TOWARDS DOMAIN-INDEPENDENT MACHINE INTELLIGENCE

Robert Levinson

UCSC-CRL-93-39
November 4, 1993

Department of Computer and Information Sciences
University of California Santa Cruz
Santa Cruz, CA 95064 U.S.A
Phone: 408-459-2097
FAX: 408-459-4829
E-mail: levinson@cis.ucsc.edu

ABSTRACT

Adaptive predictive search (APS), is a learning system framework, which given little initial domain knowledge, increases its decision-making abilities in complex problems domains. In this paper we give an entirely domain-independent version of APS that we are implementing in the PEIRCE conceptual graphs workbench. By using conceptual graphs as the "base language" a learning system is capable of refining its own pattern language for evaluating states in the given domain that it finds itself in. In addition to generalizing APS to be domain-independent and CG-based we describe fundamental principles for the development of AI systems based on the structured pattern approach of APS. It is hoped that this effort will lead the way to a more principled, and well-founded approach to the problems of mechanizing machine intelligence.

The APS framework has been applied to a number of complex problem domains (including chess, othello, pente and image alignment) where the combinatorics of the state space is large and the learning process only receives reinforcement at the end of each search. The unique features of the APS framework are its pattern-weight representation of control knowledge and its integration of several learning techniques including temporal difference learning, simulated annealing, and genetic algorithms. Morph, an APS chess sytem, is now being translated into PEIRCE.

Keywords: Conceptual graphs, semantic distance, similarity metric, machine learning, reinforcement learning, pattern construction, pattern recognition, adaptation, domain independence, nearest neighbor, experience-based learning.

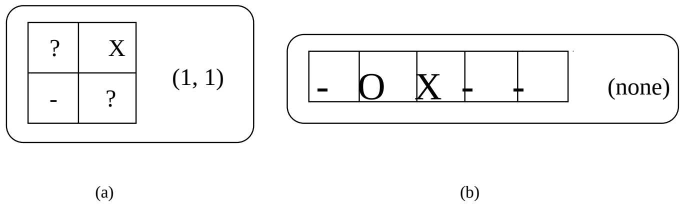


Figure 0.1: Patterns for Othello.

An 'X' represents our piece, 'O' the opponent's, '-' no piece, and '?' any piece. Note that the pattern in (a) is bound to the specific board location (1, 1) while (b) represents a pattern that may appear anywhere on the board.

0.1 Introduction

It has been the goal of AI researchers for many years now to build a successful "tabula rasa" system. That is, a system that starting from little apriori knowledge, with experience is able to cope with its environment. In this paper we discuss our own attempt at building a tabula rasa system. APS (Adaptive-Predictive Search) is a general learning system that improves with experience in what we term "complex search domains": These are problems that have a formulation as a state space search. Further, reinforcement is only provided occasionally, and for many problems only at the end of a given search. Finally, the cardinality of the state space must be sufficiently large so that attempting to store all states is impractical.

The first major aspect of an APS system is the compilation of search knowledge in the form of pattern-weight pairs (pws). Patterns are predicates that represent features of states, and weights indicate their significance with respect to expected reinforcement. Secondly, since the APS system resides within the experience-based learning framework, it must possess facilities for creating and removing search knowledge (pws). Knowledge is maintained to maximize the system's performance given space and time constraints. The APS model uses a variety of techniques for inserting and deleting patterns. Finally, a combination of several learning techniques incrementally assign appropriate weights to the patterns in the database. The specific insertion, deletion, and learning techniques will be described in the next section.

Since APS adheres to the experience-based learning framework, it can be applied to new domains without requiring the programmer to be an expert in the domain. In fact, APS has been applied to a variety of domains including chess (we call this system "Morph"), othello, pente, and image alignment [6].

For example, chess satisfies all of the abovementioned requirements of the complex problem domains considered in this paper: The game tree forms the state space, each game representing a search path through the space; reinforcement is only provided at the end of the game; and, finally, it has a large cardinality of states (around 10^{40}) [18]. Furthermore, few efforts use previous search experience in this area despite the high costs of search. In order to focus the research on the learning mechanisms, Morph has been constrained to using only one-ply of search. In addition, Morph has been given little initial chess knowledge, thus, keeping it within the experienced based learning framework. Despite these constraints, Morph now manages to draw its trainer Gnuchess (a tournament-level program) about once every 800 games, and is capable of defeating or drawing human chess novices.

Although Morph has been given very little assistance it would be false to say that it was supplied "no" domain knowledge. In earlier versions of APS, the system was supplied a pattern representation language by the user that was deemed to be appropriate for the given problem domain [6]. For example, some patterns constructed for the game Othello are displayed in Figure 0.1.

In the case of Morph, a graph representation of attacks and defends relationships was provided, as well as a notion of material. See Figure 0.2 for an example.

Further, a combining rule for patterns in a given domain and pattern creation mechanisms were also provided. In this paper, we describe recent efforts to build a fully domain-independent APS and to implement it within the PEIRCE conceptual structures workbench. For example, the patterns above can be represented in CGs as in Figure 0.3 .

We can now see that APS involves the automated construction of a semantic distance function over unpreclassified

To test these ideas APS is being applied in the "uninformed MetaGame universe" [15]. This is a universe of chess-like games in which the system is not supplied the rules or objective of the game it is playing!.

The structure of the rest of the paper is as follows. Section 2 discusses the current APS system framework in detail. In Section 3 we describe our effort in building a domain-independent pattern and semantic distance construction scheme. Section 3 also describes a scheme for consructing conceptual graphs out of sequences of raw bit data followed by reinforcement. Section 4 is the conclusion.

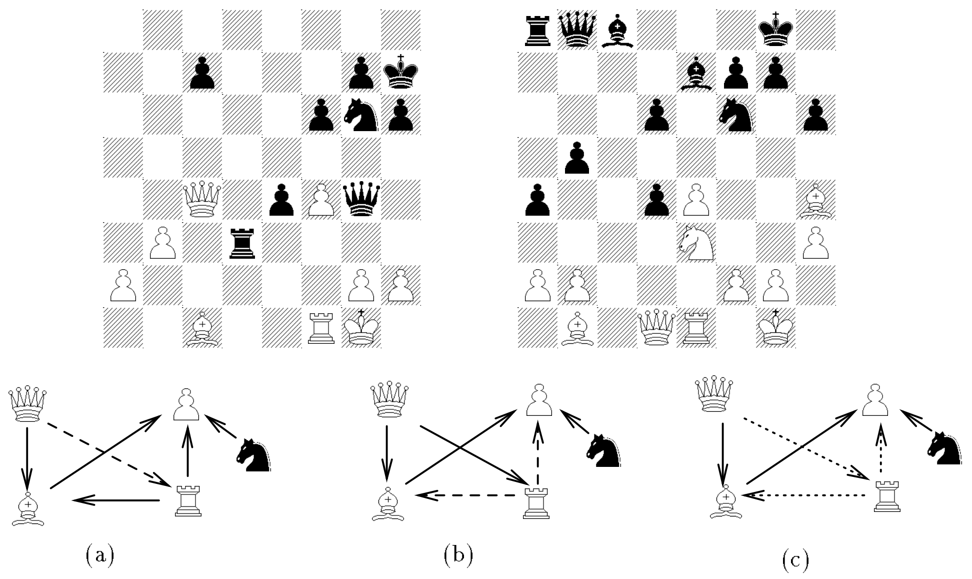


Figure 0.2: A generalization derived from two different chess positions. (a) is the subgraph derived from the board on the left and (b) is the subgraph from the board on the right. The solid edges correspond to direct edges between pieces and the dashed edges correspond to indirect edges. Graph (c) is the generalized graph derived from (a) and (b) in which the dotted edges have been generalized to generic "attacks."

0.2 The APS Model

As mentioned previously, the APS framework contains three major parts: the pattern-weight formulation of search knowledge, methods for creating and removing pws, and methods for obtaining appropriate weights for the pws with respect to reinforcement values. This section discusses each of these facets in detail, after which it describes how the parts interact and how the system performs as a whole.

0.2.1 Pattern weight formulation of search knowledge

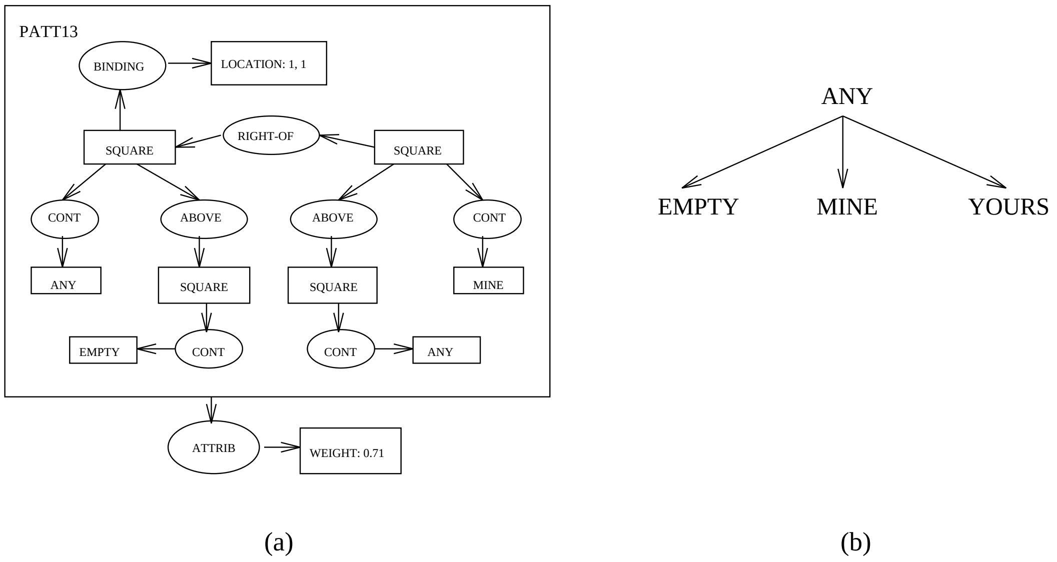
A pattern represents a boolean feature of a state in the state space. This feature typically represents a proper subset of all the possible properties of the state. That is the feature usually does not represent a single state, because such patterns would be far too specific (and numerous) to be useful in complex problem domains. Examples of patterns include graphs and sets of attributes . Often the language in which patterns are expressed is akin to the language used to represent the states, but with a higher level of abstraction. In the new domain-independent APS model all patterns for all domains are represented as conceptual graphs.

Each pattern has associated with it a weight that is a real number within the reinforcement value range. The weight denotes an expected value of reinforcement, given that the current state satisfies the pattern. For example, in a game problem domain a typical set of reinforcement values is {0,1}, for loss and win respectively. If we have a pw, <p_1, .7>, this implies that states which have p_1 as a feature are more likely to lead to a win than a loss.

The major reason for using pws over another form of knowledge representation is their uniformity. Pws can simulate other forms of search control and due to their low level of granularity and uniformity more power and flexibility is possible [8]. For example, pws have all the expressive power of macro tables. Additionally, they allow switching over from one macro sequence to another and allow for the combination of two or more macro tables[8]. Interestingly, Albert Einstein recognized the importance of macro-crossover and the patterns that arise from it:

> What precisely is "thinking"? When on the reception of sense impressions, many pictures emerge, this is not yet "thinking." When, however, a certain picture turns up in many such sequences, then - precisely by such return - it becomes an organizing element for such sequences, in that it connects sequences that in themselves are unrelated to each other.

Along these lines, patterns that occur commonly in many search or decision paths can and ought to be exploited in future problem-solving episodes.



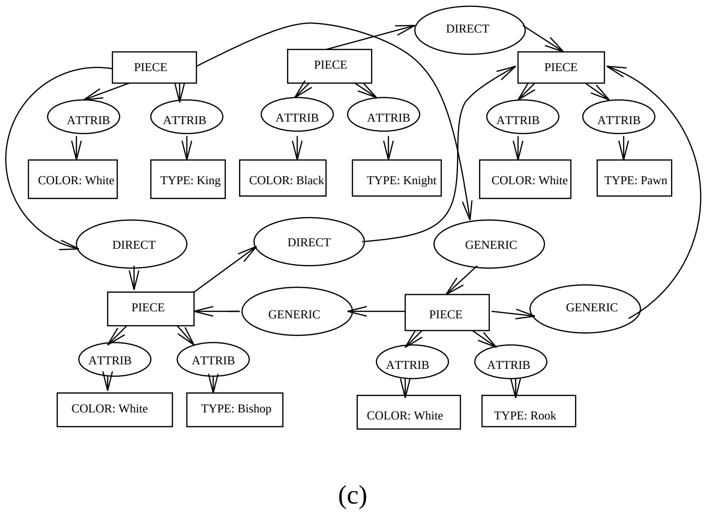


Figure 0.3: Conceptual graphs and a concept hierarchy from the Chess and Othello domains.

(a) is a representative Othello pattern. The associated concept hierarchy for square contents is in (b). (c) represents the generalized chess pattern from figure 0.2.

0.2.2 Adding and removing patterns

The patterns used to represent search knowledge are stored within a database that is organized as a partial order on the relation “more-general-than”. In the past patterns have been inserted into this database through the following four methods: search context rules, generalization and specialization, reverse engineering, and genetic operators[3]. The actual database insertion and retrieval takes place using algorithms that exploit the structure in the partial order to do a minimal number of comparison tests[5, 7].

Search context rules are the only pattern addition scheme that does not rely on patterns already in the database; thus, they are the only way patterns are added to an empty database. A search context rule takes as input a particular state and the sequence of all states in the last search and returns a pattern to be inserted into the database. A search context rule is a deterministic procedure that builds up a pattern given the previously mentioned inputs.

In concept induction schemes [9, 11, 13, 16] the goal is to find a concept description to correctly classify a set of positive and negative examples. In general, the smaller description that does the job, the better. Sometimes the concept description needs to be made more specific to make a further distinction. At other times it can be simplified (generalized) without loss of discriminative power.

Generalization patterns are created by extracting similar structures from within two patterns that have similar weights. A pattern is specialized in an APS system if its weight must be updated a large amount (indicating inaccuracy). Whereas in a standard concept induction scheme the more specific patterns may be deleted, the APS system keeps them around because they can lead to further important distinctions. The deletion module may delete them later, if they no longer prove useful.

Reverse engineering is a method used to add macro knowledge into an APS system. Macros can be represented as pws by constructing a sequence of them such that each pattern is a precondition of the following one. Successive weights in the macro sequence will gradually approach a favorable reinforcement; thus, the system is then motivated to move in this direction.

Reverse engineering extends a macro sequence by adding a pattern. This extension is similar to Explanation-Based-Generalization (EBG) [12] or goal regression. The idea is to take the most important pattern in one state, s_1, and back it up to get its preconditions in the preceding state, s_2. These preconditions then form a new pattern p_2. If pattern p_2 is the most useful pattern in state s_2, it will be backed up as well, creating a third pattern in the sequence, etc. The advantages of this technique are more than just learning "one more macro"; each of the patterns can be used to improve the evaluation of many future positions and/or to enter the macro at any point in the macro sequence.

The APS system makes use of genetic operators in order to add additional patterns into the database. Since patterns are not required to be represented as bit strings (in the new APS thet are conceptual graphs), it is up to the individual APS system to tailor the genetic operators to suit the pattern representation. The APS system does not remove all or most of a population of patterns, however, due to the large amount of time necessary in determining appropriate weights for all patterns.

Although a variety of pattern addition schemes are available, due to memory and processing restrictions, the database must be limited in size. As in genetic algorithms there must be a mechanism for insignificant, incorrect or redundant patterns to be deleted (forgotten) by the system. A pattern should contribute to making the evaluations of states it is part of more accurate. The utility of a pattern can be measured as a function of many factors including age, number of updates, uses, size, extremeness and variance. These attributes will be elaborated upon in the next section. We are exploring a variety of utility functions [10]. Using such functions, patterns below a certain level of utility can be deleted. Deletion is also necessary for efficiency considerations: the larger the database, the slower the system learns. For example, after 2000 games of training and with a database grown to 2500 patterns the Morph chess system takes twice as long to play and learn (from about 1200 games a day on a Sun Sparc II to about 600 games per day).

0.2.3 Modifying the Weights of Patterns

The modification of weights (of patterns) to more appropriate values occurs every time a reinforcement value is received from the environment. The modification process can be broken down into two parts. First, each state in the sequence of states that preceded the reinforcement value is assigned a new value using temporal difference learning. Second, the new value assigned to each state is propagated down to the patterns which matched that state. A pattern in the database matches a given state if the state satisfies the boolean feature represented by the pattern and no other pattern more specific than it matches the state.

Temporal difference (TD) learning determines new values for the states in the game sequence moving from the last state, S_n, to the first state, S_1. Since state S_n was the state at which reinforcement was delivered its new value is set to the reinforcement value. For all other states, S_i, the new value is set to a linear combination of its old value and the new value of state S_{i+1}. For example: if during play a state Ai was evaluated as 0.8 and the next state Ai+1 is determined to be 0.9 (by TD learning, since we work backwards from the last state) the next state might take on a new value of 0.85 which is halfway between the old and the new recommendation. This method differs from supervised learning, where the value of each state S_i is moved toward the reinforcement value. It has been shown that TD learning converges faster than the supervised method for Markov model learning [20]. The success of TD learning stems from the fact that value adjustments are localized. Thus, if the system makes decisions all the way through the search and makes a mistake toward the end, the earlier decisions are not harshly penalized.

Once each state S_i has been assigned its new value, each pattern matching S_i must have its weight moved towards the new value. Each weight is moved an amount relative to the contribution the pw had in determining the old value for the state.

This weight updating procedure differs from those traditionally used with TD learning in two ways. First, TD learning systems typically use a fixed set of features, whereas in an APS system, the feature set changes throughout the learning process. Second, the APS system uses a simulated annealing type scheme to give the weight of each pattern its own learning rate. In this scheme, the more a pattern gets updated, the slower its learning rate becomes. Furthermore, in addition to giving each pattern its own learning rate, the annealing scheme forces the convergence of patterns to reasonable values:

$$\text{Weight}_n = \frac{\text{Weight}_{n-1} * (n-1) + k * \text{new}}{n+k-1}$$

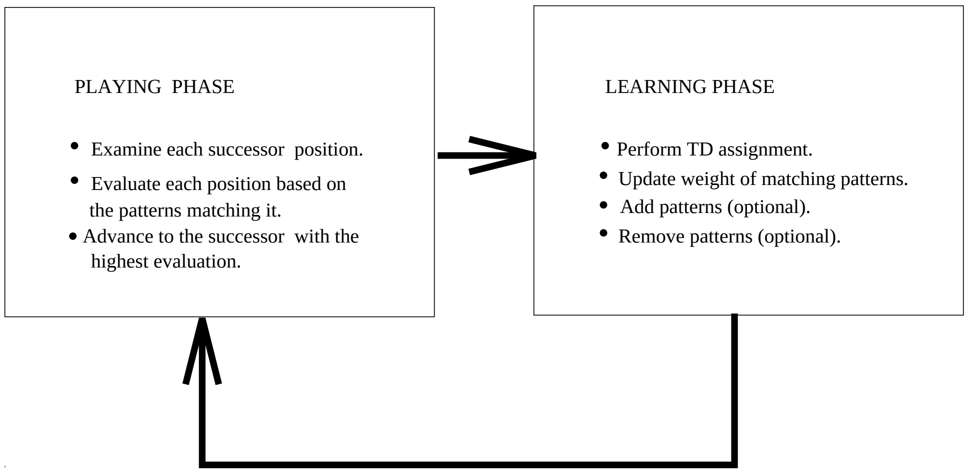

PLAYING PHASE
• Examine each successor position.
• Evaluate each position based on the patterns matching it.
• Advance to the successor with the highest evaluation.
LEARNING PHASE
• Perform TD assignment.
• Update weight of matching patterns.
• Add patterns (optional).
• Remove patterns (optional).

Figure 0.4: The execution cycle of an APS system.

$Weight_i$ is the weight after the ith update, new is what TD recommends that the weight should be, n is the number of the current update and k is the global temperature. When $k = 0$ the system only considers previous experience, when $k = 1$ the system averages all experience, and when $k > 1$ the present recommendation is weighted more heavily than previous experience. Thus, raising k creates faster moving patterns. As an example of how global temperature affects the learning rate, if the system doubles the global temperature, a pattern that has 100 updates will be updated like a pattern that has 50 updates the next time its weight is modified.

0.2.4 Integration: The workings of the entire system

This subsection describes the workings of a complete APS system by combining the parts described in the previous three subsections. An APS system executes by repeatedly cycling through two phases: a search phase and a learning phase (see Figure 0.4). In the search phase the APS system does not modify its knowledge base but instead performs a search using the current knowledge base. The learning phase then alters the knowledge base by adding and deleting patterns, and modifying the weights of patterns.

The search phase traverses a path from the initial state of the problem domain to a reinforcement state. Depth first hill climbing is the search technique used to traverse the search tree. In other words, at each state S_i in the search path the state moved into next, S_j, is determined by applying an evaluation function to each successor state of S_i and choosing that state which has the highest evaluation. Note, however, nothing prevents the database from supporting more sophisticated search strategies.

Although the exact calculations performed by the evaluation function depend on the particular APS system, the function has the following framework. It takes the state to be evaluated and determines the most specific pws in the pw database that match that state. In the hierarchy of patterns, the most specific Pws will manifest as immediate predecessors ("IPs") of the state pattern if it were inserted in the database. The weights of these most specific pws are then combined by a system dependent rule to determine the final evaluation for that state.

The learning phase, the second in the APS cycle of execution, takes as input the sequence of states traversed in the search performed by the first phase. This sequence is used by TD learning and simulated annealing to modify the weights of the patterns in the database. Patterns are then inserted into the database using the four techniques mentioned in Section 0.2.2. Finally, unimportant patterns are removed from the database as described in the same subsection.

The execution of an APS system involves the interaction of many learning techniques. A global view of this interaction is displayed in Figure 0.5. In this figure the edges are labelled with actions specifying whether a given module adds patterns, deletes patterns or modifies the weights of patterns. Central to the APS system is the pattern weight database, which holds all of the accumulated search knowledge generated and manipulated by the surrounding modules.

0.3 Making APS fully domain-independent

In current APS systems the user tailors the form of the evaluation function, pattern representation language and the pattern creation strategies (search context rules, generalization and specialization, reverse engineering and genetic operators) to a given problem domain.

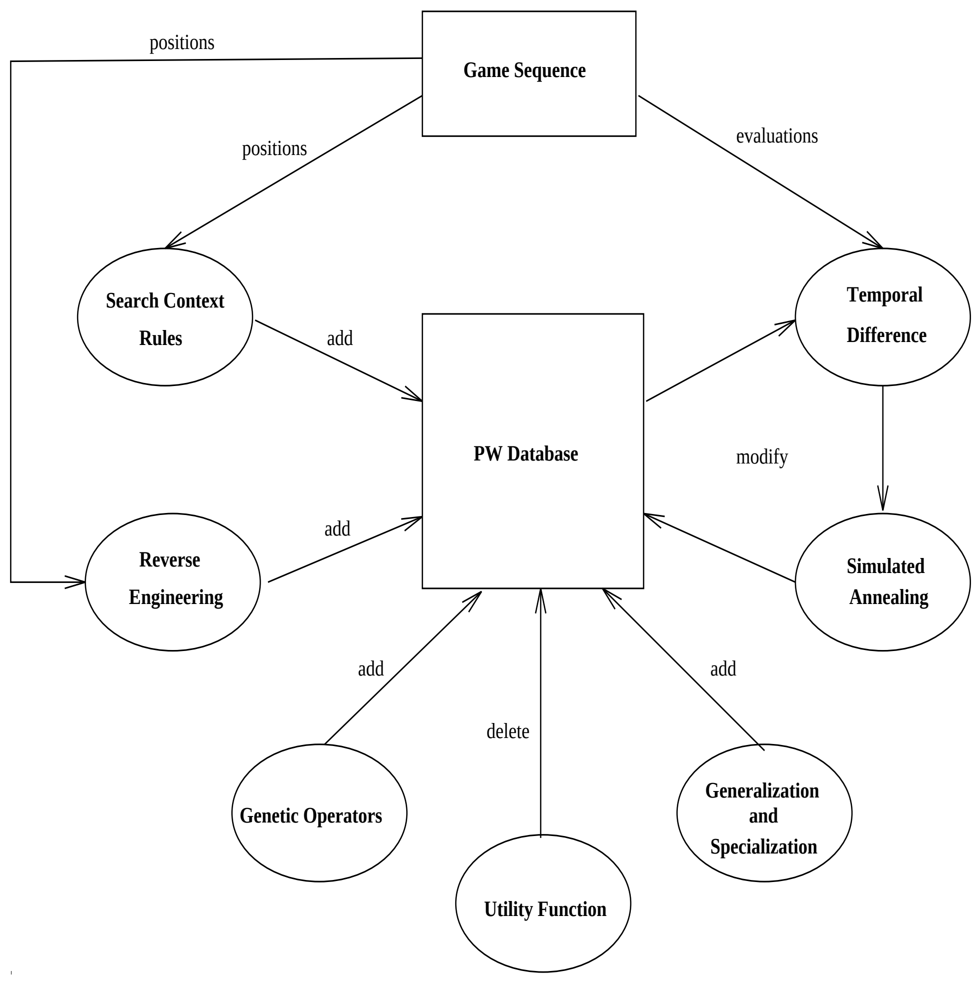


Figure 0.5: The integration of learning modules within an APS system.

Here we want to specify CG-based methods by which the system on its own becomes fully responsible for the features and relations making up its patterns, which patterns are added and how the weights of patterns are combined. The goal is to automate the knowledge acquisition process as much as possible.

We shall still be assuming however that the following has been supplied to the system:

- A state representation language in conceptual graphs must be defined. The concepts and relations in the CGs would correspond to the primitive percepts of the environment that are received (presumably) by low-level sensors or in the case of an abstract problem from the problem definition itself. In some cases it may be also appropriate to specify a type hierarchy for the concepts and relations if these are part of the sensing possibilities or the problem definition itself. But normally this additional knowledge would not be provided. not expected. In a chess problem the raw percepts would likely be the set of piece square relations on the board, though one could argue that geometric relations like “same rank”, “same file”, “near”, “far”, “same diagonal”, “same color square” could come from a visual sensor. For example states in the 8-puzzle [4] could be represented as collections of nine position-tile patterns (see Figure 0.6).

- A mechanism for periodically rewarding or punishing the system based on its performance.

  In principle this is all that would be provided to the system and all that is expected. *However, one of the beauties of APS is that if more is known about the problem domain it can be supplied by increasing the detail in the type hierarchies and state descriptions and by increasing the reinforcement method to include subgoals as well as final goals.*

  Further, an explanation system may be developed by having patterns recall those situations that led to their creation and those instances that have most influenced their weight. Although, this does not lead to a domain-theoretic explanation it does allow the system to provide examples (cases) that have led it to believe what it believes about the current situation. Using this information, the knowledge

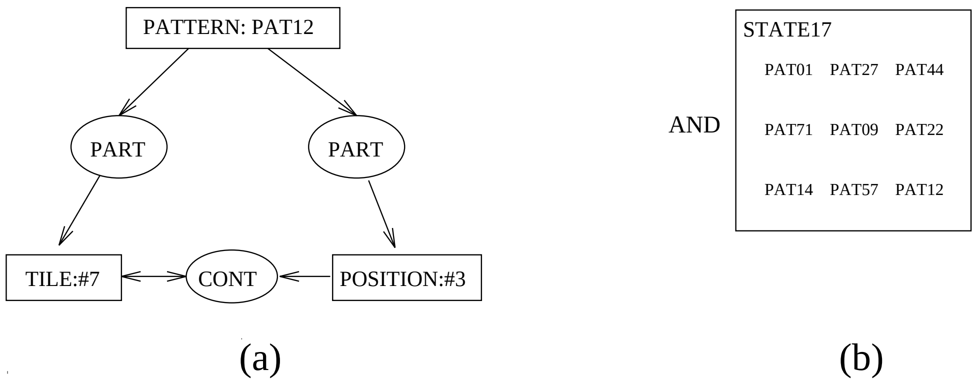


Figure 0.6: State representation in the 8 puzzle.

(a) is an individual tile-position pattern expressed as a conceptual graph. Note that the entire state space can be represented by a total of 81 such patterns. (b) is one particular board state composed of a conjunction of nine tile-position patterns. This state happens to be named **STATE17**.

engineer may be able to make further adjustments to the pattern representation scheme so that further distinctions can be achieved.

Finally, an interesting feature of the annealing scheme for weight learning is that a group of patterns (or the entire system) may be "disturbed" or "frozen" by raising or lowering their local temperature or the global temperature of the system. Thus, the domain engineer is not responsible for assigning the relative significances of system patterns, but may be able to intervene by raising the temperature of those patterns that don't appear to be behaving properly.

OK, this is well and interesting but our real goal is for the system to be as autonomous as possible! We will address two issues: How the system determines which patterns to create and how the system decides to combine the weights of patterns. But before delving in let's review the objective of APS: to improve with experience in complex state space search domains. Pattern-weight pairs are being used as estimates of distance to the goal state. Beyond this they have no meaning to the system:*the system is syntax-based: all the system cares about are the weights of the patterns that match - not the patterns themselves.* Once these weights are determined how should they be combined?

0.3.1 Giving autonomy to APS: semantic distance discovery

In the book Pattern Recognition[21], Watanabe gives the remarkable *Theorem of the Ugly Duckling*:

> Insofar as we use a finite set of predicates that are capable of distinguishing any two objects considered, the number of predicates shared by any two such objects is constant, independent of the choice of the two objects.

He continues:

> This theorem is true also if we limit the number of predicates of any given rank. If we decide to measure similarity of two objects by the number of predicates of a given rank, or by the total number of all possible predicates that are shared by the two objects, then we can say: 'Any arbitrary two objects are equally similar'. This theorem has a very profound meaning for pattern recognition. If we use the concept of distance D instead of similarity, the situation is the same; the distance between a pair of objects is the same for the choice of any of the pair.

Thus, given a fixed set of n predicates and building attributes out of all combinations of these predicates producing a boolean lattice, all objects will share (satisfy) the same number of points on this lattice. His conclusion is that the only way to distinguish between two objects is to use "extralogical" information, information from outside the set of objects themselves. On closer scrutiny one recognize that the extra-logical information must come from the context in which the system that is matching the objects is embedded. The context provides criteria for weighting each attribute in terms of importance in that context.

In the case of APS (and we believe most AI systems) the context comes from the goal or objective of the system. These goals are indirectly represented in the reinforcement the system receives. Thus, from the point-of-view of APS two states are to be considered similar if the expected value of their reinforcement is similar. In effect, the predicates of the Ugly Duckling Theorem will be weighted based on their importance in determining a final reinforcement value. From this perspective APS can be viewed as the automated constructor of semantic distance functions, where semantic similarity is a function of final reinforcement values.

Recall that all knowledge stored by APS takes the form of pattern-weight pairs: predicates coupled with weights. The weights can be viewed as the expected reinforcement value for all states that contain the pattern as a subpattern. Now each pattern set induces a set of equivalence classes over the state space: states are in the same equivalence class if they match exactly the same set of patterns.

We now suggest the following methodology by which the pattern construction and weight learning schemes can become fully domain independent as a result of incorporating the semantic distance understanding. The basic philosphy is to start with the set of domain primitive relations and concepts and gradually form more complicated graphs out of the interactions between these concepts. The pattern deletion mechanism and weight-learning mechanisms of the previous section still apply. Generalization operators, search context rules and reverse engineering however may well be no longer necessary. The notion of "adjacency" described in the subsection below on relation discovery can be used to get the effect of reverse engineering and search context rules in a fully-domain-independent manner.

1. Store primitive graphs that correspond to individual primitives and relations of the state description scheme. Give them equal weight for starters.
2. Store all state CGs previously seen. Assign each of them a weight equal to the target value supplied by TD learning. In effect we are creating the best value for the state known. Periodically, when the state store gets filled a procedure can remove those states that have not been seen for a while. If a state should re-occur in later games it gets the new value.
3. For all patterns that will be stored in the database (by future operations) assign them a weight equal to the weighted average of the weights provided by its successor graphs:

$$W_{new} = \frac{\sum_{i=1}^{n} W_i G_i}{\sum_{i=1}^{n} G_i}$$

   where n is the number of new's successors, W_i is the weight of the ith successor graph, and G_i is the game number of graph i's last occurrence. We are assuming that a state evaluated in later games (search experiences) has a more accurate value than states evaluated in earlier games.
4. Now, to evaluate a state S one uses the "semantic distance" mechanism of the Method IV retrieval scheme [5, 7] which will return for each graph a "closeness" determined as follows:
   (a) Convert S to a conceptual graph as always.
   (b) Insert S in the database. By so doing we get a set of immediate predecessors(IP) for S and maximum common subgraphs of S with everything in the database.
   (c) Now, one could evaluate S by a function of its IPs as before but instead we use these IPs to construct a semantic distance function between S and all pre-stored states:

$$SemanticDistance(S) = \sum_{i}^{sharedIPs} extr(i) - \sum_{j}^{non-sharedIPs} extr(j)$$

   where

$$extr(i) = \mid W_i - \overline{r} \mid$$

   describes the degree to which pattern i's weight deviates from the average; $\overline{r}$ is the average reinforcement value for all states and W_i is the current weight for pattern i.
5. Then by looking at graphs that share several IPs and using the semantic distance function it is possible to get a set of "close" matches. Either use the weight (pre-stored evaluation) of the closest match or a weighted average of a set of close matches of varying degree.

This scheme is similar to the "nearest neighbor" approaches applied to bit vectors in pattern recognition or information retreival tasks [14]. Here these approaches are being generalized to graphs and higher level features are being developed dynamically. Further, closeness is a function of the reinforcement values rather than assigned apriori.

The key remaining question is where do the patterns used in the above scheme come from? The most useful patterns that can be created are those which are most relevant to determining semantic distance. The ideal patterns will have extreme average weights with low standard deviation and apply to many states. Of these factors low standard deviation is most important because it ensures accuracy. The other two factors ensure general usefulness.

A utility function such as

$$\frac{N * Extremeness(i)}{\sigma}$$

where N is the number of states in the database, $Extremeness(i)$ is the extremeness of the ith pattern's weight as defined above, and σ is the standard deviation of the pattern's weight over time, will be used by the deletion function to remove patterns periodically.

Patterns are inserted based on the following domain-independent scheme: *contrast this mechanism with the domain-specific mechanisms of the previous section.* Consider the IPs found from the current query. We form three classes based on the following mergers:

- Take the two IPs with highest weight. Form a new pattern by "merging" them. Mergers take place by forming a CG out of the conjunction of the objects (using AND) (for example see Figure 0.3.1 or by creating a new CG through *maximal join* out of the IPs if they overlap.

  Do the same with IPs with the lowest weight.
- Finally, do the same with the IPs of lowest and highest weight.

These new classes should lead to more accurate semantic distance estimation than before: Merging the two highest weight IPs and two lowest weight IPs are likely to lead to classes with more extreme weights. Merging the high with the low will create a class of states that are likely to be similar to the current state since these important conflicting factors arise in both.

Finally, new patterns are also formed by a "counting" operator that counts for a given state graph the number of times a given subgraph is repeated. A pattern plus a count becomes a new pattern. This gives the system the capability of learning notions like "material" or mobility in chess which are features of an entire state rather than individual elements of that state.

It would indeed seem surprising if such a scheme could construct features that capture the essence of a state or problem domain. But we argue that in principle that if patterns exist for representing this information it will be first reflected in the domains primitives and then gradually propagated to more complex patterns. Of course for some theoretically constructed domains this may not be the case since the useful patterns may be very remote from the state descriptions, but in practice these methods should serve well , especially when coupled with the statistics-based relation learning methods described below.

0.3.2 Relationship to previous work on semantic distance and in conceptual graphs

Sowa [19] defines semantic distance based on a frequency interpretation. Types that are shared are considered to have importance inversely proportional to the frequency of occurence of them or their subtypes in "schemata" or stored database graphs. Sowa's notion of types may be generalized (as we do above) to higher order intermediate graphs that are shared. We have been weighing the sharing and non-sharing of these graphs depending on their extremeness. It is interesting to note that the patterns that have extreme values, by the nature of the normal distribution of weights of patterns (which is observable empirically) are the patterns whose weights occur less frequently). Here frequency is taken with respect to the rareness of the expected reinforcement rather than rareness of pattern occurrence. We believe that this is indeed the perspective that ought to be taken. For example, it is possible for a graph to occur infrequently (such as all squares on the queen file being occupied), yet be unimportant as well with respect to determining the reinforcement value. It is also possible for a pattern with a releatively extreme weight like "up a knight" to occur fairly often. In fact, all else being equal, the most useful patterns are those with extreme weights, that occur frequently and have a low standard deviation (weight variance over time).

Sowa and others [1, 17] also evaluate maximal joins between a query graph and a database graph by using other syntactic criteria (such as path distance between types and common subtypes) that ignore the context in which the join will be used. While these syntactic criteria have merit, there is also danger involved: To accept a purely frequency or syntax-based interpretation of relevance(importance) is to ignore the lesson of the Theorem of the Ugly Duckling, and to assume (with some risk) that the specific graphs and types stored in the hierarchy were designed with frequency or distance inversely correlated to importance in the specific application being considered.

0.3.3 Automated discovery of relations

An important remaining question is to what degree the system can learn to construct new relations and edge types on its own since these may not be part of the original state description. In fact, let us assume that NO relations or type hierarchy information is given in the original state description. All the system has to go by are a set of unlabelled (but identified) predicates (P1,.......,Pn) that can be true or false in a given state (this would be represented as a CG with n disconnected nodes), sequences of such states coupled with resulting reinforcement values. This puts us in the same framework as any learning or pattern recognition algorithm which is only given raw bit vectors or sequences of them. For the case of chess, 64 (for each square) times 13 (for each piece typ e and blank) bits would be assigned for each state. Surprisingly, from just this information and no knowledge of the rules or objectives of the domain a tremendous amount can be learned (given enough experiences and computational power). *Though not always obvious, the underlying mathematical structure of a system (problem domain) will eventually exhibit itself through interaction with another system (intelligent agent).*

We suggest the following multi-step algorithm for uncovering the structure (in terms of Conceptual Graphs with expected reinforcement) of a given problem domain:

1. Determine discrete multi-valued variables underlying the data. These will exhibit themselves as a set of predicates such at most one is true at a given time. In effect we would like to partition the space into possibly overlapping classes such that two predicates are in the same class if they are never both simultaneously true. This can be done as follows:
   (a) By iterating through all state experiences make a "compatibility matrix": a matrix of compatible pairs: (two nodes are compatible if they never co-occur as true). It is likely that this step will find all incompatible pairs, thus little backtracking should be required in the following step.

   (b) Then using brute force backtracking over the set of compatibilities form larger sets, such that all elements in the set are mutually compatible. Keep those classes that are not assumed by any others. Given a set of S compatible nodes, the remainder of the compatible nodes for that class must be a subset of the intersection of the compatibilities from each node in S. For example, in chess 64 variables for each square on a chessboard and 13 for piece types would be found. For an nxn tile puzzle, given n^2 input nodes, the 9 tiles and nine positions would be discovered.

   (c) Once the multi-valued variables have been found, in the state representation each class of n nodes can be collapsed to a single "variable" node connected to a "value" relation node that will point to 1 of n constants. (Since the value relation is unary, at the implementation level it may be omitted with the constants being attached directly as referents to a variable node). Thus the chess domain will have 77 variables and the tile puzzle 18.

2. Next one finds the statistical correlation between each variable node-value pair, single node predicates and the reinforcement value. This correlation is stored as a unary attribute with each node. (At the implementation level relation nodes are unneccesary since binary relations may be represented as directed edges.)

3. By looking at the correlations of pairs of nodes it is possible to find AND, OR, XOR, NAND and NOR relationships between pairs of nodes. These node pairs would also be labeled with the correlation with the reinforcement value. A matrix with all correlation values (unary and binary) we will call a "correlation structure". Existing well-understood statistical methods are fully applicable here [2], yet rarely have they been used as a basis for symbolic representation. As statistics is the science of processing numerical experiential data (to reveal mathematical structure), its uses within AI have been dramatically under exploited.

4. An "adjacency relation" can be defined between one and more conditions with edges going out of conditions that disappear with a single operator application (recognized as two sequential states) and an edge going in to conditions that are newly created. Edges between nodes shared by the same variable are assumed. Having an adjacent condition is a necessary but not necessarily sufficient condition for achieving the transposition defined by the two nodes. The structure that represents this adjacency information we term an "adjacency hypergraph".

5. The collection of the compatibility matrix, correlation matrix and the adjacency hypergraph we term "basis conceptual structure" for the given problem.

6. Using refinement [5] equivalence classes of nodes in the basis structure may be identified using some theshhold for matching the real-valued correlations. Nodes with similar properties should be treated as similar in processing. For example, it should be possible in principle to discover that the 8-puzzle really has only three distinct classes of tiles: blank, corners and sides and 6, 3, and 6 classes of positions, respectively, for each of these (depending on relation to the goal state). Thus a problem of 81 primitives has been reduced to 15.

7. Now for any input state, for those predicates that are true, the subgraph induced by those predicates (through projection) from the basis strucutre is formed and processing may continue as in the traditional APS and semantic distance approaches discussed above or by recursing such that the newly formed edges (due to the above steps) become the nodes on the next iteration. Note that for graph matching purposes two predicates that are shared by the same variable (or in the same equivalence class) will be considered equivalent and the labels of predicates will be ignored (what is relevant is their correlation with the reinforcement value and relationship to other predicates).

0.4 Conclusions and Ongoing Directions

We believe that several crucial mistakes have been made by AI researchers over the past thirty years. These mistakes have limited AI's success in creating a true computational model of intelligence.

The first pervasive mistake is the anthropomorphising (humanizing) of the notion of intelligence. To restrict intelligence to what people do is to obscure the true meaning of that word. Intelligence ought to be identified with optimal problem-solving(decision-making) under resource constraints in the achievement of specific goals. Any given agent (human, mechanical or otherwise) could then be placed on the spectrum (actually a partial-order) from no intelligence (random or destructive behavior) to perfect intelligence (optimality). The potential for computers to far exceed humans on this scale is due to their flawless memory, endurance, calculating accuracy, and wide resources. Very few humans ever penetrate a problem to its depths. What are called "experts" are often simply those with greater experience. And what we call "creative" or "smart" is sometimes merely clever or tricky, rather than due to deep understanding.

The second mistake, which is due to the high role humans have presumed for themselves on the spectrum of intelligence, is the belief that supplying a system with pre-digested domain knowledge is required and the proper path for AI. Such human assistance could never achieve "real intelligence" but merely simulation, fakery, or illusion. Indeed, research in AI has seemed to focus on the word "artificial" rather than "intelligence". Supplying domain knowledge to a computer certainly has practical value in applications, for testing computational models, and for exploring certain micro-issues, but it is inconsistent to rely on this while maintaining a research goal of creating true computational intelligence. For in our minds, domain-independence goes hand in hand with intelligence. To be fully domain-independent a machine must then develop its knowledge from experience. Knowledge is not some important but little-understood element, but rather an abstract representation of the underlying mathematical or statistical structure of a problem-domain.

A third mistake (arising from the other two mistakes) is to evaluate AI systems using analogies with human cognition or domain-dependent criteria. The difficulty with drawing comparisons to human cognition is that our own self-understanding is limited and that cognition is often far from optimal. Interestingly, in those cases in which human processing is highly accurate (as in the human vision system) this can be seen as the results of years of evolutionary experience. Again, experience is the key. To evaluate systems with domain-dependent criteria while supplying domain-dependent data is to do engineering and applied research - not fundamental research.

Thus the development of a computational framework for experience-based learning is a difficult but critical challenge. Here, we have argued for the necessity of a multi-strategy approach: At the least, an adaptive search system requires mechanisms for credit assignment, feature creation and deletion, weight maintenance and state evaluation. Further, it has been demonstrated that the TD error measure can provide a mechanism by which the system can monitor its own error rate and steer itself to smooth convergence. The error rate provides a metric more refined but well-correlated with the reinforcement values and more domain-specific metrics. Finally, in a system with many components (pws) to be adjusted, learning rates should be allowed to differ across these components. Simulated annealing provides this capability.

APS has produced encouraging results in a variety of domains studied as classroom projects [6], including Othello, Tetris, 8-puzzle, Tic-Tac-Toe, Pente, image alignment, Missionary and Cannibals and more. Currently, others are studying the application of the Morph-APS shell[1] to GO, Shogi and Chinese Chess. But much more distance remains to be covered before Morph or other experience-based systems will learn from experience with nearly the efficiency that humans do (and then go beyond). To achieve this, substantial refinement of the learning mechanisms and an enhancement of their mutual cooperation is required. *We believe that the new semantic distance learning approach plus the statistical relation construction described in this paper is an important step in that direction and further that the domain-independent framework has been established for future developments.* While time will tell whether the specific techniques described herein will be successful in creating true AI, we are for the moment satisfied that we are asking the right questions.

Acknowledgments

Many thanks to Claude Noshpitz who did the drawings and helped in general with the final draft. The research was partially funded by NSF grant IRI-9112862.

[1]Now publicly available via anonymous ftp from ftp.cse.ucsc.edu. The file, morph.tar.Z, is in directory /pub/morph/.

References

[1] J.L. Aronson. Truth and semantic networks. In *Fourth Annual Workshop on Conceptual Structures*, pages 161–171, 1989.

[2] H.W. Davis and S. Chenoweth. The mathematical modeling of heuristics. *Annals of Mathematics and Artificial Intelligence*, pages 191–228, 1992.

[3] J. Gould and R. Levinson. Experience-based adaptive search. In *Machine Learning:A Multi-Strategy Approach*, volume 4. Morgan Kauffman, 1992. To appear.

[4] R. E. Korf. *Learning to solve problems by searching for macro-operators.* Pitman, 1985.

[5] R. Levinson. Pattern associativity and the retrieval of semantic networks. *Computers and Mathematics with Applications*, 23(6-9):573–600, 1992. Part 2 of Special Issue on Semantic Networks in Artificial Intelligence, Fritz Lehmann, editor. Also reprinted on pages 573–600 of the book, Semantic Networks in Artificial Intelligence, Fritz Lehmann, editor, Pergammon Press, 1992.

[6] R. Levinson, B. Beach, R. Snyder, T. Dayan, and K. Sohn. Adaptive-predictive game-playing programs. *Journal of Experimental and Theoretical AI*, 1992. To appear. Also appears as Tech Report UCSC-CRL-90-12, University of California, Santa Cruz.

[7] R. Levinson and G. Ellis. Multilevel hierarchical retrieval. *Knowledge-Based Systems*, 1992. To appear.

[8] Robert Levinson. A pattern-weight formulation of search knowledge. Technical Report UCSC-CRL-91-15, University of California Santa Cruz, 1989. Revision to appear in Computational Intelligence.

[9] R. S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine learning: An Artificial Intelligence Approach.* Tioga Press, 1983.

[10] S. Minton. Constraint based generalization- learning game playing plans from single examples. In *Proceedings of AAAI-84*, pages 251–254. AAAI, 1984.

[11] T. M. Mitchell, J. G. Carbonell, and R. S. Michalski, editors. *Machine Learning: A Guide to Current Research.* Kluwer Academic Publishers, 1986.

[12] T. M. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation based generalization: A unifying view. In *Machine Learning 1*, pages 47–80. Kluwer Academic Publishers, Boston, 1986.

[13] T Niblett and A. Shapiro. Automatic induction of classification rules for chess endgames. Technical Report MIP-R-129, Machine Intelligence Research Unit, University of Edinburgh, 1981.

[14] S. Omohundro. Efficient algorithms with neural network behavior. Technical Report UIUCDCS-R-87-1331, University of Illinois, April 1987.

[15] Barney Pell. A computer game-learning tournament. (In Preparation), 1991.

[16] J. R. Quinlan. Induction on decision trees. *Machine Learning*, 1:81–106, 1986.

[17] A. Ralescu and A. Fadlalla. The issue of semantic distance in knowledge representation with conceptual graphs. In *Proceedings of Fifth Annual Workshop on Conceptual Structures*, pages 141–142, 1990.

[18] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(7):256–275, 1950. See also Levy, D.N.L. (ed) (1988), *Computer Games I*, Springer Verlag, New York, pp. 81–88.

[19] J. F. Sowa. *Conceptual Structures.* Addison-Wesley, 1983.

[20] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, August 1988.

[21] S. Watanabe. *Pattern Recognition:Human and Mechanical.* Wiley, New York, 1985.