General Game-Playing and Reinforcement Learning

Robert Levinson

UCSC-CRL-95-06
supersedes UCSC-CRL-93-38 and UCSC-CRL-94-32
partially supported by NSF Grant IRI-9112862
May 5, 1995

Department of Computer Science, University of California, Santa Cruz, CA 95060
E-mail:levinson@cse.ucsc.edu
Phone: 408-459-2087

ABSTRACT

This paper gives a blueprint for the development of a fully domain-independent single-agent and multi-agent heuristic search system. It gives a graph-theoretic representation of search problems based on conceptual graphs, and outlines two different learning systems. One, an "informed learner," makes use of the the graph-theoretic definition of a search problem or game in playing and adapting to a game in the given environment. The other, a "blind learner," is not given access to the rules of a domain, but must discover and then exploit the underlying mathematical structure of a given domain. Relevant work of others is referenced within the context of the blueprint.

To illustrate further how one might go about creating general game-playing agents, we show how we can generalize the understanding obtained with the Morph chess system to all games involving the interactions of abstract mathematical relations. An example of a monitor for such domains is presented, along with an implementation of a blind and informed learning system known as MorphII. Performance results with MorphII are preliminary but encouraging and provide a few more data points with which to understand and evaluate the blueprint.

Keywords: games, mathematical structure, heuristic search, machine learning, hypergraphs, neural networks, analogical reasoning, RETE, relational patterns, hierarchical reinforcement learning

Contents

1 Introduction

Artificial intelligence research would benefit from a unified view from which to study the past and a practical plan from which to proceed into the future. With this objective in mind, we suggest the following tenets or guiding principles for developing machine intelligence:

1. Intelligence is optimal problem solving in pursuit of specific goals under resource constraints. (Note that no mention is made of human cognition or consciousness).
2. Given this definition, domain-independence and adaptability are fundamental aspects of intelligence. (Not to adapt, if economical, is sub-optimal)
3. Single and multi-agent state space search problems are a large and important class of problems in which to develop and study machine intelligence.
4. The task before us is to develop fully domain-independent methods for working in state-space search domains.
5. The knowledge required to perform well in these domains is embodied in the definition of the domain and the mathematical structure of the state space, i.e. the reasons why a heuristic is good are explainable within the framework of the mathematics of the space.
6. Experience in a state space reveals mathematical structure that can be exploited by an optimal problem solver. (What is not revealed, need not be exploited).
7. For many practical problems, experience alone reveals enough structure to lead to efficient problem solving.
8. The underlying mathematical structure in these domains is independent of the labels given to the conditions and operators in the state-space definition.
9. This mathematical structure is embodied in the interactions of the conditions, defined by the operators of the domain, the relationship of this interaction complex to a given state, and the relationship of the given state to the goal state.
10. The effects of such interaction complexes are entirely domain-independent and are governed by regular laws, just as such interactions in matter and energy are governed by the laws (not necessarily all known) of physics.
11. These laws once discovered and exploited by computers will make them intelligent under the definition given above.
12. Computers may be an important tool in the process of discovering these laws.

With these tenets in mind and the ultimate goal of discovering the laws of state-space search before us, we suggest a division of research efforts to work on these separate but absolutely complementary projects:

- GENERIC-REPRESENTATION: A mathematical (probably based on graph theory and group theory) representation of the definition of a single-agent or multi-agent state-space search problem that is independent of all domain-specific or arbitrary labels, i.e. a program that can convert the declarative definition of a state space into a more generic, but equivalent, mathematical representation.
- INFORMED-LEARNER: A program that performs well in state-space search given the abstract mathematical definition of the state space, supplied domain-independent heuristics and experience.

- BLIND-LEARNER: A program that performs well on state-space search problems given no definition of the state space or pre-supplied heuristics — just the rewards at the end of the game, experience and legal states (as raw bit vectors) to choose from at each choice point.
- MONITOR: A programming environment that allows experiments to be easily designed and carried out on the above topics.

None of these projects is designed to directly produce the laws of state space search but to gradually give us an understanding that could lead to these laws and to their empirical validation. Meanwhile work on these problems increases the power of the machine. In this paper, in addition to summarizing previous research, we outline our own work on each of the four subprojects which has led to a system known as MorphII. We hope that an understanding of MorphII will lead to further appreciation of (and work on) the blueprint we have laid out above for AI research efforts.

MorphII is a successor of the Morph chess system [LS91, GL94] that has achieved approximately novice strength despite using just 1-ply of search and few human-supplied heuristics. MorphII is a fully domain-independent version of Morph as opposed to the original which despite good intentions, carries some human biases and chess idiosyncrasies. Further, we believe the new learning mechanism described in the sections on informed-learning and blind learning directly addresses limitations of the original Morph model: overly-specific graph patterns and insufficient freedom given to the system with respect to the class of patterns that can be formed and to the combination of their weights.

1.1 Previous Work By Others on General-Game Playing

Fortunately, work on these four projects (directly and indirectly) has been taking place both within the computer game-playing community and with AI in general. The most popular approach to date in game-playing research has been the incorporation of search coupled with game-specific evaluation. The search approaches [Kai90] typically meet our objectives for domain-independence and generality, but the evaluation mechanisms generally do not. In addition, until a domain is solved (i.e., perfect play can be achieved within specified resource constraints) the choice of a search algorithm is itself a heuristic. Thus, even the traditional domain-dependent approach could benefit from a more thorough mathematical understanding of state-spaces.

Minimax alpha-beta search was introduced in 1956 by John McCarthy [RN94], and has been used in Samuel's original checkers program and by the large majority of game-playing systems since. Thorough analytic studies of alpha-beta were done by Knuth and Moore [KM75] and later by Pearl [Pea82]. In addition to alpha-beta there has also been a large amount of research on selective search and pruning algorithms [Kai90]. Although many of these algorithms have been applied only to a small number of popular games like chess, checkers and Othello, they are general enough that they could be applied to any perfect-information game. Algorithms in this category include B* [Ber79], SSS* [Sto79] and conspiracy number search [McC98]. Baum et al. recently gave a mathematically well-founded algorithm [Bau93]. Beal [Bea80] and Nau [Nau80] constructed hypothetical "pathological" games in which alpha-beta search becomes less accurate the more deeply it searches, and thereby showed that the approach is not fully applicable, despite the fact that it seems to work well on most popular games. Search strategies also become significantly less effective in games of imperfect-information or games with large branching factors (such

as Go). Finally, minimax search strategies maximize performance, assuming a worst-case scenario in which the opponent always selects the best moves. A number of important research issues remain in which this assumption has been relaxed (e.g., trying to play into positions in which the opponent is likely to not find the winning line [Jan90]). In this case statistical and mathematical models of the game and the opponent increase in importance.

Recently, Beal [BS94] showed experimentally the interesting result that for certain games such as chess, even random evaluations when coupled with minimax can lead to improved play with increasing search-depth, because maximizing and minimizing favors nodes with large branching factors and hence roughly corresponds to a mobility coefficient. This apparently simple result has important consequences for adaptive game-playing systems, as it provides a mechanism by which a system can bootstrap itself from no domain knowledge and the ability to use a few ply of search, in those domains where mobility is relevant.

Single-agent problems have, of course, also been approached through search with such algorithms as A* [Nil80], best-first-search [RK91], and means-ends-analysis [EN69]. To discuss single agent search in depth is beyond the scope of this paper. The important thing to note about the traditional search approach is that it tends to obviate the need for the program or programmer to have a thorough mathematical understanding of a domain, given that a fast and reasonably accurate heuristic evaluation function is available. Although such an approach has many positive practical consequences, it does not necessarily increase our scientific understanding of how and why heuristics and search algorithms perform as they do. Further, as the existing search approaches have important limitations (such as horizon effects) in complex domains, a thorough understanding of the mathematics of state-space problems would undoubtedly lead to more effective search algorithms as well. The limitations of the search approaches have suggested the use of abstract and hierarchical planning algorithms that attempt to view a game state as a collection of subproblems to be solved [Wil80, Pit76, Min84, LNR87, FD89]. To date, these approaches have not proved powerful or general enough to be serious practical competitors to brute-force search. But some researchers expect that to change once the mathematics behind planning is understood and a wider variety of games such as those with imperfect information are considered [SN93].

A number of individual games have succumbed to mathematical and computer-aided analysis [BCG82, All92, Vaj92]. We believe that through a general graph-theoretic approach this type of analysis can be extended to much wider classes of games and problems. For example, mathematical techniques have recently been applied successfully to previously considered intractable Go endgames [BW94].

Reinforcement learning has already proved to be highly successful in learning the evaluation functions for specific games such as checkers [Sam59], Othello [LM88] and backgammon [TS89]. The latter two programs developed to world-class players in their respective domains. Such successes provide important datapoints in pursuing the blueprint; these systems used only a moderate amount of human assistance (in defining their feature and training sets) and thus were not too far from the ideal of full domain-independence. Reinforcement learning in general is becoming increasingly popular [Sut91] because it can minimize the need for human assistance.

Neural networks [RM86] and genetic algorithms [Bet81, BGH89, Gol89, Hol75] can be viewed as blind-learners. These systems attempt to learn functions given pre-classified input-output pairs. To the extent that the supplied classifications can also be generated automatically (with temporal-difference learning [Sut88], for example) the learned functions coupled with search would produce a general game-playing system. However, these systems

do not yet incorporate a thorough graph-theoretic understanding of machine learning domains in their structure. They would be lost in complex domains such as chess, in which high-level relationships and analogies (or large amounts of search, as is currently done) must be exploited to handle the large combinatorics of the sample space. It should also be noted that most reinforcement learning systems and neural nets do not derive their structure directly from a declarative specification of a domain , but instead have their structure supplied by a human or have a fixed structure that is hoped to be generally applicable. To date, most such systems only perform well in domains for which their application was anticipated and for which they were tuned. Inductive learning algorithms [Qui86, Mic83] have tended to suffer from similar limitations, despite showing strength in well-controlled settings. Recent developments in inductive logic programming [Mor94] may eventually make inductive learning more generally applicable. To achieve the goal of the informed learning project, the structure of the learner itself may need to be dynamically adjusted by the system in a way that is specifically suited to the problem in question. With MorphII we illustrate one method in which this might be done.

2 Generic Representation of Games and Search Problems

In this section we illustrate how the GENERIC-REPRESENTATION of games may be pursued in graph-theoretic terms. We claim that a large class of state-space games can be represented by utilizing the notion of directed hypergraphs. A *hypergraph* is a set of nodes S coupled with a set of hyperedges that are subsets of S. A hyperedge is *directed* if its nodes are ordered. A hypergraph is *nested* if its nodes themselves may be hypergraphs. We will call a hypergraph *basic* if it is not nested.

We now define the following game that we call the "generic-hypergraph-game." The specifications have alternatives that may be selected to form other games.

- Each player starts with 0 points.
- Let P1,...,Pn be a finite set of primitive boolean conditions.
- States are n-tuples [P1,...,Pn] representing the truth-values of these conditions.
- The game starts in some specified initial state or the initial state is selected randomly. Precisely, the initial state is chosen randomly from a set of states satisfying a given set of conditions.
- An operator is an ordered pair of sets of primitive boolean conditions. The first represents its preconditions and the second its postconditions. An operator is legal at a given state if its preconditions are satisfied. If an operator is selected, minimal changes are made to the current state so that the postconditions become true. Alternatively, operators could have probabilistic effects. At each state one agent is asked to select an operator from among the legal ones.
- Another set of operators is executed automatically after each state is created. These "reward operators" are usually used for standard bookkeeping operations such as assigning the proper number of points to each player.
- Terminal conditions are hyperedges, such that if each of their conditions is achieved the game is over. Games also terminate if no legal moves are available for the player to move or (possibly in addition) if a position repeats with the same player to move.
- Players alternate turns selecting applicable operators. The player with the most points at the end of the game wins.

- Knowledge limitations: Normally each player has perfect knowledge of the operators and of the current state. Variants include restricting each player's knowledge to a certain set of conditions in the current state, or to certain definitions of the operators.

2.1 Generic Game Taxonomy of Popular Games

Here we outline how popular games can be viewed as variants of the generic hypergraph game of the previous section.

- Single-agent search problems such as the tile puzzles fit naturally in this framework. If the object is to use as few moves as possible, a primitive condition is set that is unaltered by the rules and decrements the player's score by 1 (through the reward operators) each time it occurs.
- Tic-tac-toe-like games (such as Qubic [All92], Renju and GoMoku) in which objects are placed and never moved, have operators with one pre-condition, one post-condition and all terminal hyperedges also produce positive rewards.
- Hex is a game played on a grid of hexagons. Players alternate taking possession of hexagons; one player tries to make a path of his own hexagons from top to bottom and the other player tries to make a path from left to right. Hex fits the tic-tac-toe format with the hexagons as nodes, the various paths as the hyperedges, and rewards assigned correspondingly to whichever player can win in that path. But Hex can also be viewed as a "more complex" tic-tac-toe like game in that there are an exponential number of hyperedges per nodes in the graph. Hypergraphs which exhibit this property (including those for checkers and Go) have been shown to be PSPACE-hard (for arbitrarily large boards) and are PSPACE-complete if restricted to polynomial length games.

  > ...the fact that a problem is PSPACE-complete is even stronger indication that it is intractable than if it were NP-Complete; we could have P=NP even if P is not equal to P-Space [GJ79].

- Chess-like games, such as those defined in MetaGame [Pel92], have primitive operators that are more complicated than those in Hex. Further, conditions once true may become false. Note that in a natural "bit" representation for these games there may be no explicit pieces or squares but boolean conditions representing piece-square pairs.
- Go-like games, such as Othello, with point accumulation have richer reward structures than tic-tac-toe. Although related to Hex, they have an even greater combinatorial explosion of goal states over primitive nodes since they involve goals that are, in essence, sets of hyperedges (nested hyperedges) themselves.
- Bridge-like games have random initial states. Bridge can be represented as 52 x 6 conditions for which player owns which card, whether it has been played, and whether it has been played in the current trick. The communication aspect of bridge can be defined by simply making explicit the reward structure that encourages collaboration and giving each player access to the information on other player's selected operators (corresponding to bids).[1]

[1] The simplicity of the specification, should not obscure the fact that there are rich and complex research problems in getting computers to bid effectively or to learn to do so.

- Chance games, such as Monopoly and backgammon, require stochastic conditions to be set on each move that then limit the legal moves available to the next player. The notion of money in Monopoly may be defined directly as the number of points a player has. Note that unlike previously mentioned games, having enough points may be a necessary precondition for certain operators.

Clearly, this representation is not fully adequate. Certain games, such as the stock market, fall out of this framework; some games are expressed unnaturally and are combinatorially intractable. Simply enumerating all of the terminal sets of conditions in chess or Go is a daunting task for example. Still, by illuminating the mathematical structure of these games, the verbiage accompanying individual domains that make us deal with them singularly rather than holistically has been removed.[2] To address the intractability of the generic hypergraph language, abstract operators, relations and variables need to be considered. MetaGame is an excellent example of generating generic chess-like games at the normal abstract operator level. Barney Pell also has general heuristics that allow reasonable MetaGames to be played without any domain knowledge other than the rules. These heuristics are, however, confined to these chess-like games and do not generally deal with the mathematics of state-space search itself, although it would seem that generalizing them to this larger class of games is possible.

Hoyle [Eps92] is another domain-independent game-playing and learning system that deals at the abstract operator level. It carries with it a rich set of advisors that embody human-supplied heuristics. To the degree that these heuristics are truly domain-independent and operate on a generic as opposed to specific game-representation we can say that Hoyle is an informed learner, but certainly not a blind learner. In the next section we show how a more natural graph-theoretic generic game structure may be developed by taking advantage of the concept of variables and abstract operators, in much the same way as predicate calculus extends propositional logic.

3 Generic Games with Abstract Operators

The generic game representation of the previous section suffers from a potential combinatorial explosion and an "unnaturalness" frequently characteristic of low-level languages. The complexity at the low-level often obscures richer and simpler structures at higher levels of description. To reach a more natural definition of generic games we add the notions of domain objects, static relations, dynamic relations, variables and bindings, but retain the "label-free" framework by omitting the arbitrary names assigned to objects, conditions and operators.[3] The framework, incorporating state space search, is inspired by Peirce's existential graphs [Rob92] and the more modern version "conceptual graphs" developed by John Sowa [Sow83] and our own work in experience-based planning [LK93].

The state space search paradigm breaks problems down into initial conditions, terminal conditions (goals) and operators. It continues to be one of the most pervasive problem

[2]Even precisely defined and well-accepted mathematical concepts are not always as general and as appropriate as they could be, and thus may obscure deeper relationships [Ham72].

[3]Once this is done conceptually the names may be retained as mnemonic aids for humans, but recognized as arbitrary by the machine.

solving models used in AI research.[4] We now show how all state space search problems as they are normally formalized [Kor87] can be viewed as relation-based transformations over hypergraphs, (which are a special case of conceptual graphs). Relations (in the form of predicates and functions) are also the basis for formal first-order logic, the foundation of many of the declarative representations traditionally used in symbolic AI. It is our thesis that logic, graph theory and state-space representation should be tied tightly together. In particular, the notion of static and dynamic relations presented below is original and adds a temporal dimension to traditional model theory [Tar56] by relating logic and search.

The following is a description of the components of a search problem, with running examples taken from tic-tac-toe[5] and Towers of Hanoi on three disks:[6]

- *Each domain has a finite set of domain objects.* In tic-tac-toe the objects will be squares {S1,S2,S3,S4,S5,S6,S7,S8,S9} and pieces {X,O,B (for blank)}. In Towers of Hanoi, the objects are Pegs{P1,P2,P3} and Disks {D1,D2,D3}.
- *Unary, binary and higher relations may be defined on these objects.* An n-ary relation is a set of n-tuples of domain objects. In the finite search domains, rather than defining types explicitly we shall simply note that they are implicitly defined as the set of objects that occur in any single field (attribute) of a relation. At the implementation level, relations that are symmetric or transitive may be abbreviated by specifying a kernel set of tuples and then giving the mathematical properties from which the remaining tuples can be inferred. Other abbreviations and computation of relations are possible, such as finding adjacent squares on the chessboard through calculation based on Cartesian coordinates. We divide relations into two classes: *static relations* are those whose definitions (tuples) remain constant for a given game, and *dynamic relations* are those whose content can change from state to state. A frequent use of static relations is in defining board topology (adjacency of squares). Most domains have a dynamic relation corresponding to ON to say which pieces are on which squares.

  In tic-tac-toe, we define a THREE-IN-A-ROW static relation corresponding to winning sets of squares: {(S1, S2, S3), (S4, S5, S6),...} and the ON dynamic relation initialized as {(B,S1),(B,S2) etc.}. In Towers of Hanoi there is a static relation SMALLER-THAN initialized to: {(D1,D2),(D2,D3),(D1,D3)}. Towers of Hanoi also has the dynamic relation ON, initialized to: {(D3,P1),(D2,P1),(D1,P1)}. Thus, the initial state of Towers of Hanoi would be viewed as the following hypergraph: 6 nodes labeled P1,P2,P3,D1,D2, and D3 and 3 binary directed hyperedges labeled ON connecting D3,D2, and D1 each to P1. The goal is to transform the graph to a similar one in which P3 has replaced the role of P1.
- *Operators define transformations over states by changing the contents of dynamic relations.* Operators are specified by giving sets of preconditions, additions and

[4]The objective of a state-space search problem is to find a sequence of operators (transformations) that will convert a state that satisfies the initial conditions into one that satisfies the terminal conditions, under various optimality criteria and resource constraints.

[5]Tic-tac-toe is played on a 3x3 board. Two players X and O alternate turns selecting cells; X goes first. The first player to complete a row, column or diagonal wins.

[6]Towers of Hanoi is a single-agent game involving three disks "small", "medium", "large" placed on top of each other in order on the first peg of three. The object is to move the disks one at a time such that at no time is a disk on top of a disk smaller than it on the same peg and such that the disks finish all on the third peg.

deletions. Each of these sets can be viewed as a conceptual graph or hypergraph and their combination is a nested hypergraph.

For Tic-Tac-Toe there is an operator MOVE(piece,square) with precondition ON(B, S1), add condition ON(X,S1) and delete condition ON(B,S1). It is assumed the board is always oriented with X to move. For Towers of Hanoi we have the following MOVE operator:

```
MOVE(d1:disk,p1:peg,p2:peg) =
pre: ON(d1,p1) AND ~(p1=p2)  AND ~((ON(d2,p2) OR ON(d2,p1))
     AND SMALLER_THAN(d2,d1)).
add: ON(d1, p2)
del: ON(d1, p1)
```

As a further example, the SWAP operator which switches a bottom block for a top block in the blocks world [Nil80] and vice versa in a tower of three blocks has this representation:

```
SWAP(block:x, block:y, block:z)=
pre: ON(x,y) AND ON(y,z) AND CLEAR(X)
add: ON(z,y)
     ON(y,x)
     CLEAR(z)
del: ON(x,y)
     ON(y,z)
     CLEAR(x)
```

This representation has an equivalent operator graph involving three variable nodes and six dynamic relation edges. Such a representation of SWAP could apply to thousands of domains that involve this type of swapping. Moves in tile puzzles are an instance, but where one of the variables is replaced by a domain object (the blank tile).

- *States are hypergraphs over domain objects.* As the tuples in the static relations are always the same, it is only necessary to state the dynamic relations when representing a state. Operators directly affect the contents of dynamic relations.
- *An operator is applicable in a given state iff there is a 1-1 mapping in variables of the preconditions of the operator to domain objects, such that all relations specified in the operator are true (or false, if negated) of those domain objects.* The result of applying the operator is to remove from the current state those tuples corresponding to the operator's deletions and add those tuples associated with the operator's additions to the contents of the dynamic relations. Since static relations are constant throughout the problem solving process, those bindings of objects to variables that could ever possibly (constrained by the static relations) satisfy an operator definition, can in principle be computed ahead of time, leaving only the dynamic conditions to be checked. This is effected in the implementation presented in Section 7.
- *Terminal conditions are defined in exactly the same way as preconditions, addition and deletion conditions of operators.* It is automatically assumed that a player having no legal moves is terminal. For tic-tac-toe, we have THREE-IN-A-ROW(s1,s2,s3) AND ON(X,s1) AND ON(X,s2) AND ON(X,s3) as terminal. In Towers of Hanoi, we have ON(D1, P3) AND ON(D2, P3) AND ON(D3, P3) as terminal conditions.
- *Reward conditions are conditions that are coupled with a reward to each player based on the outcome of the game.* For Towers of Hanoi and tic-tac-toe, reward conditions

are the same as terminals and assign a *win* to the player who has just moved.

- *Finally, FLIP is a static binary relation over domain objects used to define symmetries, so that a game can be encoded from one player's perspective only.* For tic-tac-toe FLIP is: {(X,O),(O,X),(B,B),(S1,S1),(S2,S2)...}. Towers of Hanoi, as a single-agent game, does not require a FLIP operator. Similar transformations were introduced in MetaGame.
- *In summary: An abstract game or search problem is defined as a finite set of domain objects, and finite sets of static relations, dynamic relations, operators, terminal conditions and reward conditions. Finally, for convenience in encoding, we define a symmetry condition known as FLIP.*

Thus, many single and multi-agent search problems can be viewed as games of (hyper)graph-to-graph transformation. This conclusion is not surprising, given that conceptual graphs and other semantic network schemes have been shown to carry the same expressive power as first-order logic. *The conclusion is significant, however, in that it suggests the potential for graph-theoretic analysis of the rules of a domain and ensuing experience for uncovering powerful heuristics and decision-making strategies* [LK93, LS93]. It also suggests that state-space search can be monitored in a uniform manner; this topic is discussed in Section 7.

3.1 Evaluation of the Graph-theoretic Representation Scheme

The hypergraph representation scheme presented above does not include facilities for inference over static and dynamic relations. Such a facility is available in conceptual graph theory and will be available shortly in the Peirce conceptual graphs workbench [EL92, Gai93] in which our learning system is implemented.

An important step in showing the generality of the graph-theoretic representation is to show that games generated from the MetaGame generator can fit this structure. Although all the implications of manipulating such a structure are yet unknown, initial results show great potential. For example, any macro-operator that is executable in one search domain will work in any other (single-agent) domain that carries that operator structure. With such a domain-independent graph-theoretic representation we have been able to show that the entire TWEAK planning system can be reduced to 5 abstract operators and a simple control structure, and that Roach's robot problem and Sussman's anomaly can be solved using the same database of domain-abstracted operators as macros [LK93].

Using the variable-free, graph-theoretic definition of problems it is also possible to define a hierarchy of single-agent problems by "easier than" using sub-hypergraph-isomorphism. Informally, Problem A is easier-than Problem B if there is a 1-1 mapping of bits in problem A to bits in problem B and a mapping, not necessarily 1-1, from operators in B to less-restricted ones in B. An operator X1 is less-restricted than another operator X2 under a bit mapping M, iff every state-to-state transformation by X2 can also be accomplished by X1 under the mapping M.

Knowing that problem A is easier than B becomes a good source of heuristics. A solution path between two states in B is also a solution path to corresponding states in A, though perhaps not optimal. Hence the length of such a path becomes an upperbound on the optimal solution length in A. Likewise the length of a solution path in A becomes a lower bound on the length of an optimal solution path in B and thus conforms to the traditional notion of admissible heuristic [Nil80]. Finally, the hierarchy over problems by easier-than is

in conformance with the hierarchies generated dynamically during abstract planning [Kor88, Sac74].

4 Extending Tic-tac-toe and Other Games to All Hypergraphs.

To further illustrate the unifying power of studying games based on their mathematical structure let's take a closer look at the tic-tac-toe-like games. Such games reduce to the following basic hypergraph game.[7] Given a hypergraph, players alternate selecting nodes and the first player who owns all nodes in any given hyperedge wins. We shall assume in the discussion below that there are only 2 agents. For example, 3x3 tic-tac-toe has a graph of nine nodes and 8 hyperedges. 3x3x3 has 27 nodes and 48 hyperedges. 4x4x4x4 (Qubic) has 64 nodes and 84 hyperedges. The discussion here builds directly on previous work on forks [Eps90], GoMoku and Qubic [All92], hopefully putting that work in proper perspective.

The hypergraph representation makes symmetries fully realizable through graph isomorphism. This mathematical representation also lends itself to reasonable heuristics. For example, the quality of a node choice is proportional to the number of "live edges" it is involved in and inversely proportional to the number of nodes remaining in each of those edges. Through the use of subgraph analysis more precise heuristics can be developed [Eps90].

4.1 Reductions in the Basic Tic-tac-toe Hypergraph Game

It is useful to become familiar with reductions that preserve game-theoretic value in basic hypergraph games, since such reductions may very well not have been apparent from the traditional state representation. After each move, a hypergraph may be translated to a smaller but equivalent representation to calculate the value of a state assuming optimal play. Given the owner of each edge, and which nodes have been played, edges with nodes owned by both players may be removed. Edges involving exactly the same nodes can be reduced to a single-edge that preserves ownership if all owners are the same, or carries no owner if both agents own such an edge. Similarly, if ownership is the same, edges that are subsumed by other edges are removed. So with each move:

1. Remove each edge incident to the move but owned by another player.
2. Remove the node itself and set the ownership of all other affected edges to the player who has moved.
3. If any edge now has zero nodes, the player who moved wins. If no more edges remain, the game is a draw.
4. Remove all but one out of a set of duplicate edges and update ownership as described above. Remove any edge that is a superset of another and carries the same ownership.

4.2 Other Basic Hypergraph Games

We have already shown how a game like tic-tac-toe may be played on *any* hypergraph Other popular games may be generalized to all basic hypergraphs. These generalized games may require more complex winning strategies than their smaller, fixed-size counterparts. Reductions such as those above for tic-tac-toe are possible in most hypergraph games and lead to more efficient and accurate reasoning.

[7] The hypergraphs in this section are basic, as opposed to the nested ones used in previous sections.

The birthday table or round table game [Vaj92] is described as follows:

> We will assume two players, let us call them Left and Right. Left will seat all the boys and Right will seat all the girls around a circular table with 15 seats. Assuming an unlimited supply of children of both sexes, both players will alternate in their 'moves' (seatings). To preserve decorum no child may be seated next to another of the opposite sex. Whoever is first unable to seat a child loses and will be left to cope with the angry parents.

A generalized birthday game can be played on any hypergraph of n nodes and m edges where each edge has one node designated as the center. With each move to a node, a player gains that node and ownership of all hyperedges of which that node is a center. A player is not allowed to move to a hyperedge that is owned by the other player. The first player that has no legal moves loses. The specific 15 seat birthday game above would be played on a hypergraph of 15 nodes and 15 hyperedges, with each node as a center of a 3-node edge. The round table birthday games have a simple optimal strategy: if the number of seats is even the second player is guaranteed a win by always moving to the seat exactly opposite the last move of the first player. If the number of seats is odd, the first player wins by moving anywhere and then invoking the even number of seats strategy, as if the first player went second.

Another popular single-agent game is known as Merlin's magic square [Vaj92]. This solitaire game, often played on a 3x3 grid can be be generalized to any hypergraph. All nodes start with parity 0. Each move changes the parity of all nodes in a given hyperedge, i.e., 0 becomes 1 and 1 becomes 0. The goal is to design a sequence of moves that changes all nodes to parity 1. Since the order of moves does not matter (all moves are legal at each step) most of these games can be solved directly using linear programming or Gaussian elimination. It is interesting to contrast the Merlin games to other search problems, where move order does matter and applying linear programming then becomes very expensive or impossible.

Finally, Nim [Vaj92] can also be extended for play on any hypergraph. Players alternate taking sets of nodes from the hypergraph, but such that each set is contained within a given hyperedge. The player who takes the last node wins. Most nim games are played on hypergraphs in which all edges are disjoint. In these disconnected versions a simple winning strategy based on binary arithmetic exists. Does this strategy extend to connected hypergraphs as well? In addition to these games, many NP-complete problems [GJ79] such as "set covering" can be formulated to take place on basic hypergraphs.

The characteristics of the basic hypergraph games are much simpler than the generic nested hypergraphs of the previous section, which represent all state-space search problems, but their analysis may lead to a better understanding of the more general case. For instance it is desirable to analyze the chess-like games. These games have the following additional features:

1. Operators are more complex because they have multiple preconditions and postconditions.
2. Conditions can change non-monotonically; a condition which is true can become false and vice versa.

Chess-like games are examples of perfect information, two-agent search problems. It is the complexity of these games and the lack of detailed analytic understanding that has led us to suggest the INFORMED LEARNING and BLIND LEARNING subprojects.

In the following sections we discuss our group's approach to blind and informed learning with graph analysis and reinforcement learning. Many other learning approaches may prove equally viable such as: neural networks [RM86], genetic algorithms [Hol75], constraint satisfaction [RK91], inductive logic programming, and explanation-based generalization [FD89, KM90, RN94]. Most of these methods, however, must be extended to the general case of all single and double-agent search problems, just as MorphII generalizes Morph.

5 Review of Original Morph Model

Morph is an application of our APS (Adaptive-Predictive Search) method for improving search with experience. In APS, knowledge is stored as pattern-weight pairs (pws) [LF4a], where patterns represented by conceptual graphs are boolean predicates over states, and weights are estimates of the expected distance of states satisfying the pattern from a goal state. Starting from a virtually empty database, pws are learned from search experiences using a combination of learning techniques: temporal-difference learning, weight updating, and pattern creation/deletion. Patterns are stored in a partially-ordered hierarchy by more-general-than to facilitate efficient associative recall. Each state's evaluation is formed as a function of the weights of the most specific stored patterns that apply to that state. Ideally, an APS system should converge to a database that serves as a reliable evaluation function using 1-ply lookahead. In practice, APS learning agents couple with a guided search based on previous experience. In addition to weights, other statistics such as "number of uses" or variance may be stored with patterns to be used in determining their importance and whether they should be maintained by the system. APS systems are similar to genetic classifier systems, except that structural patterns are used, no fitness function is available beyond the outcome of a given search, and the pattern representation and creation mechanisms exploit domain-dependent symbolic knowledge.

To use APS in a given domain, a pattern language and pattern addition strategies must be added for that domain. Thus, APS is applied to chess in Morph by supplying a graph pattern language that depicts attacking and defending relationships between pieces and other pieces or squares. Nodes are labeled with pieces types, e.g., white bishop, black king etc. or full and partial square designations, e.g., e2, d5, on-rank-3, on-f-file, and directed edges are labeled (direct, discovered, or indirect). To evaluate a position, the position is translated into a directed graph using the pattern representation language and then matched against a database of pws representing potential subgraphs of the given graph. A typical position might use a weighted average of as many as 50 matched subpatterns in its evaluation. Starting from an empty database, learning patterns of this type and an additional pattern type that reflects material for each side, with 1-ply search Morph learns the relative values of the chess pieces, defeats human chess novices, and draws its Gnuchess trainer regularly.

Morph obviously benefits from a well-chosen pattern representation language. To continue toward the blueprint objective of a domain-independent learner, more responsibility for pattern-creation must be placed on the learning system. It is this desire for generality and deeper understanding that led us to the view of chess as an instance of a game of abstract mathematical relations and to MorphII. It is hoped that an understanding of how Morph has been adapted to the game of abstract mathematical relations may lend insight on how other systems may be generalized as well.

6 Morph II: Improving on Morph.

Once we understand that search problems are games of graph-to-graph transformations (as described in Section 2), we can also understand how knowledge of the existence of a subgraph of a graph representing the current state may carry predictive value of the outcome of the game. Inherent in such a graph are the potentialities of operators that can be applied (or prevented) and a relationship to terminal conditions and goals of the game. Our current understanding of state-space search does not enable us to assess the value as expected outcome of such a subgraph directly. Instead it may be learned statistically, from experience. In experiments, Morph is often limited to searching 1-ply ahead to force us to focus on issues associated with heuristic construction and development.[8]

6.1 Weaknesses in Morph

If the the weights of subgraphs can be approximated accurately, why is Morph not a better chess player?.

1. *Inappropriate mechanisms for combining the values recommended by each of the individual subgraphs.* The original hypothesis was that the pattern values could be combined numerically, independent of which patterns have produced these values and the relationship between these patterns. We now believe that this hypothesis is false, if we are to produce a strong heuristic. The Manhattan Distance for tile puzzles suffers similar shortcomings. In fact, experiments we have conducted in which one attempts to combine the values of higher-level patterns (involving several tiles in the 8-puzzle), without knowledge of the underlying patterns but *having their correct values* (as expected distance to the goal for states having the pattern) have also proved unpromising. These experiments were done by hand rather than using statistical methods to form the combination equation as described below, but still demonstrate the difficulty of using even perfect knowledge about a subset of patterns in a complex domain.
2. *Specificity of the graphs in the chess system.* Although the graphs are reasonable features, the system lacks the ability to have information learned about one graph directly influence the values of other, similar graphs. This leads to many learning inefficiencies, especially considering that some specific graphs may be only seen a few times in the system's career and that this may provide insufficient information to give them accurate values.

Fortunately, these difficulties can be addressed. The original hypothesis underlying the Morph design is that *knowledge of the relationships between objects must be exploited.* This hypothesis has been borne out by the moderate success Morph has had using its graphs. The hypothesis must simply be carried further: the two difficulties above both refer to lack of exploitation of the relationships among the graphs themselves. In short, a scheme must be developed that allows graphs to influence and benefit from the weights of similar graphs.

To proceed further, the following mathematical understanding is required: Morph's graphs are equivalent to a collection of dynamic binary relations that may occur in a position, e.g., the direct attack relation, indirect attack relation. Thus, the same relations

[8] In actuality, the restriction to 1-ply search is one of degree rather than kind. Testing for Morph's patterns requires the equivalent of 1 or 2 ply of search.

that are used to describe the rules of a domain for a monitor may be used as the basis for learning in that domain as well.

6.2 The GLM and the Reinforcement Hierarchy

To make Morph more general and to address the two learning limitations above we developed the notion of a generic learning module (GLM) with the following properties:

- It contains n subsystems.
- It estimates the value of a state S, by first consulting its subsystems (themselves generic learning modules) to get their estimates of the value of S. It combines their values numerically (perhaps non-linearly) using an equation that it has learned. The combining rule exploits the past accuracy of the subsystems.
- The module receives feedback for its predictions from a higher-level system.
- The module learns by modifying its combination equation.
- A primitive GLM is a GLM in with no subsystems; it simply attempts to predict the feedback value using a dynamic constant function.

To focus this discussion on the important high-level learning issues defined above, we may ignore the learning method used by the GLM. Conceptually, any function learning method will do. *We focus on the identification of the subsystems themselves and towards their interaction.* The notion of a hierarchy of control modules is consistent with the hierarchical general systems theory of Mesarovic [MMT70, Kli85]. Our contributions include the generation of the systems hierarchy from the declarative specifications of the domain by exploiting the isomorphism of the terms "logical predicate" and mathematical relation, the formation of higher-order controllers by combining relations, and the connection between standard AI concepts such as state-space search, machine learning algorithm, and TD-learning with graph theory and systems theory. The richness of these connections has been achieved through the exploitation of the appropriate mathematical abstractions and thus, lends some support to the emphasis on mathematics in the blueprint.

The individual learning modules form a partially-ordered hierarchy based on the "subsystem" relation. The top level modules of the hierarchy receive feedback for individual states in a state sequence using the same temporal difference (TD) learning [Sut88] mechanism as in Morph.[9] These modules in turn send feedback to their subsystems and so on, bottoming out in primitive GLMs. During play, higher-level systems compose their predictions by combining the recommendations of lower-level systems.

As games become more complex, the complexity of each individual learning module and the hierarchy of learning modules will need to increase as well. For simple games, such as 3x3 tic-tac-toe it is possible to write an evaluation function for optimal play that is simply a linear combination of the values of objects (X,O, or B) on individual squares. This corresponds to a 2-level hierarchy, with nine modules on the lower level and one module on the top. However, for a game such as chess, an evaluation function based directly on a linear combination of the values of objects on the 64 squares will not perform well, because it is the relationship between objects and not individual objects that is critical. To simply expect a non-linear evaluation function to succeed begs the question, "where do the complex interactions it needs to consider come from?". Also, to *efficiently* learn the proper

[9] After a game is completed, TD learning is used to assign new evaluations to all states that occurred during the game, given the new information provided by the outcome of the game.

coefficients for this complex function, knowledge must be transferred and shared between terms. Otherwise, for example, all instances of a black pawn attacking the white queen would have to be learned separately,

6.3 Generating the Reinforcement Hierarchy From the Rules of a Domain.

We use the rules themselves to dictate the structure of the system hierarchy. Our hypothesis is that for most domains the relations given in the initial description of a domain are enough to support the patterns necessary for learning. A GLM is created corresponding to each relation. GLMs are stored hierarchically based on dependencies. For example, in the tic-tac-toe definition of Section 2 the hierarchy has a middle-level module corresponding to ON , and two higher-level modules corresponding to MOVE and TERMINAL. Towers of Hanoi has a similar hierarchy. The primitive low-level GLMs in these hierarchies correspond to the weights of individual tuples that may be stored in the individual modules. Each tuple of a higher-order module provides feedback to the tuples of lower-level modules that they depend on. For example, the value of MOVE(D2,P2,P3) in the Towers of Hanoi is built as a combination of the values of ON(D2,P2), ON(D1,P2) and ON(D1,P3).[10]

To evaluate a state, the monitor proceeds as follows:

1. The monitor calculates incrementally, based on the previous state and the current move, which tuples match and do not match a given state.
2. The weights of the lowest level tuples are propagated up the hierarchy and combined to form the values of higher level tuples.
3. Each relation combines the values of its tuples and lower-level relations to produce recommendations to higher-level relations.
4. The values of the highest-level relations are combined to produce the evaluation of the state itself.

The weight combinations are based on the weights themselves as well as the previous accuracy of the subsystems.

This design deals directly with the first weakness in Morph discussed above. Now each module has its own combination function, so that weight combination is localized and specific to the patterns under consideration. In the new framework two weights may be combined differently by the problem solving system based on their location within the GLM hierarchy.

The second weakness, inability of patterns to share information, has also been addressed. Patterns with the same structure all occur in the same relational table. Thus the weights of these patterns may be shared and generalized across the contents of the table. Further, rather than patterns being added one at a time, pattern skeletons represented by new relational tables may be inserted into the hierarchy and thus many new patterns processed simultaneously. The exact nature of these relation table insertions is a topic for further work. Finally, higher-level patterns are influenced directly by the values of low-level patterns.

[10] In MorphII, the weights of the low-level tuples learned in the primitive GLMs use a learning rule that takes into account the average of training values seen over time, the most recent training value, and the accuracy of previous predictions.

An important improvement is the use of relational tables as the GLMs in the hierarchy. Most search and game domains can be represented using a small finite number of domain objects. The relational tables may therefore be stored as boolean matrices, where each bit corresponds to an individual tuple. Thus, the expensive graph matching of the original Morph system has been replaced by simple bit matching operations (see Section 8). The boolean matrices also potentially allow for the application and mathematical analysis of efficient learning schemes based on linear algebra for the GLM.

7 Blind Learning

Clearly, blind learning is more difficult than informed learning. For example, in MorphII with blind learning there would be no domain definition to guide the construction of the learning hierarchy. Not only do the GLMs have to learn as before, the question of which generic modules are formed and their interrelationship becomes a critical issue. Further, recognizing redundancy so it can be exploited becomes a difficult matter. Not only does an informed learner start from a better place, it should be able to classify its experience more accurately.

Despite the difficulties, we feel that it is important for AI researchers to study blind learning for several reasons:

1. In many practical domains one is not given access to any declarative description of the rules.
2. The additional difficulties in blind learning give us a better appreciation for the information supplied in rules.
3. Blind learning forces the learning system to make use of all available information. In terms of MorphII, the system must develop a relational structure for a given domain that works.

In our proposed environment for blind learning systems, the agent is given privy to the following information and no more:

- The current state of the board as a finite-length vector of boolean conditions. This bit description is *exactly* the information a legal move generator would need to generate the correct set of legal moves in a given state. Beyond this, no interpretation is given to the bits and they may be placed in any order. Just as informed learning may operate in domains with rule encodings ranging from malicious to benevolent, blind learning must operate with state encodings of varying quality.
- A set of rules , where the legal moves and rewards from any set of conditions is fixed and the same for each player under some inversion or symmetry of the board or domain objects.
- The states resulting from each of its legal moves at any given point in the game.
- The reinforcement for the game in terms of some reward that the agent is trying to maximize.

Here is how tic-tac-toe might appear to the blind learning agent. The board is 18 bits: 2 bits for each square where 00 is empty, 01 is X, 10 is O, and 11 is an unused code. To make things more obscure, one can interpret can consider the board as representing a base 3 number, so that each state could now be represented as a 14-bit binary number. Thus, even a straightforward game like tic-tac-toe could quickly disguise any resemblance to the standard board and the form of the rules of play. Such difficulties are intended. We believe

that methods must be developed to produce a general learner, one capable of adapting given adequate experience to any game-playing environment.

7.1 Blind Learning in MorphII

A system that can exploit not only relations between objects, but also higher-order and analogous relations on them will surpass in performance most learning systems currently in existence. Most systems do not exploit graph-isomorphism and higher-order morphisms between structures despite the fact that such relationships are at the core of the structure of those domains. For example, many implemented learners have difficulty even learning the simple concept "any three consecutive bits are 1" [Hun94], apparently because the simple adjacency relationship between bits is not being exploited or ternary relations are not considered.

Our goal is to build a blind learner that constructs and exploits a reinforcement hierarchy, just as in informed learning. The study of how blind learning might do this also leads to insights on how the informed hierarchy may dynamically modify itself by adding new relations.

Initially, the blind learner starts out with one low-level "state" GLM. This module views the state as a binary relation, with potentially matching tuples represented by its bits. As before, the weights of the individual bits are stored and learned in primitive GLMs.

The system uses its experience with the values of bits and their cross-correlation to create higher-order and, hopefully, more useful relations. The system learns and stores the following information due to pairing its bits.

- The frequency of two bit values (00,01,10,11) occurring together.
- The performance weights associated with the pairs of values.
- A linear or non-linear function that predicts with least squared error the weight of a pair of values given the weights of individual values.

With an n-bit representation, there are 4*n*(n-1) possibilities to be studied. Under computational constraints the pairing operations might be restricted to those bits with the most extreme or predictive weights.

The system then creates new relations based on the following:

- A set of bits that never occur together, i.e., at most one bit is on at a given time is called a *variable*. For example, variables might correspond to the pieces or squares in board games.
- Pairs of bits that have similar values and similar weight combination rules.
- Sets of bits whose parity changes by a single operator application. These sets of bits give topological information about the structure of the search space.

Currently, the system uses fuzzy graph matching and clustering [Wat85] to form higher-level relations and the associated GLMs. For example, two variables whose bit pairs (one bit from each variable) have similar combining rules are then combined to form a new binary relation. We believe that through more thorough mathematical analysis, it may be possible to put forth probabilistic arguments as to which relations should be created and which should not be. Currently, the relation GLMs are monitored to determine which are the most reliable predictors; those that are the best are retained. Higher-order relations may then treat the absence or presence of tuples in an analagous manner to the way bits were treated at the lowest level by the blind learner to create yet higher relations.

8 Exploiting Analogous Relationships in Blind Learning

To gain further insight and understanding into the blind learning method proposed in the previous section, consider the combinatorial explosion associated with chess, for example, how hard it is to learn not to place the white queen where a black pawn can capture, including places it has not yet been placed. Here we assume no knowledge of piece, square, pawn, or queen. has been supplied.

Assume that the blind learning system has observed several instances of the pawn-can-take-queen relationship (relationships between two uninterpreted piece-square conditions due to this relationship) and then encounters a new instance, for example, a state where the bit for ON(WQ,e4) is 1 and the bit for ON(BP,d5) is 1. If this instance, being a capture, occurred before there would already be a specific tuple stored for this pattern. However, if this instance has not been encountered before a generalization from similar instances is required. We propose the following scenario for how this chess generalization, and many others, might be learned:

1. The values of individual bits are learned through sufficient experience. Let X equal the value learned for ON(WQ,e4) and Y equal the value learned for ON(bp,d5).
2. Variables are discovered from the information about which bits occur together. Let V1 be the variable corresponding to the white queen, V2 to e4 and V3 to d5.
3. Binary relations are formed from pairs of variables whose pairs of bits have similar combining rules. The combining rule between two variables is a function that, given the values of the bits which are on in the two variables, produces the value for their conjunction. A binary relation that includes tuples from which a black piece on e4 attacks a white piece on d5 can be discovered as a result of the value of most attack tuples being worth much less to white than the values of the components of the tuples. Substituting the values of X and Y in this rule would give an approximately correct result, especially if the value of the combination is inversely proportional to the value of X.

Relations such as the "e4-attacks-d5" relation above might be collected together to form a general "attacks" relation or specialized further to form a "wp-e4-attacks-d5" relation. These concepts are very similar to Morph's original human supplied edge types. Just like the edge labels in Morph, the learning of new relations facilitates the analogical reasoning process by making precise the combining rule for its components. Once made explicit, these combining rules can then themselves be matched to find higher-level patterns.

The assumption is that games and search problems of interest have regular underlying structure. To the degree that this assumption of regularity is false, the learning task will be more difficult and more false inferences will be made before the proper structure can be learned, if it can be learned at all. If no connection between these squares has ever been noticed an agent ignorant of board topology and the rules, would find it virtually impossible to recognize the interaction, unless it were inferable indirectly from other patterns.

9 MorphII: Domain-Independent Games Environment in C++

The blind and informed learning testbeds described above are available as part of the public domain software known as the Peirce Conceptual Graphs Workbench [EL92]. The learning system in Peirce, known as MorphII, accepts the rules of a single-agent or a multi-agent state-space search domain, translates them into conceptual graphs, and then monitors

games and learning via a "super-referee." Thanks to object-oriented C++ code, domain independence is maintained through polymorphism of domain objects, and independent C++ modules can be coded and tested and modifications can be made to our initial implementations for blind learning and informed learning algorithms. Declarative rule sets for a number of games are available, including chess, Towers of Hanoi, tic-tac-toe, Hexpawn3, Hexpawn6, the 8-puzzle, Nim and the birthday party game. It is also possible to test the power of different search algorithms and human-supplied heuristics in these domains. This section discusses our implementation methodology, and summarizes performance results.

9.1 Monitoring State-space Search Incrementally Using UDS

Our primary discovery is that a relational hierarchy can efficiently monitor and enforce the rules of a given domain, while it also serves as the basis for the hierarchical reinforcement learner described in Section 4. We call the relational hierarchy and the algorithms that operate over it UDS, for "Universal Data Structure" [Lev94].

The "Universal" in UDS refers to an effective monitor and executor of the specifications for any given state space search domain. Due to the relation-based perspective of UDS, the following ideas from the RETE algorithm [For82, Mir87] can be exploited with little adjustment to the relational hierarchy defined above.

- *The firing of an individual operator does not affect the current state radically.*
- *If an operator did not match in the previous cycle, it most likely will not match in the current cycle either.*
- *On each cycle we should only try to rematch operators that could have been affected by the previous operator application.*
- *Different operators may share a large amount of the same structure. Thus, separate conditions of operators should only be matched once per cycle.*
- *Variable bindings from cycle to cycle remain relatively consistent.*

UDS monitors search problems as follows. The hierarchy is used to represent dynamic relations. Specific relations that are true are stored beneath the schema declarations for the relations as specifications. A schema declaration and its tuples are equivalent to a table in a traditional relational database and are called a *table* here. The preconditions of operators are stored using the graph hierarchy. Repeated parts of operators are only represented once in the relation hierarchy. Figure 9.1 depicts the initial UDS network for monitoring the Towers of Hanoi. UDS differs from standard RETE implementations; UDS exploits the relation-based representation of conceptual graphs to extend the types of patterns that can be matched and speed of their matching. However, UDS naturally supports RETE as well as a variety of other data manipulation methods appropriate to relational databases, conceptual graphs and semantic networks.

Those dynamic relations that do not depend on any other relations in their definition are known as *primitive dynamic relations*.[11] The post-conditions of operators work directly on the primitive dynamic relations through pointers to add or delete tuples from their contents. Static relations are compiled away at network generation time because the set of tuples that satisfy them remains constant. Conceptual graphs representing the preconditions of operators are only re-matched if the content of one of their composing relations changes. Only that part of the conceptual graph affected by the change need be re-matched.

[11] The primitive dynamic relations form the low level GLMs that sit directly above the primitive GLMs (for weights) in the reinforcement hierarchy.

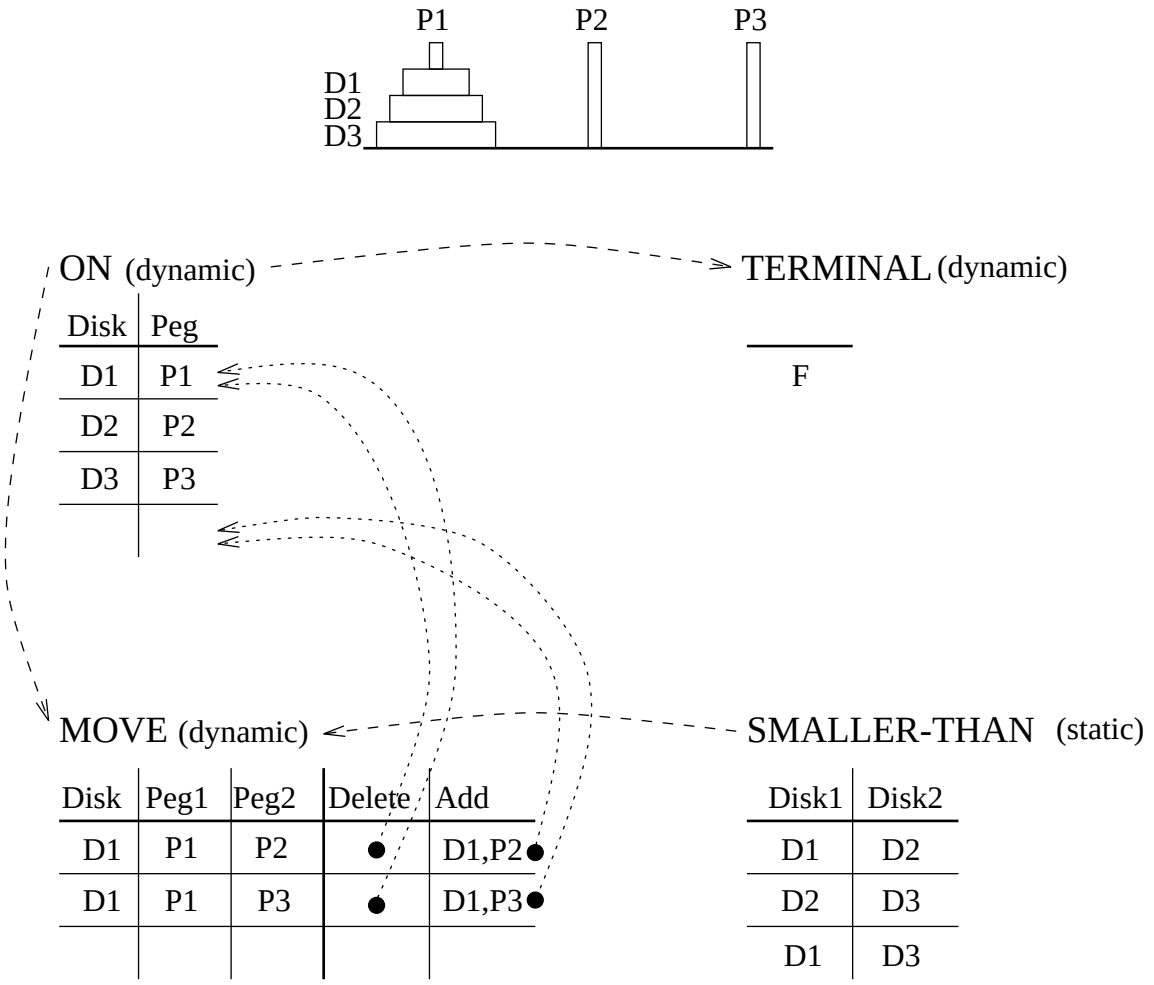

Figure 9.1: Initial state of UDS network for monitoring Towers of Hanoi.

9.2 What Happens After an Operator is Selected

1. A selected operator corresponds to a tuple in one of UDS's operator tables, for example, MOVE(D1,P1,P3) in Towers of Hanoi.
2. This tuple is bound to the given variable-arguments of that operator schema and lists of add and delete tuples are built based on the add and delete conditions of that schema. In our example, ON(D1,P1) will be deleted and ON(D1,P3) will be added.
3. These lists of tuples are added and deleted immediately from the appropriate primitive dynamic relation tables in the net, and these new changes are propagated up the net.
4. Each of the created tuples in the add/delete lists is iterated through for each table in the net that it may directly affect, changing the truth values of tuples in that table. If one of the added or deleted tuples matches a pre-condition of a table, then a "re-join" procedure is called to update the tuples currently being stored in the table. Those tables not affected by the new add/delete tuples are ignored. Thus, in our example, the firing of Move(D1,P1,D3) removes (D1,P1,P2) and (D1,P1,D3) from the Move table and adds (D1,P2,P1) and (D2,P1,P3) to the Move table.

Further details on the implementation, including how join and merge work, can be found in [Lev94].

9.3 Performance Results

With this scheme, we have been able to monitor a variety of domains including tic-tac-toe, Towers of Hanoi, 8-puzzle, and Hexpawn[12] at a level of efficiency that is faster (in some cases up to 10 times) than previous programs of ours that had been written specifically for these domains but were not incremental.

UDS has played at various search depths and monitored a variety of search domains compiled into its network from rule definitions as in Section 6. UDS is implemented in C++ on a SUN SparcII with 64 megabytes of main memory. Although domain-independent, its results are reasonable and would likely outperform (for domains with high branching factors) a hard-coded program that does not exploit incrementality.

Table 9.1 presents results from our earlier implementation in which relation tables are stored as sparse matrices. In particular, a relation is stored as a set of tuples where each tuple is identified by its arguments. In the table, "Random vs Random" means that a random agent competes against another random agent; "n Ply" means that one agent searches this deep, evaluating leaf nodes with random numbers and then using minimax for move selection, while the other agent plays randomly with no lookahead. The 500 moves Towers of Hanoi results are with game restarts and the 500 move 8-puzzle results are for 1 game.

Table 9.2 presents results from our most recent implementation in which the multi-dimensional bit array method is used for storing relation tables. Each tuple is an individual bit cell in the matrix. A 1 means the tuple is present; a 0 means the tuple is absent. The subscripts of the cell correspond to the arguments of the tuple. The main advantages of the bit matrix storage scheme is that processing is uniform for all tuples and that row operations on the matrices can exploit the bit-parallel concurrency provided by the logic operations of the workstation. In effect, relational join has been reduced to matrix multiplication. Compared to Table 9.1 there are speedups of a factor of 5 to 81, typically around 25. Towers of Hanoi shows the greatest speedup, and Hexpawn the least. With the exception of Hexpawn, speedups are significantly greater at higher ply than lower ply.

Tables 9.3 and 9.4 summarize the quality of play with blind learning. Towers of Hanoi starts from randomly selected legal initial states. For both tic-tac-toe and Hexpawn, draws are counted as wins for the second player. The statistics presented are for a very basic blind learner that simply learns the values of individual bits (through gradient-descent weight updating) in the uninterpreted state representation, or randomly selected sets of bits in the underlying representation. For Towers of Hanoi, 9 bits are used to represent which disk is on which peg. In both Hexpawn and tic-tac-toe 18 bits are used to represent the state (00 for blank, 01 for O or white pawn, 10 for X or black pawn for each of the 9 positions; 11 is not used). No effort was made to identify and exploit generalized relationships such as three-in-a-row for tic-tac-toe.

[12] Hexpawn is a simple version of chess played on a 3x3 board. Each player has 3 pawns. In the starting configuration each player's pawns are along their respective edge of the board, with the middle rank unoccupied. White moves first and players alternate moves. Pawns move one square forward and capture diagonally as in normal chess. The first player to get get his/her pieces to the opposite side of the board or to capture all the opponent's pieces wins. The game is a draw given optimal play. Hexpawn 6 is Hexpawn but on a 6x6 board where each player has 6 pawns.

	Total	Average	Total	Average
Single–Agent Games	100 Moves		500 Moves	
Towers of Hanoi				
Random vs Random	< 1.0	–	2.0	0.004
1 Ply	3.0	0.03	15.0	0.03
2 Ply	19.0	0.19	102.0	0.20
3 Ply	110.0	1.10	618.0	1.24
8 Puzzle				
Random vs Random	< 1.0	–	2.0	0.004
1 Ply	4.0	0.04	22.0	0.04
2 Ply	42.0	0.42	205.0	0.41
3 Ply	298.0	2.98	1605.0	3.21
Double–Agent Games	100 Games		500 Games	
Hex Pawn				
Random vs Random	2.0	0.02	10.0	0.02
1 Ply	6.0	0.06	30.0	0.06
2 Ply	22.0	0.22	110.0	0.22
3 Ply	80.0	0.80	405.0	0.81
Tic-tac-toe				
Random vs Random	3.0	0.03	15.0	0.03
1 Ply	9.0	0.09	45.0	0.09
2 Ply	55.0	0.55	290.0	0.58
3 Ply	415.0	4.15	2095.0	4.19

Table 9.1: Tuple–List Execution Speeds, in seconds. The amount of time used to make 100 and 500 moves or play 100 or 500 games is depicted at different search depths.

Tables 9.5, 9.6, 9.7, and 9.8 present results for the Generic Learning Module (GLM). Although the GLM is in an informed setting, it does not "analyze" the rules, but instead uses them to form its monitor and reinforcement hierarchy. While future GLMs will add new relations and patterns to their database, for these experiments *only those patterns necessary for monitoring the games were stored.* Table 9.5 shows how performance against a random agent grows with early training for hex pawn, hex pawn 6, and tic-tac-toe. Percentages are given for wins (W), draws (D), and losses (L) and are cumulative over all games played. In those cases where learning does not seem to improve (as in 1-Ply APS versus random in Hex Pawn) very good performance was already achieved in 10 games. We believe that the slight degradation is probably due to learning against a (now) weaker opponent.

Table 9.6 shows how performance grows with longer term training for pennies and NIM.[13] In NIM the raw APS agent did not fare as well as in other domains. This is because the patterns required to monitor NIM do not include the critical feature: the relationship between the stacks. By manually adding a 3-ary dynamic relation that includes the number

[13]In pennies there is a stack of 15 pennies. Players alternate taking 1–3 pennies from the stack. The person who takes the last penny wins. In NIM there are three stacks of 3, 5 and 7 sticks, respectively. Players alternate taking one or more sticks from a given stack; the player who gets the last stick wins. Both of these games are wins for the first player given perfect play.

	Total	Average	Total	Average
Single–Agent Games	100 Moves		500 Moves	
Towers of Hanoi				
Random vs Random	0.02	0.0002	0.14	0.0003
1 Ply	0.12	0.0012	0.49	0.0010
2 Ply	0.49	0.0049	2.15	0.0044
3 Ply	1.41	0.0141	6.61	0.0134
8 Puzzle				
Random vs Random	0.03	0.0003	0.29	0.0005
1 Ply	0.18	0.0018	0.97	0.0019
2 Ply	0.61	0.0061	3.19	0.0063
3 Ply	1.78	0.0178	8.87	0.0178
Double–Agent Games	100 Games		500 Games	
Hex Pawn				
Random vs Random	0.20	0.0020	0.96	0.00193
1 Ply	0.44	0.0044	4.85	0.00447
2 Ply	1.12	0.112	4.85	0.00995
3 Ply	1.96	0.0196	9.80	0.0196
Tic-tac-toe				
Random vs Random	0.18	0.018	0.84	0.0017
1 Ply	0.54	0.0054	2.72	0.0054
2 Ply	1.96	0.0196	9.24	0.01867
3 Ply	8.36	0.836	39.52	0.0798

Table 9.2: Bit–Matrix Execution Speeds, in seconds. The amount of time used to make 100 and 500 moves or play 100 or 500 games is depicted at different search depths.

Towers of Hanoi			
	Lowest	Highest	Average
Number of moves	7	21	11
Games won in $\leq$ 10 moves			77%

Table 9.3: Blind Learner, Single–Agent Games (100 Games) The number of moves required by a blind learner to solve Towers of Hanoi is depicted.

of sticks on each stack the improved performance under NIM 2 was achieved. Future developments will make such feature addition automatic.

Table 9.7 shows performance against a greedy opponent over 1000 games *after* learning during 500 games (no learning occurs during the 1000 games). The greedy agent is a random agent except it recognizes (knows the correct value of) winning and loosing positions. "R-APS" refers to the APS agent trained against the random agent, "G-APS" to the APS agent trained against the greedy agent. Table 9.8 presents the same conditions, but where APS is the second agent to move. Over the coming months we will be extending our results to backgammon, checkers, Othello, and chess and building a GLM that processes the bit matrix relational tables as "images."

Agent		Wins	
1	2	Agent 1	Agent 2
Hex Pawn			
Random	Random	59%	41%
Random	Blind Learner	21%	79%
Blind Learner	Random	67%	33%
Tic-tac-toe			
Random	Random	74%	26%
Random	Blind Learner	44%	56%
Blind Learner	Random	76%	24%

Table 9.4: Blind Learner, Double–Agent Games (500 Games) Blind learning versus a random opponent. Percentage of wins is shown.

10 Conclusion

Eric Baum recently pointed out [Bau93] the inherent potential in of information (mathematical structure) inherent in a declaration of the rules of a given domain:

> The computer science approach has since Shannon basically regarded a game as defined by its game tree. But what makes a game interesting is that it has a low complexity, algorithmically efficient definition apart from the game tree.... Any procedure which only accesses the underlying simplicity of a game in the form of an evaluation function is inherently doing the wrong thing.. .. the main open question is how to go beyond the evaluation function picture of games.

We agree strongly with this insight and in this paper have suggested 4 projects that we believe will lead to a practical exploitation of the underlying structure of game-domains:

Agent		Agent 1								
1	2	10 Games			100 Games			500 Games		
		W	D	L	W	D	L	W	D	L
Hex Pawn										
Random	Random	30	40	30	41	41	18	45	27	28
1-Ply APS	Random	60	40	0	59	29	12	50	40	10
2-Ply APS	Random	60	40	0	59	29	12	51	45	4
Hex Pawn 6										
Random	Random	50	10	40	44	7	49	48	7	45
1-Ply APS	Random	70	10	20	68	4	28	82	4	14
2-Ply APS	Random	50	20	30	77	9	14	87	4	9
Tic-tac-toe										
Random	Random	70	10	20	60	12	28	58	12	30
1-Ply APS	Random	80	10	10	87	5	8	87	5	8
2-Ply APS	Random	60	10	30	89	7	4	93	5	2

Table 9.5: GLM, Double–Agent Board Games (numbers are percentages of wins,losses and draws.) The results of the GLM playing a random opponent at various search depths is depicted.

Agent		Agent 1					
1	2	250 Games		1000 Games		2000 Games	
		W	L	W	L	W	L
Pennies							
1-Ply APS	Random	93	7	95	5	97	3
NIM							
1-Ply APS	Random	72	28	75	25	75	25
NIM 2							
1-Ply APS	Random	84	16	89	11	92	8

Table 9.6: GLM, Double–Agent Stack Games (numbers are percentages of wins, losses and draws. The GLM versus a random opponent for 250, 1000, and 2000 games.

.

1. GENERIC REPRESENTATION: Design a domain-independent mathematical representation of state-space search domains.
2. INFORMED LEARNER: Design a program that performs well in state-space search given just the mathematical definition of a given domain, domain-independent heuristics and experience.
3. BLIND LEARNER: Design a program that performs well on state-space search given just experience, the rewards at the end of a given domain and legal states (as raw bit vectors) to choose from at each move.
4. MONITOR: A programming environment that supports the design and execution of experiments on these topics.

We have shown how individual search and game domains may be viewed as instances of a general graph-theoretic game of abstract mathematical relations, and discussed the relationship of these graphs to relations and first-order logic. We argue that the advantages of taking this perspective are many:

1. An efficient, domain-independent monitor can be developed that exploits the matrix and logic operations of the underlying hardware.
2. A study can be made of the relationship between the structure of the rules (problem definition) of a given domain and the structure of a hierarchical reinforcement learner or neural net to experientially learn the value of states in that domain. In particular, we outline an algorithm to convert from the relation-based representation of a domain to a reinforcement learning network for that domain.
3. The problem definition, monitor and learner for a given domain can be seen as arising together from the inherent structure of that domain, rather than being three separate processes.
4. As a result of this coherence, domains that are similar in structure give rise to similar learners and monitors. In fact, UDS illustrates how the same network data structure that monitors the domain may be the basis of a reasonable learner for that domain.
5. The relationships of parts-and-wholes in the domain definition may be extracted to identify the placement of "Generic Learning Modules" in the reinforcement hierarchy. Thus learning research might be pursued from the higher perspective of the interaction of learning modules rather than as a choice between methods.

Agent		Agent 1		
1	2	W	D	L
Hex Pawn				
random	greedy	42	29	29
greedy	greedy	47	26	27
1-Ply R-APS	greedy	51	37	12
1-Ply G-APS	greedy	43	57	0
2-Ply R-APS	greedy	55	45	0
2-Ply G-APS	greedy	56	44	0
Hex Pawn 6				
random	greedy	35	4	61
greedy	greedy	52	5	43
1-Ply R-APS	greedy	73	3	24
1-Ply G-APS	greedy	77	3	20
Pennies				
random	greedy	26		74
greedy	greedy	50		50
1-Ply R-APS	greedy	100		0
1-Ply G-APS	greedy	100		0
NIM				
random	greedy	37		63
greedy	greedy	52		48
1-Ply R-APS	greedy	21		79
1-Ply G-APS	greedy	20		80
2-Ply R-APS	greedy	73		27
2-Ply G-APS	greedy	67		33
NIM 2				
random	greedy	37		63
greedy	greedy	52		48
1-Ply R-APS	greedy	69		31
2-Ply R-APS	greedy	99		1

Table 9.7: GLM, Double-Agent Games After Learning (numbers are percentages of wins, losses and draws.) The results of tournaments of 100 games between a greedy, random, and APS agents that have been trained for 500 games against greedy or random opponents is depicted. APS agents using 1-ply and 2-ply search are included. APS moves first.

The performance of our matrix-based monitor (as opposed to learner) lends credence to the practicality of the general graph-theoretic view of search domains. The results presented here, are preliminary but encouraging; they demonstrate that reasonably strong reinforcement learners of differing structure can be constructed automatically from a problem description. Further research will study how the reinforcement learner can enhance itself by adding new relational tables to the network. This self-organization, coupled with parameterless (not requiring human tuning) learning rules, will further support our theme of giving the computer maximum responsibility for its learning structure.

Agent		Agent 1		
1	2	W	D	L
Hex Pawn				
greedy	random	50	31	19
greedy	1-Ply R-APS	0	56	44
greedy	1-Ply G-APS	0	58	42
Hex Pawn 6				
greedy	random	65	6	30
greedy	1-Ply R-APS	29	4	67
greedy	1-Ply G-APS	27	6	67
Pennies				
greedy	random	74		26
greedy	1-Ply R-APS	3		97
greedy	1-Ply G-APS	5		95
NIM				
greedy	random	65		35
greedy	1-Ply R-APS	77		23
greedy	1-Ply G-APS	82		18
NIM 2				
greedy	random	65		35
greedy	1-Ply R-APS	28		72
greedy	2-Ply R-APS	3		97

Table 9.8: GLM, Double-Agent Games After Learning (numbers are percentages). The results of tournaments of 100 games between a greedy, random, and APS agents that have been trained for 500 games against greedy or random opponents is depicted. APS agents using 1-ply and 2-ply search are included. APS moves second.

As effort by various research groups on the projects of the blueprint proceeds we hope that computer science and Artificial Intelligence will be appreciated as important branches of mathematics dealing with optimal problem solving under resource constraints, and that computer game-playing will have played an important role in that development.

Acknowledgments

Barney Pell, and the anonymous reviewers provided useful criticism and encouragement during the writing of the paper. John Amenta developed much of the MorphII software and obtained performance results. Yuxia Zhang assisted in developing the declarative definition of games and implemented the Generic Learning Module. James D. Roberts helped with the figures and a late draft of this paper. Susan Epstein provided exhaustive stylistic and editorial corrections to the writing and other useful comments. Radhika Grover proofread a late draft.

References

[All92] V. Allis. Qubic solved again. In J.V.D. Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3*, pages 192–204. Ellis Horwood, 1992.

[Bau93] E.B. Baum. How a bayesian approaches games like chess. In *Games: Planning and Learning, Proceedings of the AAAI Fall Symposium*, Menlo Park, CA, October 1993. AAAI Press.

[BCG82] E.R. Berlekamp, J.H. Conway, and R.K. Guy. *Winning Ways for Your Mathematical Plays*. Academic Press, New York, 1982.

[Bea80] D.F. Beal. An analysis of minimax. In M.R.B. Clarke, editor, *Advances in Computer Chess 2*, pages 103–109. Edinburgh University Press, Edinburgh, Scotland, 1980.

[Ber79] Hans Berliner. The B* tree search algorithm: A best first proof procedure. *Artificial Intelligence*, 12(1):23–40, 1979.

[Bet81] A. D. Bethke. *Genetic Algorithms as Function Optimizers*. PhD thesis, University of Michigan, DAI 41(9), 3503B, 1981.

[BGH89] L.B. Booker, D.E. Goldberg, and J.H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.

[BS94] D. Beal and M. C. Smith. Random evaluations in chess. *International Computer Chess Association Journal*, 17(1):3–9, March 1994.

[BW94] E. Berlekamp and D. Wolfe. *Mathematical GO Endgames: Nightmares for the Professional Go Player*. A.K. Peters, Wellesley, Massachusetts, 1994.

[EL92] Gerard Ellis and Robert Levinson, editors. *Proceedings of the First International Workshop on PEIRCE: A Conceptual Graphs Workbench*. Department of Computer Science, The University of Queensland, 1992.

[EN69] G. W. Ernst and A. Newell. *GPS: A Case Study in Generality and Problem-Solving*. Academic Press, New York, 1969.

[Eps90] S.L. Epstein. Learning plans for competitive domains. In *Proceedings of the 7th International Conference on Machine Learning*, pages 190–197, Austin, TX., 1990. Morgan Kaufmann.

[Eps92] S.L. Epstein. Prior knowledge strengthens learning to control search in weak theory domains. *International Journal of Intelligent Systems*, 7:547–586, 1992.

[FD89] N. S. Flann and T. G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187–226, 1989.

[For82] C.L. Forgy. Rete: A fast algorithm for the many pattern/many object patern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[Gai93] B. R. Gaines. Representation, discourse, logic and truth: Situated knowledge technology. In *Conceptual Graphs for Knowledge Representation*, number 699 in Lecture Notes in Computer Science, pages 36–63. Springer-Verlag, 1993.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, Murray-Hill, 1979.

[GL94] J. Gould and R. Levinson. Experience-based adaptive search. In R. Michalski and G. Tecuci, editors, *Machine Learning:A Multi-Strategy Approach*, volume 4, pages 579–604. Morgan Kauffman, 1994.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Welsley, Reading, MA, 1989.

[Ham72] P. C. Hammer. Mathematics and systems theory. In G. Klir, editor, *Trends in General Systems Theory*, pages 408–433. Wiley and Sons, New York, 1972.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[Hun94] L. Hunter. Disjunctive concept learning. December 1994. A communication in the moderated electronic Machine Learning List.

[Jan90] P. Jansen. Problematic positions and speculative play. In T. A. Marsland and J. Schaeffer, editors, *Computer, Chess and Cognition*, chapter 10, pages 169–181. Springer-Verlag, 1990.

[Kai90] H. Kaindl. Tree searching algorithms. In A.T. Marsland and J. Schaeffer, editors, *Computers, Chess and Cognition*, pages 133–158. Springer-Verlag, 1990.

[Kli85] G. J. Klir. *Architecture of Systems Problem-Solving*. Plenum Publishing, New York, 1985.

[KM75] D. E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 191975.

[KM90] Yves Kodratoff and Ryszard Michalski. *Machine Learning An Artificial Intelligence Approach*, volume 3, chapter 1, pages 13–16. Morgan Kaufman, 1990. See the bibliography of this book for extensive references to recent work in constructive induction.

[Kor87] R. E. Korf. Planning as search. *Artificial Intelligence*, 1987.

[Kor88] R. E. Korf. Optimal path-finding algorithms. In L. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 223–267. Springer-Verlag, 1988.

[Lev94] R. A. Levinson. Uds: A universal data structure. In W.M. Tepfenhart, J.P. Dick, and J.F. Sowa, editors, *Conceptual Structures: Theory and Practice*, number 835 in Lecture Notes in AI, pages 230–250. Springer-Verlag, Berlin, 1994.

[LF4a] Robert Levinson and Gil Fuchs. A pattern-weight formulation of search knowledge. Technical Report UCSC-CRL-91-15, University of California Santa Cruz, 1994a. Revision to appear in Computational Intelligence.

[LK93] R. Levinson and K. Karplus. Graph-isomorphism and experience-based planning. In D. Subramaniam, editor, *Proceedings of Workshop on Knowledge Compilation and Speed-Up Learning*, Amherst, MA., June 1993.

[LM88] K. F. Lee and S. Mahajan. A pattern classification approach to evaluation function learning. *Artificial Intelligence*, 36:1–25, 1988.

[LNR87] J. Laird, A. Newell, and P. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[LS91] R. Levinson and R. Snyder. Adaptive pattern oriented chess. In *Proceedings of AAAI-91*, pages 601–605. Morgan-Kaufman, 1991.

[LS93] R. Levinson and R. Snyder. Distance: Towards the unification of chess knowledge. *International Computer Chess Association Journal*, 16(3):315–337, September 1993.

[McC98] D. A. McCallester. Conspiracy numbers for minmax search. *Artificial Intelligence*, 35(3):287–310, 1998.

[Mic83] 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.

[Min84] S. Minton. Constraint based generalization- learning game playing plans from single examples. In *Proceedings of AAAI-84*, pages 251–254. AAAI, AAAI Press, 1984.

[Mir87] D. P. Miranker. Treat: A better match algorithm for ai production systems. In *Proceedings of AAAI-87*, pages 42–47. AAAI Press, 1987.

[MMT70] M.D. Mesarovic, D. Macko, and Y. Takahara. *Theory of Hierarchical, Multi-Level Systems*. Academic Press, Massachusetts, 1970.

[Mor94] E. Morales. Learning patterns for playing strategies. *International Computer Chess Association Journal*, 17(1):15–26, March 1994.

[Nau80] D.S. Nau. Pathology on game-trees: A summary of results. In *Proceedings of the First National Conference on Artificial Intelligence (AAAI-80)*, pages 102–104, Stanford, Calif., 1980. AAAI Press.

[Nil80] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, 1980.

[Pea82] J. Pearl. The solution for the branching factor of the alpha-beta pruning alghorithm and its optimality. *Communications of the ACM*, 25(8):559–564, 1982.

[Pel92] Barney Pell. METAGAME: A new challenge for games and learning. In H. J. van den Herik and L. V. Allis, editors, *Programming in Artificial Intellegence: The Third Computer Olympiad*. Ellis Horwood, 1992.

[Pit76] J. Pitrat. A program for learning to play chess. In *Pattern Recognition and Artificial Intelligence*. Academic Press, 1976.

[Qui86] J. R. Quinlan. Induction on decision trees. *Machine Learning*, 1:81–106, 1986.

[RK91] E. Rich and K. Knight. *Artificial Intelligence*. McGraw-Hill, 1991.

[RM86] E. D. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*, volume 1–2. MIT Press, 1986.

[RN94] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Massachusetts, 1994.

[Rob92] D.D. Roberts. The existential graphs. In *Semantic Networks in Artificial Intelligence*, pages 639–664. Roberts, 1992.

[Sac74] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[Sam59] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

[SN93] S.J. Smith and D.S. Nau. Strategic planning for imperfect information games. In *Proceedings of 1993 AAAI Fall Symposium on Games: Planning and Learning*, Menlo Park., 1993. AAAI Press.

[Sow83] J. F. Sowa. *Conceptual Structures*. Addison-Wesley, 1983.

[Sto79] G. Stockman. A minimax algorithm better than alpha-beta. *Artificial Intelligence*, 12(2):179–196, 1979.

[Sut88] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, August 1988.

[Sut91] R.S. Sutton. Special issue on reinforcement learning. *Machine Learning*, 1991.

[Tar56] A. Tarski. *Logic, Semantics, Metamathematics: Papers from 1923 to 1938*. Oxford University Press, Oxford, 1956.

[TS89] G. Tesauro and T. J. Sejnowski. A parallel network that learns to play backgammon. *Artificial Intelligence*, 39:357–390, 1989.

[Vaj92] S. Vajda. *Mathematical Games and How to Play Them*. Ellis Horwood Ltd., Great Britain, 1992.

[Wat85] S. Watanabe. *Pattern Recognition:Human and Mechanical*. Wiley, New York, 1985.

[Wil80] D. Wilkins. Using patterns and plans in chess. *Artificial Intelligence*, 14(2):165–203, 1980.