$\} \subset L(G_0)$ ♦

REFERENCES

[1] Botta, A. 1991. *A Theory of Natural Learning.* Ph.D. Thesis, NYU, Courant Institute of Mathematical Sciences.

[2] Caianiello, E.R.; Capocelli, R.M. 1971. *On Form and Language: The Procrustes Algorithm for Feature Extraction.* Kybernetik 8, Springer Verlag

[3] Caianiello, P. 1989. *Learning as Evolution of Representation.* Ph.D. Thesis, NYU, Courant Institute of Mathematical Sciences

[4] Chaitin, G.J. 1975. *A Theory of Program Size Formally Identical to Information Theory.* J. of the A.C.M. 22(3)

[5] Chaitin, G.J. 1977. *Algorithmic Information Theory.* IBM Journal of Research and Development

[6] Chaitin, G.J. 1978. *Toward a mathematical Definition of "Life"*, in Levine, R.D.; Tribus, M. : The Maximum Entropy Formalism, M.I.T. Press

[7] Cohen, W.W. 1988. *Generalizing Number and Learning from Multiple Examples in Explanation Based Learning.* Proceedings of the 5th International Conference on Machine Learning.

[8] DeJong, G.F.; Mooney, R. 1986. *Explanation Based Learning - An Alternative View*, Machine Learning 1

[9] Gennari, J.H.; Langley, P.; Fisher, D. 1989. *Model of Incremental Concept Formation.* Artificial Intelligence 40

[10] Gerwin, D.G. 1974. *Information Processing, Data Inferences, Scientific Generalization.* Behavioral Sciences 19

[11] Hall, R. 1988. *Learning by Failing to Explain: Using Partial Explanations to Learn in Incomplete or Intractable Domains.* Machine Learning 3 (1)

[12] Hart G.W. 1987. Minimum Information Estimation of Structure, Ph.D. Thesis MIT, LIDS-TH-1664 *(1987)*

[13] Holder L.B. : Empirical Substructure Discovery, *Proceedings of the 6th International Conference on Machine Learning (1989)*

[14] Koppel, M. 1988. *Structure.* in Herken,R. (ed.): The Universal Turing Machine - a Half-Century Survey. Oxford University Press

[15] Laird, Ph.; Gamble, E. 1990. *Extending EBG to Term-Rewriting Systems.* Proceedings of the 8th National Conference on AI

[16] Lempel, A.; Ziv, J. 1976. *On the Complexity of Finite Sequences.* IEEE Trans. on Information Theory, IT-22 (1)

[17] Mitchell, T.; Keller, R.; Kedar-Cabelli, S. 1986. *Explanation Based Learning - A Unifying View.* Machine Learning 1

[18] Mostow, J.; Bhatnagar N. 1987. *Failsafe - A Floor Planner That Uses EBG To Learn From Its Failures.* Proceedings of the 10th IJCAI

[19] Pednault, E.P.D. ed. 1990. *Proceedings of the Symposium on TheTheory and Applications of Minimal Length Encoding.* Preprint.

[20] Rissanen, J. 1986. *Stochastic Complexity and Modeling.* The Annals of Statistics 14

[21] Shavlik, J.W. 1990. *Acquiring Recursive Concepts with Explanation Based Learning*, Machine Learning 5 (1)

[22] Solomonoff, R.J. 1964. *A formal Theory of Inductive Inference (part I,II).* Information and Control 7