Report No. UIUCDCS-R-85-1198

UILU-ENG-85-1704

Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search

by

Larry Rendell

85-2

January 1985

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search

by

Larry Rendell

January 1985

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801-2987

This work was supported in part by the National Science Foundation under Grant No. DCR 84-06801

SUBSTANTIAL CONSTRUCTIVE INDUCTION USING LAYERED INFORMATION COMPRESSION: TRACTABLE FEATURE FORMATION IN SEARCH

This paper addresses a problem of practical inductive inference which is more difficult than any comparable work in AI. The subject of the present research is a hard *problem of new terms*, a task of realistic *constructive induction*: mechanized *feature formation* from data represented in elementary form. A *feature, attribute*, or *property* such as "piece advantage" in checkers is much more abstract than an elementary descriptor or *primitive* such as contents of a checkerboard square. Features have often been used in *evaluation functions*; primitives are usually too detailed for this. To create higher level features from primitives (i.e. to restructure data), a new form of clustering is used which involves layering of knowledge and invariance of *utility* (i.e. usefulness in attaining a goal). The scheme, which is both model- and data-driven, requires almost no background, domain-specific knowledge, but rather constructs it. The method achieves considerable generality with superior noise management and low computational complexity. Although the domains addressed are difficult, initial experimental results are encouraging.

1. INTRODUCTION

A fundamental problem in AI is the automation of *inductive inference* [3, 16, 25].[1] Induction can be described as generalization from particular cases [6, 16], as learning categorizations from examples or observation [16, 19, 21], as intelligent compression of massive data [25, 32, 33], or as discovering regular, coherent characterizations of cases or *events* [16, 24 25].[2] Whatever the interpretation, induction begins either with events or with partially formed groupings, and creates one or more *classes*. These meaningful sets are also called *categories* or *concepts*. They are usually expressed concisely using some descriptive language such as predicate logic [2, 16]. A concise representation of knowledge is imperative for reasons of space and time efficiency; in practical application detailed information is simply too expensive either to store or to acquire [6, 25].

While mechanized induction is crucial,[3] the problem is inherently difficult [1, 6, 32]. This can

1. The problem of induction is also a basic study outside of AI. It has occupied scholars in philosophy [19], psychology [14, 28], pattern recognition [2, 3, 31, 33], and other fields [32]. An increasingly prevalent attitude is that the same basic principles underlie both machine and human induction (and other learning) [15, 16]. However, no theory has yet emerged which can explain human effectiveness and efficiency.

2. Formally, given a set S of individual events, induction is the inference of a larger class T, such that $S \subset T$. S and T may be specified by *extension* (as an exhaustive list of events) or by *intention* (as a combination of properties to be satisfied for membership) [32, 33]. Often the original set S is given by extension and the *hypothesis* T by intention. Since T is a generalization it may not be true; associated with T is its *credibility*, the estimated veracity of the induction.

3. In an expert system, computer generalization allows mechanized knowledge acquisition [16, 25], which may reduce costs. Present expert systems are prone to unexpected error, whereas induction would increase reliability and obviate maintenance. Moreover, even in its primitive state, automated induction has outperformed knowledge engineering approaches [18].

be explained in various ways, all involving the fact that the end product of induction is one or more purposeful classes. In general, very few of *many* hypothetical categorizations are plausible, useful, or sensible, i.e. *credible*. A credible hypothesis may be sought in a space of category *descriptions*. In this view of the inductive process, search is conducted through a set of expressions which are systematically related to one another to facilitate their explicit formation (such expressions are otherwise only implicit, because of their immense number) [2,6]. In this search formalization, exploration of plausible hypotheses must cope with exponential growth of their explicit representations. Another view of the inductive problem involves *classification* [32,33]. Here the number of ways of combining events into categories is similarly extreme. This aspect of induction may be important because the number of classifications may be finite in some cases when the number of descriptions is not, and thus it may be possible to ascertain properties of the domain and of induction algorithms only if the classification model is used. The classification view of induction is illustrated in Fig. 1. A third way of analyzing the difficulty of induction considers the quantitative problem as a qualitative one [16,19]. Classification often requires *restructuring* of data; events must be reorganized by transforming description variables. This means a change of knowledge representation (*constructive induction*, the *problem of new terms*), which is particularly difficult to automate [6,16,25]. Fig. 1 compares constructive induction with the simpler *selective* induction which needs no reorganization.

To order data, an expressive language such as predicate logic is required, but there are tradeoffs between expressiveness and efficiency (at least with current models). Greater expressive power causes a worse combinatorial explosion in search. Given any language of even moderate power, the number of possible candidates is immense, while the number of credible hypotheses is small [6,25].

One approach to this problem is to limit candidate expressions by imposing constraints [16,25,34]. These constraints may take various forms. A straightforward tactic is simply to limit the description language without confining its power too much. For example, many methods permit conjunctions but not disjunctions [7]. Another way of restricting candidate descriptions is to use some criterion to narrow search. Examples of criteria include "simplicity", quality of "fit" to the data [16,19], and "utility divergence" related to a goal in task performance [23,24,25]. Constraints may include some kind of invariance [8]. In the author's scheme the utility divergence criterion is the "dual" of utility similarity, which is a relaxed form of invariance [23,25]. Overall, constraining search for abstract knowledge has seen only limited success in terms of efficiency, effectiveness, and extensibility [7,15,34].

Generalization algorithms have been designed either for selective induction [6,7,16,21,23,30] or for quite simple constructive induction [11,16]. The true nature of this AI work has sometimes been obscure because little attention has been given to the usual methods of science: delineation of abstract phenomena, detection of relevant variables, measurement of precise relationships, and development of guiding principles. Unified views of inductive systems are scarce (though there are [6,7,16,25]). Furthermore, AI research often ignores earlier germane results, including [2,32]. Consequently no standard exists for answering important questions such as: How difficult is the inductive task being studied? How much knowledge is acquired autonomously, versus the amount given by the user? Similar questions have recently been considered elsewhere [12,27].

While the current state of induction in AI is understandable (the field being new and difficult), the time may have come for a more rigorous approach. (See [5,13,16,27] for similar sentiments.) In keeping with this goal, the present author has begun to pursue a means for comparing inductive tasks and systems. Based on [32,33], this involves a quantification of *inductive difficulty*, both for task domains and for learning systems. This attempt began in [24,25] and continues in [26].

The idea is simple: since induction creates a class T from a set of events S, the difficulty of the generalization task depends on the nature of the information compression from cases S to concept T.[4] First, the more cases T must cover, the harder it is to describe T accurately while differentiating S from negative instances of the concept (see the approximation of *conceptual knowledge* as amount of *information compression* in [24,25]). Secondly, if the attributes describing S do not support straightforward selective induction but instead must be redefined for concise expression of the class T (i.e. if constructive induction is necessary -- see Fig. 1), then inductive difficulty increases with the degree of reconstruction required (see the quantification of *irregularity* in [26]). When examined in this light, many systems (such as [11,16]) perform little mechanized induction, compared with human capability. In these systems the total number of possible generalizations is confined to be relatively small from the outset (see [26] for an analysis).

In contrast, the current research attempts a very difficult task. Not only is the amount of required information compression very large, but, more important, a large degree of constructive induction is needed. In the proposed system little domain-specific guidance is provided by the user or program. This scheme is conceptually compact and appears tractable. It is related to a system for selective induction (*PLS1*) which has produced unique results (e.g. efficient discovery of locally optimal evaluation functions [23,25]). One aspect common to PLS1 and the new system *PLS0* is the observation of a goal-oriented measure, the *utility*. The utility *divergence*, partly a function of "features" of events, and partly based on data patterns, is the criterion for induction. When this is used for categorization (conceptual clustering), the number of resulting knowledge structures is minimized [23,25]. The algorithm *automatically* produces an effect similar to a criterion using "similarity" and "fit" [16,25]; moreover this processing is inherently efficient. Still another advantage of the scheme is that it has been *incremental* since its inception.

PLS0 implements a new form of constructive induction unlike other systems in important respects: although the inductive difficulty is great, little background knowledge is given, the language of concept expression is quite general, the algorithms have low computational complexity, and the approach appears extensible. These claims are elaborated in the following sections.

4. This is a simplification. More than one class T may be involved, and the set of events S may already be partially formed into categories. For present purposes these details may be ignored without affecting the essence of inductive difficulty.

2. INDUCTIVE DIFFICULTY: AN EXAMPLE IN HEURISTIC PROBLEM SOLVING

An example of a product of induction is the concept *pair adjacency:* two diagonally juxtaposed, friendly men in checkers (Fig. 2). In the description of such a concept, certain spatial patterns are essential (adjacency) while other aspects are immaterial (location of the pattern and presence of extraneous pieces). Other "useful" concepts in checkers are piece advantage, mobility, center control, etc. These are useful because they relate to winning, and the ultimate concept in a game is the strength or *utility* for one player. (Similar utilities arise is other kinds of problems [25].) Utility is the most abstract concept desired.

Utility is expressed using an *evaluation function H* which ranks *states* (here board configurations).[5] In the terminology of induction, states are elementary descriptions (events), and the evaluation given by H is the most concise description. Pair adjacency and other properties (piece advantage, etc.) become attributes or *features* to be combined in H. Feature values are intermediate between events and utilities, as the following shows.

As it happens, a resolution of roughly 100 distinct *utility classes* is sufficient to differentiate among moves (see the discussions in [24, 25]). Fig. 3 reflects this; here utility values lie in the interval [0, 1] and are given to two decimal places. In contrast, an event (board configuration or state) is a vector of 32 *primitives* indicating the contents of each permissible square (black king, black man, vacant, white man, white king; or -2, -1, 0, 1, 2), and there are roughly 10^{20} legal states. Full inductive learning would require the assignment of each of these configurations to its proper utility class, for an average of about $10^{20} / 100 = 10^{18}$ in each class. In terms of primitives, a (logic) description of such a class would be highly irregular, involving a combination of an immense number of terms.

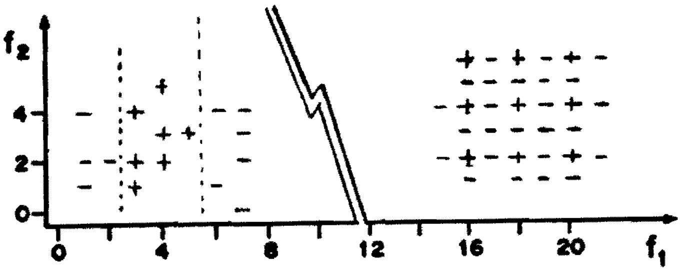

Figure 1. **Degrees of inductive difficulty. Positive (success) events occur in (e.g.) a two-dimensional feature space. A simple case of classification (clustering) requires only a few boundaries to be inserted (left). This is *selective induction*. Complex cases may involve varying degrees of heterogeneity, which logical reconstruction may unravel to provide concisely described concepts. An example is the concept defined by variables f_1 and f_2 both being even (right). This is *constructive induction*.**

5. Using an evaluation function for search is fully general [9]. From this implicit knowledge an explicit plan or strategy might be mechanically derived [10, 25].

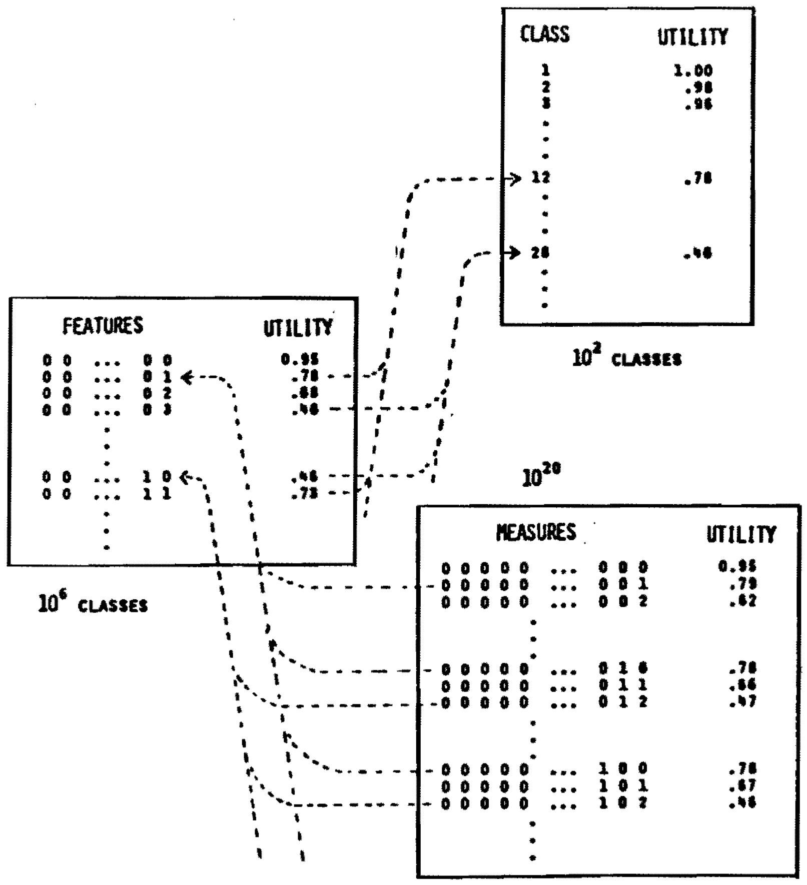

Figure 3. Levels of conceptual knowledge in search. Elementary data represent fully detailed knowledge but no abstraction, and are massive and infeasible to gather. At the other extreme, maximal induction represents utility classes concisely (using predicate logic, although the classes are simply enumerated here). Intermediate representations facilitate expression, e.g. features discriminate utility well and bear a smooth relationship with it. In contrast, primitive measurements determine utility irregularly.

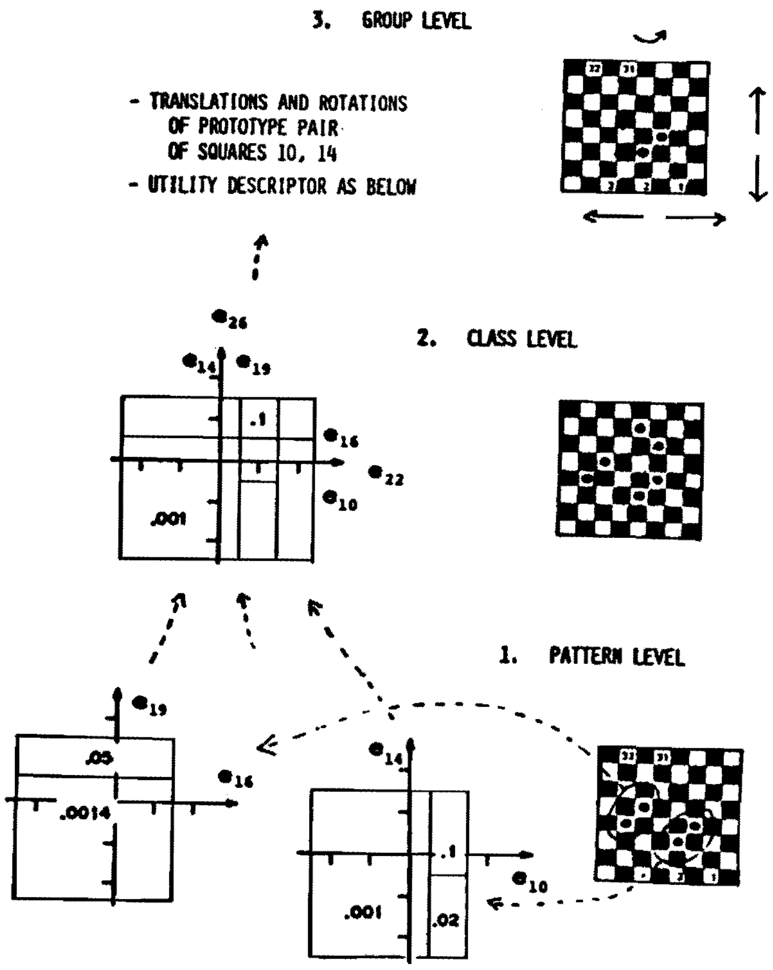
3. GROUP LEVEL
- TRANSLATIONS AND ROTATIONS OF PROTOTYPE PAIR OF SQUARES 10, 14
- UTILITY DESCRIPTOR AS BELOW
2. CLASS LEVEL
1. PATTERN LEVEL
e26
e14
e19
e16
e22
e10
.1
.001
.05
.0014
.02

Figure 2. The three level information structure of PLS0. Shown is the construction of a feature f which counts the number of diagonal juxtapositions of two friendly pieces in checkers. This begins with a primitive description of board configurations in terms of 32 primitives e_i giving the contents of individual squares. When subspaces such as e_{10} e_{14} and e_{16} e_{19} are examined and corresponding utilities are CLUSTERed (overlaid), patterns begin to emerge having common *utility descriptors* (UD's or primitive region sets). A *pattern class* results (level 2). Eventually a class may be generalized as a group of transformations (level 3).

Utility classes are much easier to describe in terms of predefined features. As Fig. 3 illustrates, the number of classes at the feature level is very roughly 10^6 (this is the size of a typical feature space [30]; see also [25]). Features exhibit a smooth relationship with utility, so the induction required is merely selective (Fig. 1), and only a few simple descriptions are needed (Fig. 4).[6] In contrast, not only are raw data descriptions (events, primitive vectors) more detailed (having 10^{20} values as opposed to 10^6), but also utility-primitive relationships are discontinuous, so the much more difficult constructive induction is needed (see Figs. 1 and 2).

This paper begins to explore a method for discovery of features from primitives, a problem hardly addressed previously (but see [8, 21, 31]). Conceptually and experimentally, the approach appears tractable and error resilient. A layered, 'divide and conquer' approach restricts complexity, not just 'generating and testing' hypotheses, but rather constructing simple ones from previously validated components.

How can irregular utility-primitive relationships be captured and generalized? Induction is infeasible without some guidance from regularities, assumed or else discovered [34]. One technique, curve fitting, is often inadequate even in selective induction with features the starting point [4, 6, 25, 30]. A more flexible approach is to record information in feature space cells [23, 30]. An important tool for inducing this knowledge autonomously is the method of *clustering*. Since it is related to PLS0's more powerful clustering for constructive induction, clustering will first be outlined as used in PLS1's selective induction (see also [23, 25]).

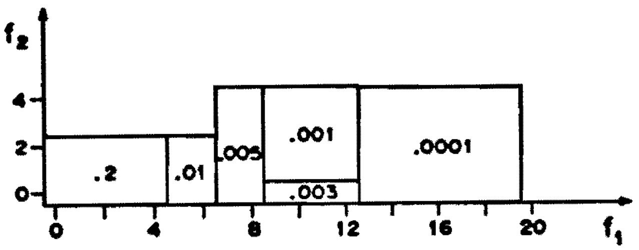

Figure 4. A region set is a partition of feature space with associated information. Shown are rectangles (concepts) r and their values (*utilities*) u for some purpose. A *region* can simply be the pair (r, u). In PLS1 a feature space region set is used as an evaluation function. In PLS0 a primitive space region set is used to create features.

3. CLUSTERING, A TOOL FOR INDUCTIVE INFERENCE

Clustering classifies data so that cases or *events* are *similar* within any class, but dissimilar across classes [1]. This technique has been used in successful learning systems by Michalski [16, 18, 19], and by Rendell [22, 23], who independently introduced what Michalski calls *conceptual clustering*. This takes into account not only independent variables (feature values), but also the environment (e.g. observed *utility*).

6. Because most of the "knowledge" resides so regularly in the features, an evaluation function can often be a linear combination of them. See [25].

Clustering has several desirable properties. Within ascribed boundaries of a class, missing data presumably share characteristics of their neighbors, so clustering can be predictive (see Figs. 1a and 3). Once a class is formed, its determining data may be dismissed, so storage and computation may be economical. If statistics is employed, susceptibility to error may be low. Further, the structure used for clustering may concisely describe a *concept* that emerges automatically as information is compressed [25], cf [21]. As an example of all these characteristics, consider the leftmost rectangle of Fig. 4. Because of the imposed rectangular shape, the description of this "concept" is simple and easily stored: $(0 \le f_1 \le 4) \cap (0 \le f_2 \le 2)$. Moreover the associated utility $u = 0.2$ may have come from many data: out of perhaps 100 events observed in this rectangle, 20 of them may have been "successful". If one or a few observations were in error, the value of $u = 20/100 = 0.2$ would still be close to the "true" utility. As long as the utility does not vary too much within the rectangle's boundaries (a case of selective induction), u may relate to any enclosed point, observed or unobserved. As an added benefit, u may be more accurate because is has been measured over many "similar" events. This important coincidence of information compression, concept formation, and accuracy improvement may be called *mutual data support* [25].

Clustering is central in the family of *probabilistic learning systems* (PLS) [23, 24, 25], in which utility (probability of usefulness) is the dependent variable and feature values are independent variables. In PLS, the *utility* u is obtained by counting the number g of "good" states (events) contributing to solution of a problem or win in a game (or similar success measure in another domain), divided by the total number t of observed events. PLS1 clusters utility, thereby associating state descriptions of equivalent quality. The cluster or *region* R is a triple (r, u, ϵ), feature space volume r having utility u with error ϵ (this codes the interval $[u/\epsilon, u\epsilon]$). Because utility bears a smooth relationship to typical features, clustering may profitably be restricted as a partition of feature space, the *region set* (this is selective induction; see Figs. 1a and 3). This knowledge structure accumulates information incrementally, regions gradually being resolved into smaller units just adequate to express current knowledge (each region representing the worth u of cell r).[7] Characteristics of regions (e.g. rectangle size and shape) are determined by data, so a region set remains small, consistent with known information. Hence computation is inexpensive. It is also more autonomous than Samuel's signature table method [30] since no previous grouping of features is required (more of the inductive difficulty is handled by PLS). Other benefits include inherent insensitivity to noise (like [30]), and efficient convergence to locally optimal evaluation functions (a unique result -- see [23, 25]).

To accomplish this inference the PLS1 algorithm CLUSTER [23, 25] uses *splitting* [1]. A region R is refined when utility data are found to diverge within it. The criterion for splitting involves a dissimilarity (distance) measure d. If u_1 and u_2 are the two utilities for a tentative dichotomy, and ϵ_1 and ϵ_2 their errors, then $d = |\log u_1 - \log u_2| - \log(\epsilon_1 \epsilon_2)$. This distance is computed for all boundary insertions parallel to any feature space axis. If the largest d is positive, the

7. Splitting cells is a specialization rule. Generalization is also involved in PLS1; conditions or features are dropped. Reformulation takes place in PLS2; rectangular partitions are redrawn. See [25].

corresponding split is retained. The process repeated until additional refinement is unwarranted by the data (until $d \leq 0$). Notice that larger d means more assuredly dissimilar regions. If the values of d are summed for successive permanent splits of R, the result is a measure of reliability (credibility) of the complete clustering operation, the ***discrimination assurance D***. One factor affecting the error ϵ (and hence D) is sample size N (number of states); D improves as N increases.

As an example, consider Fig. 4, and suppose that the two leftmost rectangles resulted from the larger one $(0 \leq f_1 \leq 6) \cap (0 \leq f_2 \leq 2)$. Assume that the sample size N in each new rectangle was 100, and that the error ϵ is the logarithm of $1 + 1 / \sqrt{N}$. Then the total discrimination assurance $D = d = |\log[20/100] - \log[1/100]| - \log[(1.1)(1.1)] = -1.6 - 4.6 - 0.2 = 2.8$. If the sample size were 10, D would be 2.5 (still a high credibility).

While very useful for creating an evaluation function from typical features [23, 25], CLUSTER is inadequate for manipulating primitives, since their utility relationships are so disorderly (i.e. constructive induction is required -- see Fig. 1). Nevertheless, CLUSTER forms the basis for a more powerful algorithm in the feature formation system PLS0.

4. LAYERED KNOWLEDGE: STRUCTURES FOR COMPLEX CLUSTERING

Automatic construction of an evaluation function from features is difficult [23 ,25, 30], yet often a small part of the whole inductive problem. The main part, formation of features from primitives, should perhaps be accomplished in steps, since gradual information compression is easier [33]. In PLS0, feature creation is automated using three stages. Each uses a characteristic knowledge structure which progressively generalizes the level below; each stage imposes constraints, reduces complexity, extracts meaning, and increases regularity. The basis for generalization at each level is ***utility similarity*** in substructures of problem states (a relaxed form of invariance).

To illustrate the knowledge structures, their relationships, and their conformation into features, consider the example of Fig. 2. Here f is the number of instances of diagonal juxtapositions of two men in checkers (some degree of insurance against being jumped). We shall follow the hypothetical generation of f at each of the three knowledge levels. Linearly index the thirty-two elementary measures e_i as is standard in checkers. Suppose $e_i(B) = 0$ means board B has no piece currently occupying position i, while $e_i(B) = k$ indicates the presence of a man if $k=1$, or of a king if $k=2$ (with corresponding negative values if the piece is the opponent's). Then f would be the number of cases in which $e_i > 0$, $e_j > 0$, for "diagonally adjacent" i, j.

Creation of f requires the discovery that the condition ($e_i > 0$ and $e_j > 0$) is similarly good for these i, j. For example the utility of simultaneously positive values of e_{10} and e_{14} is about equal to the utility of the same condition on e_{18} and e_{19} (see Fig. 2). A prerequisite for mechanizing this inference is some language to express utility in the two dimensional ***subspace*** determined by e_i and e_j, separate from any other coordinate e_k. To permit this structuring, ***knowledge level 1*** incorporates projections of the n-dimensional primitive space (here $n=32$): a ***subspace specifier (SS)*** is a string $e_{i_1} e_{i_2} \ldots e_{i_s}$ of length $s \leq n$. Our illustrative example f can be expressed using a uniform combination of SS's ($s=2$). Paired with each SS $e_{i_1} e_{i_2} \ldots e_{i_s}$ is its ***utility descriptor (UD)***, a region set expressing utility relationships in this primitive *sub*space. An SS together with its UD is a ***primitive pattern.*** Level 1 of Fig. 2 shows two primitive patterns, one for the subspace specifier $e_{10}e_{14}$, and the other for the SS $e_{16}e_{19}$. Since the two subspaces represent identical patterns, one just a translation of the other, their UD's are somewhat alike.

The purpose of knowledge level 1 is to compress utility relationships into a concise form (utility descriptors which are primitive region sets) and to distinguish striking aspects of states (primitive subspace specifiers). Importantly, this compaction allows retention of just the meaningful information, so wasted space and time are minimized.

Knowledge level 2 facilitates search for regular utility patterns; elements exhibiting similar behavior are ***merged,*** eventually to produce a sensible feature. Regularities are recorded in a ***pattern class,*** a union of primitive patterns, i.e. a set of SS's with a ***common*** UD (Fig. 2). In our example, SS's would arise consisting of two adjacent coordinates, such as $e_{10}e_{14}$, $e_{16}e_{19}$ and $e_{22}e_{26}$. These would be placed in a distinctive pattern class because their individual UD's are similar and can be combined (details are given later). This category would indicate indistinguishable utility behavior of each component SS and also nonseparability of $e_{10}e_{14}$ into single coordinates e_{10} and e_{14}, etc.; i.e. the primitives are meaningful as ***pairs.***

The overall purpose of knowledge level 2 is to unify similar forms (SS's) as functionally equivalent substructures of events (problem states), i.e. to cluster SS's whose associated utility descriptors agree when superimposed. This correspondence of UD's strengthens knowledge about primitive-utility relationships, since more information is present in the union (mutual data support). These regularized pattern classes become prospective feature elements. Little information is wasted since only the strong patterns survive or even appear.

Knowledge level 3 is the most advanced structure for feature formation. A level 2 pattern class is augmented by a set of transformation operators which, when applied to primitive subspace specifiers in the class, reproduce extant members and fill in "missing" SS's. Operators are selected which give the "best" closure in this induction of the ***pattern group.*** The feature f requires translation and rotation (of 90°). Several pairs like $e_{10}e_{14}$, etc., might be needed for confidence in the general transformation, which induces a group of 56 primitive patterns.

Summarizing: At the lowest knowledge level, data are CLUSTERed to distinguish utility patterns in primitive subspaces (SS's); discriminating primitives are identified and their utility relationships are condensed as utility descriptors (UD's). At the second level, these results are consolidated into sets of corresponding primitive subspaces having mutually similar utility patterns (invariances). At the third knowledge level, individual primitive patterns of a given class are used to induce a complete group: a general rule is discovered for transforming one member of the class into another, and missing elements are inferred. Overall, this three level scheme develops a complex clustering structure (instead of the simpler partition or tree normally used [1]). As explained below, utility invariances help to generate these compound structures incrementally and efficiently.

5. REALISTIC CONSTRUCTIVE INDUCTION (FEATURE FORMATION)

This section outlines feature creation from the knowledge structures just described and considers lower computational complexity resulting from their restriction.

Features are produced from pattern classes (knowledge level 2) or from pattern groups (level 3). In either case, the associated utility descriptor (UD) for the category provides information about a formative feature f, defined thus: An event x to be assessed by f is first mapped into primitive space, then for each SS $e_{i_1} e_{i_2} \ldots e_{i_S}$ of the class, the utility given by the UD is summed. For our illustrative feature f, a pattern class might include $e_{10}e_{14}$, $e_{16}e_{19}$, and $e_{22}e_{26}$. Suppose (i) $e_{10}(x) = -1$ (enemy piece), $e_{14}(x) = 0$ (clear square), while (ii) $e_{16}(x) = e_{19}(x) = 1$ and (iii) $e_{22}(x) = e_{26}(x) = 1$ (all friendly pieces). Typical utilities given by the UD for (i) might be $u = 0.001$, but for (ii) and (iii), $u = 0.1$ (Fig. 2, level 2). Then f(x) is the sum of these, or .201. Since juxtaposed friendly men have higher utility, f becomes a measure of the "pair adjacency" for the entire checker board (if f is formed from a pattern group), or from parts of the board (if only a pattern class is known).[8] Thus, generation of a feature is straightforward. The main difficulty is *efficient* creation of reasonable knowledge structures preceding this ultimate step.

Even using exhaustive search, knowledge acquisition at level 3 is relatively inexpensive for current domains (although not generally [31]). Here we will focus on levels 1 and 2, which, combinatorially, are extremely complex. The number of possible subspace specifiers (SS's) is 2^{32} for checkers, and finding appropriate SS's is only one step in the induction. However the problem may be simplified.

In PLS0 the explicit creation of an SS begins with straightforward primitive CLUSTERing which determines various *ground* values and structures (D_G, UD_G, etc. -- see Section 3). Since not all variables differentiate utility in a practical (small) data set, this ground processing reduces the effective dimensionality (by about two thirds in trial experiments). The remaining primitives are *active* for current data. The fact that only some primitives are active accounts for the phenomenon of useful information extraction in ground CLUSTERing; even though features are confounded at the primitive level, partial patterns arise here in small data sets. Moreover, strong patterns appear first.

Simple utility-primitive relationships are reinforced through comparison with other formative patterns in events (i.e. with other components of the board). To discover meaningful patterns among these substructures (i.e. among subspaces of primitive space), certain arrangements A of variables from the active set S are considered. Let the size of S be n; each A is a *relation* [29] over S^k, where $k \le n$. For example, suppose $S = \{e_2, e_6, e_{10}, e_{14}, e_{15}\}$ is the active set. Then $n = 5$. For $k = 2$, one A is $\{(e_2,e_6), (e_{10},e_{14}), (e_{10},e_{15})\}$; (e_2,e_6) determines the SS e_2e_6, etc., and A defines a *superposition* of primitive subspaces, i.e. a *class* of SS's (here the class is $SS_A = \{e_2e_6, e_{10}e_{14}, e_{10}e_{15}\}$, a precursor of f).

8. A feature is tentative until enough support is found for its determinative pattern class. This support increases gradually as experience is gained (additional data result in more regions per UD and in more patterns per class). Once defined, features are independently assessed and selected by PLS1 [23, 25].

Given a superposition A of active variables, CLUSTER (Section 3) is now run with overlaid primitives treated as one. E.g., for SS_A above, e_2 and e_{10} would be identified. Because of this equivalence, the data counts (Section 3) are effectively increased (by a factor roughly equal to the number of superpositions -- it is patterns *within* states -- subobjects -- that are counted). Since larger counts imply lower errors, the discrimination assurance D is higher in cases where primitive patterns coincide; i.e. when utilities match up in the overlaid dimensions and merged UD's support each other (see Fig. 4). If, instead, utility behavior differs, the mutual support is weaker, and if misalignment is extreme, D is even lower than the ground value D_G . The simple example Fig. 5 shows three cases of superposition, in three tables. The leftmost is ground clustering (no overlaying) with subspace specifier $e_{10}e_{14}e_{16}$. In the first row, $(0,1,1)$ has a sample size t of only 5. In the third row, $(1,1,0)$ has $t = 8$. Note that in the former, $e_{10} = 0$, while in the latter, $e_{16} = 0$, and in no other row is either variable zero. The center table shows identification of e_{10} with e_{16}. Because of this identification and in the light of the above discussion, the t values from the first and third rows of the left table are summed in the first row of the center table, giving a sample size of 13 for e_{10} *or* e_{16} equal to zero (and $e_{14} = 1$). The other table entries are determined similarly. As it happens, the resulting divergence in utility (i.e. g/t -- see Section 3) is strong in the central table but practically nonexistent in the right one. Hence the right case has a low discrimination assurance D. However, since the numbers are larger there, D is higher in the center case than in the ground case, so the superposition of e_{10} with e_{16} is supported (these two variables are "similar" in this context). This is ***small sample similarity***, which has been verified in experiments. When various superpositions are examined, certain of them are found to stand out, having high D's in this way; these become parts of pattern classes. In summary, overlaid CLUSTERing extracts utility commonalities in components of the event (board description), and thereby discovers and strengthens patterns of meaningful structure.

e_{10}	e_{14}	e_{16}	g	t
0	1	1	3	5
1	0	1	0	6
1	1	0	3	8
		⋮		

$e_{10} \vee e_{16}$	e_{14}	g	t
0	1	6	13
1	0	0	12
1	1	6	13
		⋮	

$e_{10} \vee e_{14}$	e_{16}	g	t
0	1	3	11
1	0	6	16
1	1	3	11
	⋮	⋮	

Figure 5. **Three tables showing simplified data for ground clustering (left) and for superimposed clustering (center and right). The utility divergence (discrimination assurance D – see Section 3) depends on (i) dissimilar utilities g/t where t is the sample size and g is the number of successes, and (ii) the size of t, which affects error. Since t becomes larger with more superpositions of primitive variables (e.g. center and right), utility relationships may thus be strengthened. The central case is supported but the rightmost case is not. Hence e_{10} and e_{16} are similar but e_{10} and e_{14} are not. Superposition allows emergence of meaningful structure in events. See Section 5.**

Without guidance, examining superpositions is extremely complex; the number of trials is $O(2^{[n^n]})$, where n is the number of active primitives. However the value of n is reduced by screening inactive variables with ground CLUSTERing. More important, trials need not be exhaustive, but rather uniformly guided by general heuristics. Intrinsic bonds in primitive variables manifest in poorer D's when elements of the combination are unduly superimposed. This discovery of small sample *dissimilarity* -- e.g. of e_{10} and e_{14} above and in Fig. 5 -- disqualifies the pair of primitives as part of *any other* trial superposition, which means that only *pairs* need be overlaid in preliminary testing; the complexity of this is $O(n^2)$. After pairs of small sample *similar primitives* are formed (e.g. $A_1 = \{\{e_{10}\},\{e_{16}\}\}$, $A_2 = \{\{e_{14}\},\{e_{19}\}\}$, etc.), CLUSTERing is subsequently reapplied to merge pairs of similar primitives into larger sets (at this point overlays are still one-dimensional). Hence, similar patterns emerge and cluster into mutually *dissimilar* sets. Finally, general SS's are constructed by appropriate selection: for example if there are two dissimilar sets $S_1 = \{e_{10}, e_{16}, e_{22}\}$ and $S_2 = \{e_2, e_{14}, e_{19}, e_{26}, e_{30}\}$, one SS class would be $\{e_{10}e_{14}, e_{16}e_{19}, e_{22}e_{26}\}$, thus preserving primitive dissimilarity in distinct dimensions while identifying similar primitives and thereby creating structural pattern classes.

These algorithms have been programmed and tested with the fifteen puzzle.[9] The 15 primitives used were city block distances of *individual* tiles from their home positions. (This gives a 10% information compression advantage compared with the 80% of high level features -- see [25].) The first feature induced was the total distance score f_d . Experiments show that discrimination assurance D is a good measure for trial superpositions. When data were gathered using breadth-first search, f_d was created and no other pattern classes formed. In contrast, after f_d emerged to guide search, D values were less uniform in overlays, and other pattern classes arose.

To summarize: Feature formation is straightforward once appropriate knowledge structures have been created. In limiting the huge quantity of possible structures [32], PLS0 imposes few artificial constraints (cf [7]). Rather, the data themselves help to simplify search: structural patterns (primitive subspaces -- SS's) and goal-directed information (utility relationships -- UD's) are preferentially combined, consistent with intrinsic bonds among primitive components, using a credibility measure (discrimination assurance D) to assess candidate hypotheses (overlays). This process creates a *change of knowledge representation* (to meaningful event *components* expressed as classes and groups of patterns). PLS0 is designed for any cases in which components of events may be identified (whenever translations, rotations, etc., leave utility invariant). This applies to a large class of problems since goal-direction and invariance (similarity of event components) are usually inherent.

9. Other ideas have not yet been programmed. One involves comparison with predicted D's assuming perfect superposition (i.e. completely agreeing utility patterns). Still another heuristic involves guidance from pattern classes already forming: since PLS0 is incremental, current data may be compressed preferentially to match extant primitive patterns and classes -- by searching for supporting superpositions, e.g. those having the same SS length as established classes.

6. CONCLUSIONS

While uncompleted, PLS0 is a general, substantial, and promising learning system for realistic constructive induction. Its superior capability can be quantified (Section 2). When assessed according to criteria for constructive induction [7], PLS0 is appears solid. The three level knowledge *representation* is *adequate* for domains considered and *extensible* to others. The *rules of generalization* are powerful, suited to problems not previously explorable. Heuristics are general across domains. This stochastic scheme is *insensitive to error*. Finally, PLS0 is *efficient*; it effectively reduces computational complexity (e.g. from double exponential to polynomial in an important part of the problem -- see Section 5).

To improve efficiency, PLS0 feature creation uses a 'divide and conquer' scheme having three stages, each of which builds from elements verified at the level below. Structures are generated, assessed, and improved so that the generate-and-test cycle of inductive inference is meaningfully constrained. The basis for generalization is classification of utility patterns; invariance of utility relationships is used to extract regularities under transformation of primitives (basic descriptions). A new form of clustering allows the necessary structuring and *change of knowledge representation* resulting in a system which is both model- and data-driven. The progressive structuring relies on *mutual data support* during clustering of utilities and patterns; information is simultaneously regularized and reinforced. Mutual data support is important, it is the simultaneous occurrence of concept formation, noise management, accuracy improvement, and complexity reduction.

The computational complexity of the generalization problem may be reducible from intractably exponential to practically polynomial.

REFERENCES

(1) Anderberg, M.R., *Cluster Analysis for Applications*, Academic Press, 1973.

(2) Banerji, R.B., Some linguistic and statistical problems in pattern recognition, *Pattern Recognition 3,* (1971), 409-419.

(3) Banerji, R.B., Pattern recognition: Structural description languages, in Belzer, J. (Ed.), *Encyclopedia of Computer Science and Technology 12* (1978), 1-28.

(4) Berliner, H., On the construction of evaluation functions for large domains, *Proc. Sixth International Joint Conference on Artificial Intelligence,* 1979, 53-55.

(5) Bierre, P., The professor's challenge, *AI Magazine 5,* 4 (Winter 1985), 60-75.

(6) Dietterich, T.G., London, B., Clarkson, K., and Dromey, G., Learning and inductive inference, STAN-CS-82-913, Stanford University, also Chapter XIV of *The Handbook of Artificial Intelligence,* Cohen, P.R., and Feigenbaum, E.A. (Ed.), Kaufmann, 1982.

(7) Dietterich, T.G. and Michalski, R.S., A comparative review of selected methods for learning from examples, in [17] (1983), 41-81.

(8) Ernst, G.W. and Goldstein, M.M., Mechanical discovery of classes of problem-solving strategies, *J. ACM 29,* (1982) 1-33.

(9) Feldman, J.A. and Yakimovsky, Y., Decision theory and artificial intelligence: I. A semantics-based region analizer, *Artificial Intelligence 5* (1974) 349-371.

(10) Georgeff, M.P., Strategies in heuristic search, *Artificial Intelligence 20,* 4 (1983), 393-425.

(11) Langley, P., Bradshaw, G.L., and Simon, H.A., Rediscovering chemistry with the Bacon system, in [17] (1983), 307-329.

(12) Lenat, D.B. and Brown, J.S., Why AM and Eurisko appear to work, *Artificial Intelligence 23* (1984), 269-294.

(13) McCarthy, J., AI needs more emphasis on basic research, *AI Magazine 4,* 4 (Winter 1983), 5.

(14) Medin, D.L. and Smith, E.E., Concepts and concept formation, *Annual Review of Psychology 35,* 113-138, (1984).

(15) Medin, D.L., Wattenmaker, W.D., and Michalski, R.S., The problem of constraints in inductive learning (as yet unpublished manuscript), 1984.

(16) Michalski, R.S., A theory and methodology of inductive learning, *Artificial Intelligence 20,* 2 (1983), 111-161; reprinted in [17], 83-134.

(17) Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. (Ed.), *Machine Learning: An Artificial Intelligence Approach,* Tioga, 1983.

(18) Michalski, R.S. and Chilausky, R.L., Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis, *Int. J. Policy Analysis and Information Systems 4,* 2 (1980), 125-161.

(19) Michalski, R.S. and Stepp, R.E., Learning from observation: Conceptual clustering, in [17], 331-363.

(20) Popper, K.R., *Objective Knowledge: An Evolutionary Approach,* Clarendon Press.

(21) Quinlan, J.R., Learning efficient classification procedures and their application to chess end games, in [17] (1983), 463-482.

(22) Rendell, L.A., A method for automatic generation of heuristics for state-space problems, Dept of Computer Science CS-76-10, University of Waterloo, 1976.

(23) Rendell, L.A., A new basis for state-space learning systems and a successful implementation, *Artificial Intelligence 20,* 4 (1983), 369-392.

(24) Rendell, L.A., Toward a unified approach for conceptual knowledge acquisition, *AI Magazine 4,* 4 (Winter 1983), 19-27.

(25) Rendell, L.A., Conceptual knowledge acquisition in search, in Bolc, L. (ed.), *Knowledge Based Learning Systems,* Springer-Verlag, 1985 (to appear).

(26) Rendell, L.A. and Michalski, R.S., A model of practical induction (in preparation).

(27) Ritchie, G.D. and Hanna, F.K., AM: a case study in AI methodology, *Artificial Intelligence 23* (1984), 249-268.

(28) Rosch, E., Mervis, C.B., Gray, W.D., Johnson, D.M., and Boyes-Braem, P., Basic objects in natural categories, *Cognitive Psychology 8,* (1976), 382-439.

(29) Sahni, S., *Concepts in Discrete Mathematics,* Camelot, 1981.

(30) Samuel, A.L., Some studies in machine learning using the game of checkers II--recent progress, *IBM J. Res. and Develop. 11* (1967) 601-617.

(31) Tou, T.T. and Gonzalez, R.C., *Pattern Recognition Principles,* Addison-Wesley, 1974.

(32) Watanabe, S., *Knowing and Guessing: A Formal and Quantitative Study* Wiley, 1969.

(33) Watanabe, S., Pattern recognition as information compression, in Watanabe, S. (Ed.), *Frontiers of Pattern Recognition,* Academic Press, 1972, 561-567.

(34) Winston, P.H., *Artificial Intelligence* (second edition), Addison Wesley, 1984.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-85-1198	2.	3. Recipient's Accession No.
4. Title and Subtitle Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search			5. Report Date January 1985
			6.
7. Author(s)			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, IL			10. Project/Task/Work Unit No.
			11. Contract/Grant No. NSF DCR 84-06801
12. Sponsoring Organization Name and Address National Science Foundation Washington, DC			13. Type of Report & Period Covered
			14.

15. Supplementary Notes

16. Abstracts

This paper addresses a problem of practical inductive inference which is more difficult than any comparable work in AI. The subject of the present research is a hard problem of new terms, a task of realistic constructive induction: mechanized feature formation from data represented in elementary form. A feature, attribute, or property such as "piece advantage" in checkers is much more abstract than an elementary descriptor or primitive such as contents of a checkerboard square. Features have often been used in evaluation functions; primitives are usually too detailed for this. To create higher level features from primitives (i.e. to restructure data), a new form of clustering is used which involves layering of knowledge and invariance of utility (i.e. usefulness in attaining a goal). The scheme, which is both model- and data-driven, requires almost no background, domain-specific knowledge, but rather constructs it. The method achieves considerable generality with superior noise management and low computational complexity. Although the domains addressed are difficult, initial experimental results are encouraging.

17. Key Words and Document Analysis. 17a. Descriptors

Generalization Learning
Constructive Induction

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 17
	20. Security Class (This Page UNCLASSIFIED	22. Price