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

Larry Rendell
Department of Computer Science,
University of Illinois at Urbana-Champaign,
1304 West Springfield Avenue, Urbana, Illinois 61801

ABSTRACT

This paper addresses a problem of induction (generalization learning) 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.*

While the approach is quite general, the system is analyzed and tested in an environment of heuristic search where noise management and incremental learning are necessary. Here constructive induction becomes *feature formation* from data represented in elementary form. A high-level attribute or *feature* 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 abstract features from primitives (i.e. to restructure data descriptions), a new form of clustering is used which involves layering of knowledge and invariance of *utility relationships* related to data primitives and task goals. The scheme, which is both model- and data-driven, requires little 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]

I INTRODUCTION

A fundamental problem in AI is the automation of *inductive inference* [3,15,25].[2] Induction can be described as generalization from particular cases [2,0,15], as learning categorizations from examples or observation [15,18,10], as intelligent compression of massive data [25,30,31], or as discovering regular, coherent characterizations of events or *objects* [15,22 25].[3] Whatever the interpretation, induction begins either with objects or with partially formed groupings, and creates one or more *classes.* These meaningful sets are also called *categories* or *concepts.* 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, while induction of classes allows *prediction* of details [6,23,25,30].

Automated induction is crucial. For example, in an expert system, computer generalization means mechanized knowledge acquisition, which may reduce costs. Present expert systems are prone to unexpected error, whereas induction would increase reliability and obviate maintenance. Even in its primitive state, automated induction has outperformed knowledge engineering approaches [17].

A. Views of Inductive Difficulty

Unfortunately, induction is inherently difficult [1,6,30]. Noisy and sparsely distributed data offer little help in distinguishing hypothetical classes or concepts, and there are *many* hypotheses which are plausible, useful, sensible, or *credible.*

Inductive multiplicity can be understood in various ways. First, a credible hypothesis may be sought in a space of category *descriptions.* In this view of induction, 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) [2,6,30]. In this *search* formalization, exploration of plausible hypotheses must cope with exponential growth of their explicit representations.

The most abstract view of the inductive problem involves *classification* [30,31]. The number of ways of combining objects into classes is extreme. If a small 10X10 grid of bits encodes letters of the alphabet, the number of different classes is $27[^{2\,100}]$, and very few of these are credible. A view of induction which is intuitively close to the classification perspective is the *feature space* representation, illustrated in Fig. 1.

A third way of analyzing the difficulty of induction considers the quantitative problem as a qualitative one [15,18,23]. Class formation often requires *restructuring* of data: objects 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 [6,25]. Fig.I compares constructive induction with the simpler *selective* induction which needs no reorganization.[4]

1. This work was supported in part by an operating grant from the Natural Sciences and Engineering Research Council of Canada.

2. The problem of induction is also a basic study outside of AI. It has occupied scholars in philosophy [5], psychology [13], pattern recognition [2,3,0,20,31], and other fields [30].

3. Formally: Given a set S of individual objects, induction is the inference of a larger class or *hypothesis* T, such that S C T. Since T is a generalization it may not be true; associated with T is its *credibility,* the estimated veracity of the induction.

4. By selective induction we mean a case in which one or a few neighborhoods in feature space can easily be partitioned. Michalski's definition of this term [15] is more precise, but it excludes some cases we would want to call simple, such as classes discriminated by linear decision functions.

To structure data, an expressive language such as predicate logic is required, but there are tradeoffs between expressiveness and efficiency (at least with current models).[8] Greater expressive power causes a worse combinatorial explosion in search.

B. Research Directions

One approach to the combinatorial problem of induction is to limit hypotheses by imposing constraints on their expression [15,25,32]. 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 [15,18], and "invariance" under transformation of task elements [8]. A relaxed form of invariance has long been used in statistics and pattern recognition, viz. "similarity" in cluster analysis [1]. The author's scheme involves a special kind of similarity constraint based on the domain environment. The precise form of this similarity has to do with success or "utility" in the performance of some task [21,23,24,25].[6] Overall, constraining search for abstract knowledge has seen only limited success in terms of efficiency, effectiveness, and extensibility [7,15,32].

Generalization algorithms have been designed either for selective induction [6,7,15,19,21,28] or for quite simple constructive induction [10,15]. 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 [6,7,15,25]). Furthermore, AI research often ignores earlier germane results, including [2,30]. 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 [11,26].

C. Quantitative Analysis

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 [5,12,15,26] for similar sentiments.) In keeping with this goal, the author has begun to pursue a means for comparing inductive tasks and systems. Based on [30,31], this involves a quantification of *inductive difficulty,* both for task domains and for learning systems. This attempt began in [22,23,25].

5. See [23] for a discussion of representation languages. Important issues include equivalences of superficially different representations, and the distinction between languages appropriate for *expression* of concepts and languages useful for concept *formation.* The current paper and [25] examine structures for *mediation* between data and concepts.

6. Task-related *utility* is used as the criterion for *clustering* in the original *probabilistic learning system PLSL* This system has produced unique results such as convergence to optimal heuristics [21,24,25]. PLS1 can handle noisy environments and incremental learning. The system is efficient.

The idea is simple: since induction creates a class T from a set of objects S, the difficulty of the generalisation task depends on the nature of the information compression from cases S to concept T.[7] 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 [22,25]). Secondly, if the attributes describing S do not support straightforward selective induction (as in Fig. Ia), but must instead be redefined by constructive induction (Fig. Ib), then inductive difficulty depends on the kind of reconstruction required. When examined in the light of these measures, many systems (such as [10,17,21]) perform only a moderate amount of induction. In these systems, the total number of possible generalizations is confined to be relatively small from the outset.

D. Substantial Induction

In contrast to these simplified approaches, the current research attempts a very difficult task. Not only is the required amount of 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.

The method is related to a successful system for selective induction *PLSL* [6] One aspect common to PLS1 and the new system PLSO is the observation of a goal-oriented measure, the *utility.* The utility *divergence,* a function of "features" of objects, and also based on data patterns, is the criterion for induction. When PLS1 uses utility for categorization (conceptual clustering), various efficiencies and other advantages ensue. The scheme has been *incremental* and *insensitive to noise* since its inception [20,21,24].

Retaining these valuable and unusual characteristics, PLSO 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 will be elaborated.

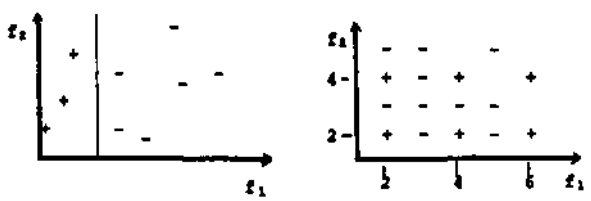


Figure 1. Degrees of inductive difficulty. A simple case of clustering successful objects (left) requires only that a few boundaries be inserted. This is *selective induction.* Complex cases may involve varying degrees of heterogeneity, which may be more concisely described using *concepts.* An example is the class defined by variables f_1 and f_2 both being even (right). Formation of concepts is *constructive induction.*

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

II AN EXAMPLE IN HEURISTIC SEARCH

This section illustrates some of the above ideas and relates them to a hard problem of constructive induction. We introduce task utility for inductive guidance, and layered abstraction of task knowledge. In later sections, these terms and those in the diagrams will be more carefully defined.

A. Goal-Directed Information Compression

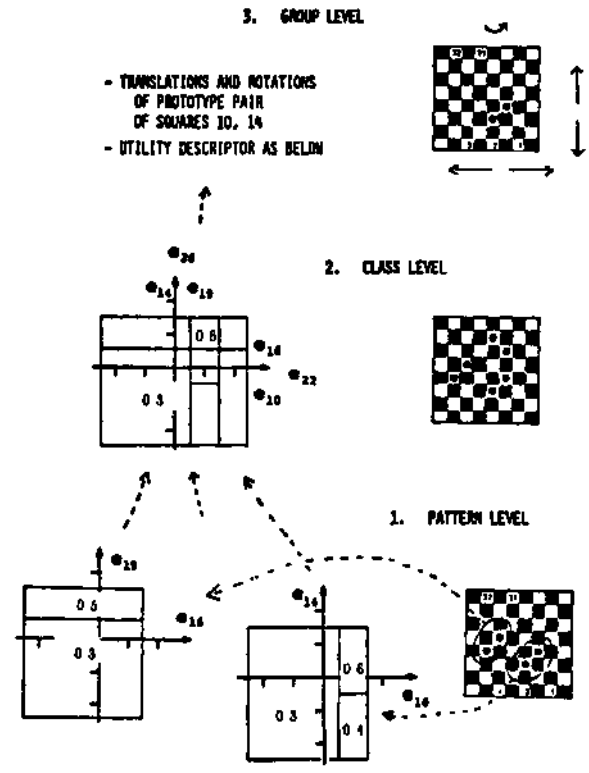
An example of a product of induction is the concept of pair *adjacency:* two diagonally juxtaposed, friendly men in checkers (Figs. 2 & 3). In the description of such a concept, certain spatial patterns are essential (adjacency) while other aspects are immaterial (location of the pattern and presence 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. Utility is the most abstract concept desired.

Here we may express utility using an *evaluation function H.* Objects assessed by H are *states* or board configurations B. To evaluate board B, H combines pair adjacency, piece advantage, and other *features.* In terms of generalization, states are elementary object descriptions, the utility value given by H is the most concise description, and feature values are intermediate between objects and their utilities.

B. Layers of Abstraction

As it happens, a resolution of roughly 100 distinct *utility classes* is sufficient to differentiate among moves in the game of checkers (see the discussions in [22,25]). Fig. 4 reflects this; here utility values lie in the interval [0,1] and are given to two decimal places. In contrast, an object (board configuration or state) is a vector of 32 *primitives* indicating the contents of each permissible square (black king, black man, vacant, red man, red 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

Figure 3. The three level information structure of FLSO. 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 detailed description of board configurations in terms of 32 *primitives* e, giving the contents of individual squares. (See Fig. 2 for detail.) When subspaces such as $e_{10}e_{14}$ and $e_{18}e_{10}$ are examined, and utilities are CLUSTERed after superposition of subspaces, important structural patterns emerge having common *utility descriptors* (UD's). A *pattern class* results (level 2). Eventually a class may be generalized using a group of transformations (level 3).



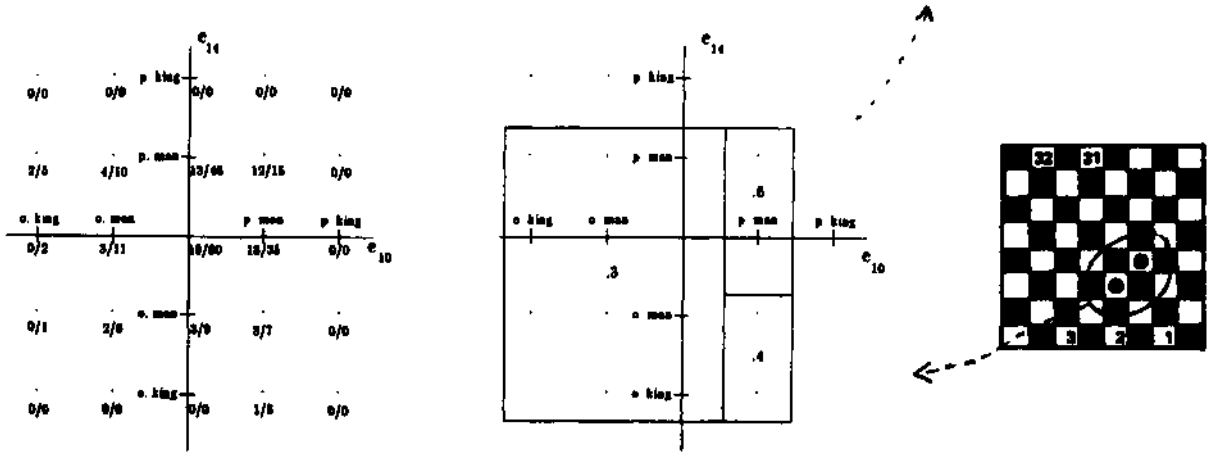


Figure 2. Clustering small samples of primitive subobjects. Shown is a projection of the 32 dimensional space of subobjects (checkerboard squares). Here the tenth and fourteenth squares are sampled. The possible values for each square are blank or 0 (origin), our player's man or 1 (p.man), opponent's king or -2 (o.king), etc., for a total of 5 X 5 = 25 coordinates in this two-dimensional subspace. The fractions beside each point (extreme left) indicate the proportion of winning states. The clusters (extreme right) compress this utility information (omitting points with no observations). This rightmost view becomes the least abstract level of Fig. 3.

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.

Utility classes are much easier to describe in terms of predefined features. As Fig. 4 illustrates, the number of classes at the feature level is very roughly 10^6 - this is the size of a typical feature space [25 ,28]. Utility bears a smooth relationship to features, so the induction required is merely selective (Fig. Ia), and only a few simple descriptions are needed (Fig. 5).[9] In contrast, raw data descriptions in the form of primitive vectors are more detailed (having 10^{20} values as opposed to 10^8). Moreover, utility-primitive relationships are discontinuous, so the much more difficult constructive induction is needed (see Figs. 1 and 3).

C. Automated Feature Construction

This paper begins to explore a method for discovery of features from primitives, a problem hardly addressed previously (but see [8,9,19]). 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.

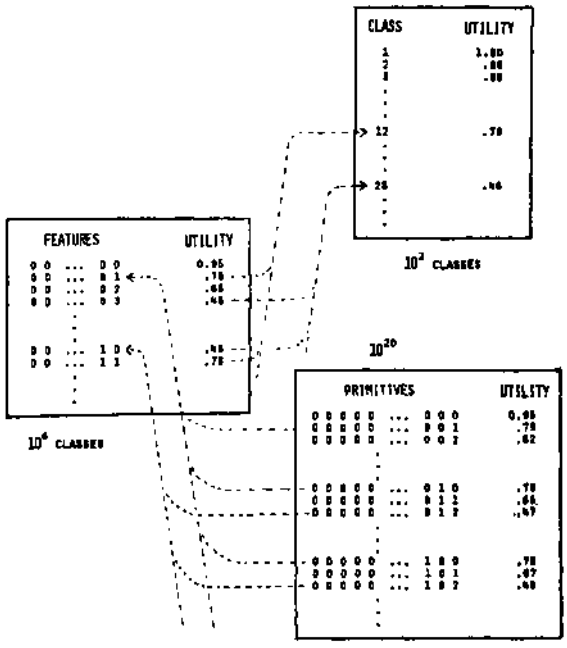


Figure 4. Levels of abstraction in search. Elementary data in the form of primitive descriptions represent fully detailed knowledge, but are massive and infeasible to gather. At the other extreme, maximal compression expresses utility classes concisely. Intermediate representations facilitate expression: e.g. features discriminate utility quite smoothly. In contrast, primitive measurements determine utility very irregularly.

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

How can utility-primitive relationships be captured and generalized? Induction is infeasible without some guidance from regularities, assumed or else discovered [32]. One technique, curve fitting, is often inadequate even in selective induction when features are the starting point [6,25,28]. A more flexible approach is to record information in feature space cells, as in Fig. 5 [21,28]. An important tool for inducing this knowledge is the method of *cluttering.*

ffI CLUSTERING, A TOOL FOR INDUCTION

Clustering classifies data so that events or *objects* are *similar* within any class, but dissimilar across classes [1]. This technique has been used in successful learning systems by Michalski [15,17,18], and earlier by the author [20,21,25], who originated a form of *conceptual clustering.* This kind of clustering takes into account not only feature values, but also forms of concepts (e.g. feature space rectangles) and aspects of the environment (e.g. observed *utility).* Michalski has emphasized concept constraints, and the author has stressed relationships between task domain and inductive algorithms. These are dual aspects: model- and data-drivenness.

In this section we describe general characteristics of clustering, and then examine PLSI's algorithm for selective induction as a preliminary for PLSO's more powerful constructive induction.

A. Mutual Data Support

Clustering has several desirable properties. Within ascribed boundaries of a cluster, missing data presumably share characteristics of their neighbors, so the process can be predictive (see Figs, la and 5). Once a class is formed, its determining data may be dismissed, so storage and computation can be economical. If statistics is employed, susceptibility to error may be low and credibility may improve as a result of clustering the data. Further, the structure used for clustering may concisely describe a *concept* that emerges automatically as information is compressed [15,25].

As an example of these characteristics, consider the leftmost rectangle of Fig. 5. 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 N = 100 objects observed in this rectangle, g = 20 of them may have been "successful". If a few observations were in error, the value of $u = g/N = 20/100 = 0.2$ would still be close to the "true" utility (here a probability). As

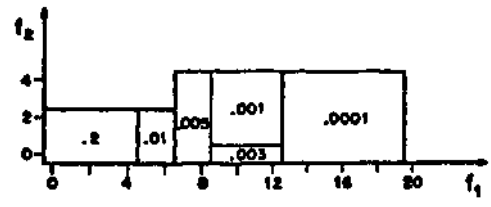


Figure 5. A region set is a partition of feature space with associated information. Shown are classes (rectangles) r and their values *(utilities)* u for some task. A *region* can simply be the pair (r,u). In PLS1 a feature space region set is used as an evaluation function. In PLSO a primitive space region set is used to create features.

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 it has been measured over many "similar" objects. We call this important coincidence of information compression, concept formation, and accuracy improvement *mutual data support* [25].

B. Goal-Directed Cluttering Using Utility

Mutual data support arises in the statistical technique of clustering. In [20, ,21,25], the author introduced a special kind of clustering which employs not just attributes or features of an object, but also the quality or *utility* of that object in the task environment.[10] This clustering criterion is central in the family of *probabilistic learning systems* (PLS) [22,24,25].[11] In PLS1, the utility u is a probability: the number g of "good" objects contributing to task success, divided by the total number N of observed objects: the *utility*

$$u = g / N.$$

E.g., the object might be a state in a problem or game, and success might mean appearing in a solution or win.

C. Regions as Clusters of Similar Utility

PLS1 clusters utility, thereby associating state descriptions of common quality. The cluster or *region* R is a triple (r, u, ϵ), feature space volume r having utility u with error factor ϵ (this codes the interval $[u/\epsilon, u\epsilon]$). Because utility bears a smooth relationship to typical features, clustering may profitably be constrained as a partition of feature space, the *region set* (see Fig. 5). The region set is suitable for selective induction (Fig. 1a); it is a compressed representation of the utility surface in this *augmented* feature space.

Characteristics of regions (e.g. rectangle size and shape) are consistent with data encountered: a region set remains small enough to economise resources but large enough to express important knowledge. To accomplish this, PLS1 employs a *utility revision* rule for correcting values of u, a *specialization* rule for refining cells r, and *generalization* and *reorganization* operators for otherwise modifying regions.

Various unusual or unique advantages result. Computation is inexpensive, and more autonomous than Samuel's signature table method [28]. The algorithms *automatically* produce an effect similar to a criterion using "similarity" and "fit" (see [21,24,25] and compare [15,18]). Results include efficient discovery of optimal heuristics.

D. Region Refinement and Credibility

Here we consider in detail the PLS1 refinement operator for *splitting* regions: the algorithm CLUSTER[21,25]. A region R is dichotomised when utility data within it are found to diverge. 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 $d = |\log u_1 - \log u_2| - \log(\epsilon_1 \epsilon_2)$. The dissimilarity d is computed for all boundary insertions parallel to any feature space axis. If the largest d is positive, the corresponding split is retained. The process is 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*. D increases with the quotient u_1/u_2, and inversely with the error ϵ. One factor affecting the error ϵ is sample size N (number of observed objects): as N increases, ϵ decreases and D improves.

As an example, consider Fig. 5, and suppose that the two leftmost rectangles resulted from $(0 \leq f_1 \leq 6) \cap (0 \leq f_2 \leq 2)$. Assume that the sample size N in each new rectangle is 100, and that the error ϵ is $\log [1 + 1/\sqrt{N}]$. Then the total discrimination assurance D is just the dissimilarity $d = |\log[20/100] - \log[1/100]| - \log[(1.1)(1.1)] = -1.6 - 4.6 - 0.2 = 2.8$. If the sample size were 10, D would be 2.5, still a high credibility. We shall consider D again in Section V.

While very useful for creating an evaluation function from typical features [21,25], 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.

IV STRUCTURES FOR COMPLEX CLUSTERING

Automatic construction of an evaluation function from features is difficult [22,25,28], yet just a small part of the whole inductive problem. The main problem, formation of features from primitives, should perhaps be subdivided, since gradual information compression is easier [31].

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 *similarity of utility relationships* in subobjects (i.e. invariance of utility surfaces in augmented primitive space).

To illustrate PLS0 knowledge structures, their relationships, and their conformation into features, consider the example of Figs. 2 & 3. Here the subobjects are squares of the checker board, and the illustrative concept C to be learned is *pair adjacency*, i.e. the diagonal juxtaposition of two men (some degree of insurance against being jumped). The feature f to be created is the number of instances of C.

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). The feature f is 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 adjacent i, j

10. See Anderberg [1, pp. 194ff,] for discussion of "external criteria" in clustering.

11. The original *PLS1* has been extended to become a more stable, powerful and efficient learning system ***PLS2*** **[24, 25],** but here we refer to both as PLSl.

pairs. 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_{16} and e_{19} (see Fig. 3). 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.

A. Knowledge Level 1: Subobject Relationships

To permit this structuring, *knowledge level 1* uses 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, all with $s=2$. For example, one member of the *adjacency* concept is e_{10} e_{14} (Fig. 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 subspace. An SS together with its UD is a *primitive pattern.* Fig. 2 and level 1 of Fig. 3 show two primitive patterns, one for the subspace specifier $e_{10}e_{14}$, and the other for the SS $e_{16}e_{19}$.

A primitive pattern is a compressed representation of the function u(e), where u is the utility, and e is the full primitive vector. The compression is of two types: projection of primitive space into subspaces, and approximation of u as a step function. The subspaces are indicated as subspace specifiers (SS's), and the step function is expressed as a utility descriptor (a UD is a primitive region set).

The purpose of knowledge level 1 is to compress utility relationships concise forms distinguishing striking aspects of objects. Consider Fig.2 again. The SS's $e_{10}e_{14}$ and $e_{16}e_{10}$ are both meaningful as subobject *pairs.* These two SS's represent similar structures, one just a translation of the other, so their UD's are somewhat alike. Primitive patterns may stand out like this, based on small, biased domain samples. We shall return to small sample effects in Section V.

B. Level 2: Pattern Classes of Similar Structures

Knowledge level 2 facilitates search for similar subobject structures: those exhibiting similar utility 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. 3). 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 pair*.

The overall purpose of knowledge level 2 is to unify similar primitive structures (e.g. patterns of checkerboard squares) as functionally equivalent subobjects, i.e. to cluster SS's whose associated utility descriptors agree when superimposed. Correspondence of UD's strengthens knowledge about utility-primitive 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.

C. Knowledge Level 3: Pattern Groups

Knowledge level S is the most advanced structure for feature formation. Here we discuss it only briefly.

A level 2 pattern class is augmented by a set of transformation operators which, when applied to 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.

An alternative formalization for knowledge level 3 is grammatical inference [9]. The above example would become a single production rule.

D. Summary of Layered Knowledge

At the lowest knowledge level, data are CLUSTERed to distinguish prominent utility surfaces in primitive subspaces. Discriminating combinations of primitives (SS's) 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 surfaces (UD's). As a consequence of matching and coalescing utility relationships, important structure in domain subobjects is identified and extracted. This structure creation emerges from *clustering utility surfaces.*

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. As explained below, utility invariances help to generate these compound structures incrementally and efficiently.

V REALISTIC CONSTRUCTIVE INDUCTION

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

A. Feature Formation

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 object 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. 3, 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).[12] Thus, generation of a feature is straightforward. The main difficulty is *efficient* creation of reasonable knowledge structures preceding this ultimate step.

B. Restricted Use of Primitives

Even using exhaustive search, knowledge acquisition at level 3 is relatively inexpensive for current domains (although not generally [28]). 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 PLSO, the explicit creation of an SS may begin with straightforward primitive CLUSTERing which determines various *ground* values and structures (D_G, UD_G, etc. - see Section III). 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 abstract features are confounded at the primitive level, structures do arise for small data sets. Moreover, strong patterns appear first.

Utility-primitive relationships are reinforced through comparison with other formative subobject structures (i.e. with similar components of the board). To discover meaningful patterns among these subobjects (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* [27] over S^k, where $k<n$. 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 $\{(e_2, e_6), (e_{10}, e_{14}), (e_{10}, e_{15})\}$. The pair ($e_2$,$e_6$) 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).

C. Mutual Data Support for Structuring

Given a superposition A of active variables, CLUSTER (Section III) is now run with overlaid primitives treated as one. E.g., for SS_A, e_2 and e_{10} would be identified. Because of this equivalence, the sample sizes N (Section III) are effectively increased (by a factor roughly equal to the number of superpositions - it is patterns *within* states - subobjects - that are counted). Since larger N implies 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 Figs. 2& 3). 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 of Fig. 6 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, $(e_{10}, e_{14}, e_{16}) = (0,1,1)$ indicates that the tenth square is blank, and that the fourteenth and sixteenth squares both contain a friendly man. This structure has a sample size N of only 5. In the third row of this leftmost table, (1,1,0) has $N = 8$. Note that in row one, $e_{10} = 0$, while in row three, $e_{16} = 0$. In no other row is either variable zero.

The center table shows identification of e_{10} with e_{16}. In the first row of this table, N is the sum of the N values from the first and third rows of the leftmost table. This gives a sample size of 13 for e_{10} or e_{16} equal to zero and $e_{14} = 1$. The other table entries are computed similarly.

Recall from Section III that utility $u = g/N$, and that D is the sum of utility dissimilarities when rectangles are split. Comparing the central and rightmost tables, we see that the discrimination assurance D should be substantial in the central table but likely zero in the rightmost one. Thus the rightmost overlay is rejected. Since the center case has utility values similar to the ground case, but the quantities are larger in the center overlay, D is higher there. Hence the superposition of e_{10} with e_{16} is supported (these two variables are "similar").

This is *small sample similarity,* which has been verified in experiments. When various superpositions are examined, certain of them are found to stand out, i.e. to have high D's; these become components of pattern classes. In summary, overlaid CLUSTERing extracts utility commonalities in components of the object (board description), and thereby discovers and strengthens patterns of meaningful structure.

D. Constrained, yet General Constructive Induction

Without some guidance, examining superpositions is extremely complex: the number of trials is $O(2^{[n^2]})$, where n is the number of active primitives. The value of n may be reduced by screening inactive variables with ground CLUSTERing (Subsection B), but more important, trials need not be exhaustive. They can be 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* is illustrated above and in Fig. 6, where e_{10} and e_{14} are distinguished. Such a discovery disqualifies the superposition $\{\{e_{10}\},\{e_{14}\}\}$ as part of *any other* trial, which means that only pairs need be overlaid in preliminary testing. The complexity of this is $0(n^2)$.

e_{10}	e_{14}	e_{16}	g	N
0	1	1	3	5
1	0	1	1	4
1	1	0	3	8

e_{10} v e_{16}	e_{14}	g	N
0	1	6	13
1	0	2	12
1	1	5	13

e_{14} v e_{16}	e_{10}	g	N
0	1	4	11
1	0	6	16
1	1	4	11

Figure 6. Three tables showing simplified data for ground clustering (left) and for superimposed clustering (center and right). The discrimination assurance D (Section III) depends on (i) dissimilar utilities g/N where N is the sample size and g is the number of successes, and (ii) the value of N, which affects error. Since N increases with superposition of primitive variables (e.g. center and right), utility relationships may 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 promotes discovery of meaningful structure in objects.

12. 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 [21,25].

After pairs of small sample similar primitives are formed (e.g. $A_1 = \{\{e_{10}\}, \{e_{18}\}\}$, $A_2 = \{\{e_{14}\}, \{e_{19}\}\}$, etc.), CLUSTERing is subsequently reapplied to classify pairs of similar primitives into larger sets (at this point, overlays are still one-dimensional). In this way, 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.[13] The 15 primitives used were city block distances of individual tiles from their home positions. (This gives a \0% 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 began to arise.

E. PLSO Outlook and Summary

One question about PLSO methodology is its generality. We can gain some insight by considering, for example, Samuel's features for checkers. Several of these, such as piece advantage, guard, etc., would present no problem. Others, though, such as mobility, would require more sophisticated transformations (or additional knowledge levels) for efficient induction. Not only games and puzzles, but also various real-world applications seem either to conform to the general PLSO approach as it stands, or else to possible extensions of the system.

To summarize: Feature formation is straightforward once appropriate knowledge structures have been created. In limiting the huge quantity of possible structures [30], PLSO imposes few arbitrary constraints (compare [7]), but rather the data help to simplify search Structural patterns (primitive subspaces — SS's) and goal-directed information (utility relationships - UD's) are meaningfully combined using a credibility measure (discrimination assurance D) to assess candidate hypotheses (overlays).

This process creates a change of knowledge representation — to meaningful object components expressed as classes and groups of structures together with information about their utility. Intrinsic bonds among primitive components become explicit. PLSO is designed for any cases in which components of objects 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 object components) are prevalent.

13. Other ideas have not yet been programmed. One involves comparison with predicted D's assuming perfect superposition (i.e. full aligned utility surfaces). Still another heuristic involves guidance from pattern classes already forming: since PLSO 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.

VI CONCLUSIONS

While still formative, PLSO is a general, substantial, and promising learning system for realistic constructive induction. Its superior capability can be quantified (Section II). When assessed according to criteria for constructive induction [7], PLSO 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, PLSO is efficient; it effectively reduces computational complexity (e.g. from double exponential to polynomial in an important part of the problem — see Section V).

To improve efficiency, PLSO 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. Each of the three stages is both model- and data-driven: general heuristics speed formation of hypothetical concepts, and task utility determines the more credible ones.

The basis for this constructive induction is clustered utility surfaces. Invariance of utility relationships in primitive description spaces leads to a new form of clustering which creates new knowledge structures. Progressive structuring relies on mutual data support during clustering of utility surfaces: 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 S, (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) Bierre, P., The professor's challenge, AI Magazine 5, 4 (Winter 1985), 60-75.

(5) Christensen, R., Foundations of Inductive Reasoning, Entropy, Ltd., 1964.

(6) Dietterich, TG„ London, B., Clarkson, K., and Dromey, G., Learning and inductive inference, STAN-CS-82-913, Stanford University, also Ch.XIV of The Handbook of Artificial Intelligence, Cohen & Feigenbaum (Ed.), Kaufmann, 1982.

(7) Dietterich, T.G. and Michalski, R.S., A comparative review of selected methods for learning from examples, in [16] (1983), 41-81.

(8) Ernst, G.W. and Goldstein, M.M., Mechanical discovery of classes of problem-solving strategies, J. ACM S9, (1982) 1-33.

(9) Fu, K.S., Syntactic Pattern Recognition and Applications, Prentice-Hall, 1982.

(10) Langley, P., Bradshaw, G.L., and Simon, H.A., Rediscovering chemistry with the Bacon system, in [16) (1983), 307-329.

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

(12) McCarthy, J., AI needs more emphasb on basic research, AI Magazine 4, 4 (Winter 1983), 6.

(13) Med in, DX. and Smith, E.E., Concepts and concept formation, Annual Review of Psychology 35, (1984), 113-138.

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

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

(16) Mkhalski, R.S., CarboneU, J.G., and Mitchell, T.M. (Ed.), Machine Learning: An Artificial Intelligence Approach, Tioga, 1983.

(17) 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.

(18) Michalski, R.S. and Stepp, R.E., Learning from observation: Conceptual clustering, in |16), 331-363.

(19) Quinlan, J.R., Learning efficient classification procedures and their application to chess end games, in |16| (1983), 463-482.

(20) 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.

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

(22) Rendell, LA., Toward a unified approach for conceptual knowledge acquisition, AI Magazine 4, 4 (Winter 1983), 19-27.

(23) Rendell, L.A., Utility patterns as criteria for efficient generalisation learning, Proc. 1085 Conference on Intelligent Systems and Machines, (also Univ. of Illinois Report No. UIUCDCS-R-85-1206), 1985.

(24) Rendell, L.A., Genetic plans and the probabilistic learning system: Synthesis and results, Univ. of Illinois Report No. UIUCDCS-R-85-1209 (submitted for publication), 1985.

(25) Rendell, L.A., Conceptual knowledge acquisition in search, University of Guelph Report CIS-83-15, Dept. of Computing and Information Science, Quelph, Ontario, Canada, 1983.

(26) Ritchie, G.D. and Hanna, F.K., AM: a case study in AI methodology, Artificial Intelligence 25(1984), 249-268.

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

(28) Samuel, AX., Some studies in machine learning using the game of checkers II-recent progress, IBM J. Ret. and Develop. II (1967) 601-617.

(29) Ton, T.T. and Gonsales, R.C., Pattern Recognition Principies, Addison-Wesley, 1974.

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

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

(32) Winston, P.H., Artificial Intelligence, Addison Wesley, 1984.

ACKNOWLEDGEMENTS

I would like to thank Steve Chien, Chris Matheus, Sheldon Nichol, and Raj Seshu for discussions of ideas in this paper, and for useful criticisms of the text. I also appreciate the helpful comments from IJCAI reviewers.

APPENDIX. GLOSSARY OF TERMS

Clustering. Cluster analysis has long been used as a tool for induction in statistics and pattern recognition [1,29]. Similar improvements to the basic techniques have been invented independently by the author and by Michabki (see Section III). This paper presents still another extension of clustering, one suitable for constructive induction.

Feature. A feature is an attribute or property of an object. Features are usually quite abstract (e.g. "mobility"). The utility (see below) varies smoothly with a feature. Compare "primitive*'.

Induction. Induction, hypothesis formation, or generalization learning is an important means for knowledge acquisition. Information is actually created [30]. Induction forms data into classes or categories in order to predict future events efficiently and effectively. Selective induction is relatively simple: it forms neighborhoods of feature space clusters. Constructive induction is much more difficult, requiring creation of concepts. See Sections I, III and IV.

Mediating structures. Successful systems tend to incorporate knowledge structures which mediate objects and concepts during inductive processing. These structures include means to record growing assurance of tentative hypotheses. See [25].

Mutual data support. This is a term coined by the author to express a subtle combination of phenomena in the inductive process. Several causes and effects are interwoven: noise management, accuracy improvement, efficient processing, and concept formation (Sections III.A and V.C).

Object. Objects are any data to be induced into categories. Components of objects (subobjects) may be interrelated, and regularities may be discovered through constructive induction. Relationships usually depend on some task domain. See "utility*.

PLS. The probabilistic learning system can learn what are sometimes called "single concepts" [6], but PLS is capable of much more difficult tasks, involving noise management, incremental learning, and normalization of biased data. PLS1 uniquely discovered locally optimal heuristics in search [21,25]; PLS2 is an effective and efficient extension using the genetic paradigm [24]; and PLS0 is the system for constructive induction examined in this paper. PLS manipulates regions in augmented feature or primitive space, using various inductive operations [23].

Primitive. A primitive is a detailed, low-level property of an object or subobject. The utility varies irregularly or discontinuously with a primitive. Compare "feature".

Region. The region is PLS's basic structure for induction (clustering). It is a compressed representation of a utility surface in augmented feature space. See Section III.C.

Utility. This is any measure of the usefulness of an object in the performance of some task. The utility u provides a link between the task domain and generalisation algorithms, u can be a probability: the number g of "good" objects contributing to task success, divided by the total number N of observed objects; i.e. $u = g/N$. Utility is related to an evidential criterion for induction. See Section HI.