Exploiting the Physics of State-Space Search

Robert Levinson*
Dept. of Computer and Information Sciences
University of California
Santa Cruz, CA, 95060 USA
levinson@cse.ucsc.edu

Abstract

This paper is a blueprint for the development of a fully domain-independent single agent and multi-agent heuristic search system. The paper 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. These learning agents are based on generalizing the understanding obtained with the Morph chess system to all games involving the interactions of abstract mathematical relations. Blind learner can be viewed as a type of neural network that makes explicit use of fuzzy graph-isomorphism to exploit at a "meta-level" analogous states of the net itself. Publically available software has been developed for supporting research in blind learning and informed learning in the context of general problem-solving. The paper is based on studying the graph-theoretic structure of large open problems in machine learning research and giving a reasoned, unifying, framework for working toward their solution. In a field seeking comprehensive theories and general testbeds we hope that providing such a framework and software for exploration is an important contribution. The ultimate success of the framework awaits further investigation.

Introduction

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. (To not 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 degree that a heuristic is good is 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 is entirely domain-independent and is 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

*Partially supported by NSF Grant IRI- 9112862

are working on these separate but absolutely complementary projects:

- GENERIC-REPRESENTATION: A graph-theoretic 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, graph-theoretic representation).
- INFORMED-LEARNER: A program that performs well in state-space search given this graph-theoretic 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.
- MORPHII: 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. The basis for the work on each of the four subprojects is described below. The summation of this work is fully domain-independent adaptive game-playing software known as MORPHII. MorphII is a successor of the Morph chess system (13; 9) that has achieved approximate novice strength despite using just 1-ply of search and few human-supplied heuristics. This paper gives 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: graphs being too specific and insufficient freedom given the system with respect to the class of patterns that can be formed and to the combination of their weights.

Generic Representation of Games and Search Problems

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 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 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.
- Operators are ordered pairs of a set of primitive boolean conditions as pre-conditions and return another set (not necessarily disjoint from the preconditions) of post-conditions. An operator is legal at a given state if its preconditions are satisfied. 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 are 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 are achieved the game is over. Games will 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 players knowledge to a certain set of conditions in the current state, or to certain definitions of the operators.

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 returns (through the reward operators) -1 each time it occurs.
- Tic-tac-toe-like games (such as Qubic, Renju and GoMoku) in which objects are placed and never moved, have operators with one condition in the pre-list , one condition in the post list and all terminal hyperedges produce positive rewards as well.
- Hex is a game played on a grid of hexagons, that players alternate taking possession of, one player tries to make a path of his own hexagons from top to bottom and the other player from left to right. Hex falls perfectly into the Tic-Tac-Toe format by considering the hexagons as nodes and the various paths as the hyperedges, with rewards assigned correspondingly to which ever player can win in that path. But Hex can 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 (8) 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.

- Chess-like games such as those defined in MetaGame(15) have primitive operators that are more complicated than in Hex. Further, conditions may change non-monotonically (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), in which there is an element of point accumulation, have richer reward structures than Tic-Tac-Toe and thus although related to Hex have an even greater combinatorial explosion of goal states over primitive nodes since they involve goals that are in essence subsets 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]
- 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 necessary preconditions for operators.

Clearly, this representation is not fully adequate in that certain games (such as the stock market?) fall out of this framework, some games are expressed unnaturally and it is 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. The "verbiage" is a very succinct way of abstracting and referring to the underlying structure. Barney Pell's Metagame (15) is an excellent example of generating generic chess-like games at the normal abstract operator level. He also has general heuristics that allow decent MetaGames to be played without domain knowledge other than the rules. These heuristics are however confined to these chess-like games and don't generally deal with the mathematics of state-space search itself. Though it would seem that generalizing them to this larger class of games may be possible.

Hoyle (5) 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 outline how a more natural graph-theoretic generic game structure may be developed by taking advantage of the concept of variables and abstract operators.

Generic Games with Abstract Operators

To reach a more natural definition of generic games we need to add the notions of domain objects, static relations, dynamic relations, variables and bindings but still keep the "label-free" framework by omitting the arbitrary names assigned to objects, conditions and operators. [2] The framework is inspired by Peirce's existential graphs (16) and the more modern version "conceptual graphs" developed by John Sowa (17) and our own work in experience-based planning (12).

- Each domain will have a finite set of domain objects O1....Om.
- 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 domain, 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 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 desired property 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. Static relations are those that are constant for a given game. A frequent use of static relations is in defining board topology.
- Abstract operators are schemata or miniature hypergraphs with the following features. Nodes represent the variables in the schema condition or are constants representing domain objects, hyperedges are directed and "labeled" (with a pointer) with the relation that they refer to. Hyperedges are further labeled as to whether they are preconditions or postconditions and whether the relation is negated - these labels being domain-independent may be assigned directly. It will

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

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

be assumed that operator preconditions and postconditions are conjunctions of possibly negated relations.

- The relations appearing in operators that are not static relations are defined in terms of variables (and sometimes domain objects) rather than domain objects, hence their contents can change dynamically from state-to-state. Thus, these are called dynamic relations. A common dynamic relation is "ON" which gives which pieces are on given squares, for example, in board games. The fact that ON is a 1-1 mapping between sets, also facilitates learning - see the discussion of variables below.
- States, then, are hypergraphs over domain objects. As the static relations are always true it is only necessary to give the dynamic relations when representing a state.
- An operator is then applicable in a given state iff there is a 1-1 mapping in variables of the operator to domain objects s.t. 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 hyperedges associated with the pre-conditions and add those edges associated with the post-conditions under the variable binding. Since static relations are constant throughout the problem-solving process those bindings of objects that could ever possibly (constrained by the relations) satisfy an operator definition, could in principle be computed ahead of time leaving only the conditions to be checked. How to organize the preconditions of the operators best for matching is a topic considered elsewhere (3; 6; 14; 2; 11).
- Thus, an abstract generic game definition (as opposed to the one based on primitive operators and conditions above) is a complex nested directed hypergraph that includes the following: A finite set of domain objects, a finite set of static relations (as hyperedges of directed hyperedges) over the domain objects, a finite set of dynamic relations (as hyperedges over variables or domain objects) , a set of operators (as hypergraphs defined in terms of variables, domain objects, dynamic conditions and static relations) (with negation, precondition and postcondition as attributes) , an initial state as a set of dynamic relations , reward operators, and terminal hypergraphs (defined analogously to the reward operators and terminal conditions in the variable free format. Such hypergraphs can be represented directly using the conceptual graphs semantic network formalism (17).

  Thus, the conclusion is that a large spectrum of single-agent and multi-agent search problems can be viewed as games of graph-transforms directly analogous to organic chemical synthesis: states are graphs, and operators are graph-to-graph productions, terminal and reward conditions may also be expressed as graphs. 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 is suggests the potential for graph-theoretic analysis of the rules of a domain and ensuing experience for uncovering powerful heuristics and decision-making strategies.

Example 1: SWAP

The SWAP operator that switches a bottom block for a top block and vice versa in a tower of three blocks would have this verbal representation: PRE: on(x,y), on(y,z), clear(x) POST on(z,y), on(y,x), clear(z). 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, except that one of the variables is instead a domain object (for the blank tile).

Example 2: Tic-tac-toe

The following is a declarative representation of the rules (for APSII) of 3x3 tic-tac-toe based on the abstract operator representation. We assume player1 (X) is the one moving and that the board is transformed appropriately to make use of this.

```
Domain : Tic Tac Toe

Domain Objects:

SQUARE{S11,S12,S13,S21,S22,S23,S31,S32,S33}
PIECE: X,O,B

Static Relation:

THREE_IN_A_ROW<SQUARE, SQUARE, SQUARE>

Dynamic Relations:

ON<SQUARE, PIECE>
PRE_ON<SQUARE, PIECE>: ON
(* PRE_ON is ON used as a precondition *)
POST_ON<SQUARE, PIECE>: ON
(* POST_ON is ON used as a postcondition *)

Operator:

PUT_A_PIECE(SQUARE: s):
PRE_ON<s, B>
POST_ON<s, X>

Static State  (definition of static relations):

THREE_IN_A_ROW
{
  <S11, S12, S13>
  <S21, S22, S23>
  <S31, S32, S33>
  <S11, S21, S31>
  <S12, S22, S32>
  <S13, S23, S33>
  <S11, S22, S33>
  <S13, S22, S31>
}
```

```
Initial State:(assumes static state included)

ON<S11, B>
ON<S12, B>
ON<S13, B>
ON<S21, B>
ON<S22, B>
ON<S23, B>
ON<S31, B>
ON<S32, B>
ON<S33, B>

Terminal Condition:
(assumes having no legal moves is terminal)

THREE_IN_A_ROW<s1, s2, s3>
ON<s1, X> ON<s2, X> ON<s3, X>

Reward Operator:

Terminal-flag
(* flag set for terminal condition*)
[PLAYER: #1]-POST_SCORE->[1].
```

Evaluation of representation scheme

The representation scheme does not yet include facilities for inference over static and dynamic relations. Such a facility is available in conceptual graph theory but has not been necessary in the relatively straightforward search domains we have analyzed but would be required for efficient automatic theorem-proving. All such facilities will be available shortly in the Peirce conceptual graphs workbench (7; 4) in which our learning system is implemented.

An important step in showing the generality of the mechanism will be showing that games generated from the MetaGame generator take this structure. The implications of managing and manipulating such a structure are yet unknown, but the potential seems high. For example, any macro-operator that is executable in one search domain will work in any other (single-agent) domain that carries that macro- structure. Using such a domain-independent graph-theoretic representation we have been able to show that the entire TWEAK planning system can be reduced to 5 such abstract operators and a simple control structure and further that two classical AI problems: Roach's robot problem and Sussman's anomaly can be solved using the same database of domain-abstracted operators as macros (12). It is hoped that the evaluation of operator-condition interactions can be done across domain structures.

The hypergraph game: a generalization of Tic-Tac-Toe

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: 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(5), GoMoku and Qubic(1), hopefully putting that work in proper perspective.

The hypergraph representation (being unlabeled) makes symmetries fully realizable through graph-isomorphism. This mathematical representation also lends itself to reasonable heuristics such as 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. (i.e summation for live edges 1/n-1 where n is the number of nodes in that edge). Through the use of subgraph analysis more precise heuristics can be developed (5).

Reductions in the basic hypergraph game

It is useful to become familiar with reductions that preserve game-theoretic value (without incorporating search) 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 (for calculating the value of a state assuming optimal play). The idea is that we only need to remember for each edge which player owns it and which nodes remain unplayed. Given this information, edges with nodes owned by both players may be removed. A set of duplicate edges (edges that involve exactly the same nodes) can be reduced to a single-edge that preserves ownership if all owners are the same, or carries no ownership 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 that the move was to that was owned by another player.
2. Remove the node itself and set the ownership of all other affected edges to the player who has moved (the ownership fields may already be so set).
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.

Now that we have studied the basic-hypergraph games, we move the analysis to chess-like games. These games have the following additional features:

1. Operators are more complex as they have multiple pre-conditions and post-conditions.
2. Conditions can change non-monotonically: a condition which is true can become false and vice versa.

Chess-like games (15) are simply examples of perfect information, two-agent, search problems.

Improving on the Morph learning model

Once we understand that search-problems are games of graph-to-graph transformations, 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. Given our current understanding of the physics of state-space search, it is not yet known how to assess the value (as expected outcome) of such a subgraph directly but instead it may be learned statistically from experience as in the Morph chess system. Recall that in that system we limit search control to exactly 1-ply of lookahead to force us to focus on issues associated with heuristic construction and development. In fact, the restriction to 1-ply search is one of degree rather than kind since determining the presence of Morph's patterns requires 1 or 2 ply of search itself. Through the knowledge of static relations in a domain and experience we wish to pre-compile many search results into the heuristic.

Given that the values of subgraphs can be approximated accurately, as we believe is the case in Morph (though further improvements to the weight learning-mechanisms are certainly possible), we must ask ourselves then "why is Morph not a better chess player!?". We cite the following two major reasons for lack of more success in the underlying playing system:

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. A second weakness is in the specificness of the graphs in the chess system. The system lacked the ability for information learned about one graph to directly influence the values of other similar graphs. This leads to many learning inefficiencies, especially considering that for some specific graphs they 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 of the graphs themselves.

To continue with our design we assume a generic learning module with the the following properties:

- It contains n subsystems.
- It estimates the value of a state S, by first consulting its subsystems (which are themselves generic learning modules) to get their estimates of the value of S and combines their values numerically (perhaps non-linearly) using an equation that it has learned.
- The module receives feedback (as a "better prediction value") for its predictions from a higher-level system.
- The module learns by modifying its combination equation.

For the purposes of this discussion, we may ignore the learning method used by the generic learning module. Conceptually, any function learning method based on neural nets, regression analysis, nearest neighbor methods, etc. will do. Instead, we will focus on the identification of the subsystems themselves and towards their interaction.

The individual learning modules are to form a partially-ordered hierarchy based on the "subsystem" relation. It is assumed that all modules of the hierarchy receive feedback for individual states in a state sequence using the same temporal difference learning (18) mechanism as in Morph. At the bottom of the hierarchy are "primitive systems" these are systems with no subsystems that attempt to learn a constant that is the best predictor of their feedback values. Every module is responsible for predicting the outcome of the game, higher-level systems base their predictions by combining the recommendations of lower-level systems. To insure "action" the primitive systems in the hierarchy are initialized using a random, Gaussian distribution around the expected reinforcement value for the game. An informed learner may already produce through analysis the values of certain subsystems. Individual subsystems correspond directly with the patterns in the Morph chess system, but are more flexible as they learn how to combine values produced by subsystems lower in the hierarchy rather than attempting to determine a fixed constant weight.

As games move from the simple to complex the complexity of the individual learning module and the hierarchy of learning modules will need to increase as well. For simple games, as in 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 - corresponding 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 the values of objects on the 64 squares will not perform well (as experiments we have conducted on neural networks have shown) as it is known that no linear combination of these objects will do (since 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, since where do the complex interactions it needs to consider come from! Also, in order to **efficiently** learn the proper coefficients for this complex function, knowledge must be transferred and shared between terms (otherwise all pawn-can-take-queen combinations for example must be learned separately).

We hypothesize that the structure of the rules themselves dictates the structure of the system hierarchy, that by exploiting this structure the learning process can be made much more efficient and that the complexity of the learned evaluation function can be greatly reduced. For example,(consistent with the reduction rules given above) it is sufficient to evaluate a state accurately in tic-tac-toe to know the number of X's and O's in each row, column and diagonal. The counting procedure for each row, column and diagonal can use the same "Three-in-a-row" module and due to symmetry considerations the system need only consider different coefficients for three different types of "three-in-a-row relation". Consider, the generalization of tic-tac-toe, given as the basic hypergraph game above. For an arbitrarily large board, it would be difficult for a single learning module to learn to evaluate states well, without knowing the underlying relationships on the board. However, we hypothesize that any state in such a game can be evaluated accurately (after reductions as above) given the following:

- Primitive subsystems for every node in the domain.
- A subsystem for each hyperedge, that stands above the hierarchy for each individual node.
- A subsystem for each node that combines the values returned by each of its hyperedges.
- A top-level system that combines the values of each of its nodes.

In general:

- The primitive systems correspond to the primitive conditions in the underlying problem domain. These are usually caused by 1-1 relationships between sets as with the dynamic relation ON between pieces and squares in many games. The tile-puzzles have 81 primitive conditions, chess has 13x64 primitive conditions for pieces and squares (and a few more dealing with the potential for en passant capturing, draw by repetition and castling). These primitive conditions form the lowest level in the hierarchy.
- The second level in the hierarchy correspond to state-variables (squares for many domains) which are sets of primitive conditions such that at most one is on at a given time. Naturally, for the state-variables the best predictor will be the value of its primitive condition that is ON.
- The next level(s) in the hierarchy correspond to the static relations and type declarations in the domain. Thus in the tile-puzzles there is a system for every adjacent tile pair, in tic-tac-toe for squares forming a 3-in-a-row relation, in chess for all sorts of relationships between squares that are relevant in the rules (In-same-rank, In-same-file, In-same-diagonal, In-L-Shape, etc.).
- The next level deals for each second-level subsystem (square), with combining the values recommended by all static relationships that it is in.
- The final top-level combines the values for all second-level subsystems. It may be that rather than using a top-level a relaxation process should be invoked that feedbacks each nodes new values back into the level two nodes and cycles until node values converge.

This design deals directly with the first weakness in Morph discussed above since the combination functions learned will be localized and specific to the patterns under consideration (i.e. in the new framework two patterns that return exactly the same value may be treated differently by the problem-solving system based on their learned interactions with other patterns).

The second weakness (inability for patterns to share information) has been diminished somewhat by the hierarchy but still remains. Consider chess for a moment: as stated above the system would have to learn new values for every "in-same-diagonal" relation. To handle such redundant effort, each relation type will correspond to exactly one subsystem, while individual instances of the relation will supply input to this subsystem, but take the output as specific input to a higher-level system. Thus only one subsystem need be created for each relation type. Similar redundancy may also be exploitable in higher order subsystems. For example, in chess each square is involved in a similar set of relations with other squares though the amount and types of these relations may vary.

As another example, consider the tile puzzles. The lowest level corresponds to the value of each position-tile pair, level-two to the value of each position, level 3 corresponds to the "adjacency" relation between two squares used to define legal operators. Level 4 combines the value of such conditions at each node. Due to symmetry considerations only 3 types of level 4 modules are necessary. Finally, Level 5 combines the nine values recommended by each position (though the learning may also take advantage of the symmetries exploited on level 4).

Blind-Learning

Obviously, the problems of blind-learning are much more substantial than in the case of informed-learning, precisely because there is no domain definition to guide the construction of the learning hierarchy. Not only do the generic learning modules 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 can an informed learner start from a much better place, experience should also be exploited more accurately in Informed Learning than in Blind Learning. Still it is important to study blind learning, because in some domains one is not given access to any declarative description of the rules, because the additional difficulties in blind-learning give us a better appreciation for the information supplied in the rules, and because it forces the learning system (and its developer) to make use of every ounce of information available. In

the blind learning framework 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 (bits). It is intended that this bit description be *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 a "scrambled 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.
- That the other agent is playing with the same set of rules it is, i.e 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).
- 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.

Consider how Tic-Tac-Toe would appear to the blind learning agent: The board can be represented as 18 bits - 2 bits for each square: 00-empty, 01-X, 10-O, 11-is an unused code. To make things more obscure one can consider the board as representing a base 3 number (even with the squares placed in a strange order) and thus states could 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. This is intended. We feel that methods must be developed to produce a general learner - one capable of adapting, given proper amounts and variety of experience to any game-playing environment. *It is our hope that the graphs denoting the subsystems will help to exploit any isomorphic interactions of values wherever they might occur. It is exactly this exploitation of "analogous relationships" through graph-isomorphism that separates MorphII from virtually all other learning systems.*

A blind-learner would start as a two-level system with primitive subsystems for individual bits and a top level system that learns to combine the values for each bit. Intermediate subsystems may evolve by creating a subsystem for every set of bits that disappear or appear with a single operator application. These add-sets and delete sets correspond to complex relations, intersections of these may correspond to simpler relations. Further, state-variables (corresponding to level 2 in the hierarchy discussed above) can be recognized by starting with all bits in a single variable and gradually splitting into smaller variables by finding counter-examples (10). Such variables correspond very well to unary relations or types in the hidden game definition. For example, these could be pieces or squares in a non-malicious encoding of chess.

Exploiting analogous relationships

In the blind-learning framework subsystems will be formed based on relation instances rather than relation types as in informed-learning. Thus, it is critical to try to identify similar relations (by comparing coefficients in their learned combination equations). Such identification requires a type of fuzzy-graph matching.

To further understand the motivation for fuzzy-graph matching consider the blind learning framework and the combinatorial explosion associated with chess. Consider, for instance, the difficulty of learning that placing one's queen where a pawn can capture it is usually bad regardless of where it occurs on the board – including in those places it has not yet occurred! Learning such a straightforward concept is actually a difficult one for all learning algorithms - especially those operating on uninterpreted structures. Here we are assuming no knowledge of piece, square, pawn or queen etc. has been supplied.

We have used the term "variable" to denote a set of bits such that at most one is on at a given time and have described how such variables can be learned. We shall say that the value of a variable at a given time is equal to the value (if any) of its positive bit. Thus the generic learning module can be applied to learning the value of a conjunction of two variables as a combination of their underlying values.

We shall assume that the blind learning system has observed several instances of the pawn-can-take-queen relationship (i.e relationships between two uninterpreted piece-square conditions due to this relationship) and encounters a new instance. If it has seen this instance as a capture before it hopefully will be fine, since it would have created a specific relation for the deleted and added conditions. the But if not seen before there may still be a learned interaction between the pawn-square condition and a variable formed by the conditions referring to the square the queen is on.

That is, the desired concept can be learned as a result of learning the square "variable" where the queen is on and having formed a conjunction of the pawn-square condition (or a variable involving it) with this variable. The variable gets instantiated with the value of the queen-on-square condition. Presumably there have been other captures by the pawn to the square the queen is on. Thus the learned interaction between these squares will hopefully look similar (to some degree) to learned interactions between other pairs that denote "attack". Thus, sharing closeness with these other interactions (where in fact queens may have been captured) a possibly accurate transfer of knowledge may be made. The assumption is that games of interest have regular underlying structure - to the degree that this assumption of regularity is false the more difficult is the learning task and the more false inferences that will need to be made before the proper structure can be learned - if at all.

If no connection between these squares has been noticed before, being ignorant of board topology and the rules, it would be virtually impossible to recognize the interaction – though possibly it would be inferrable very indirectly from other patterns.

MorphII: Domain-Independent Games Environment in C++

The blind learning and informed learning testbeds described above are available as part of the public domain software known as the Peirce Conceptual Graphs Workbench(4). The learning system in Peirce, known as MorphII, accepts the rules of a single agent and multi-agent state-space search domain, translates them into conceptual graphs, and then monitors games and learning via a "super -referee". As the code is fully objectified in C++, it is possible for learning modules to be coded and tested as well as for modifications to be made to our initial implementations of blind learning and informed learning algorithms as described above. Declarative rule sets for a number of games are available, including backgammon, chess, tic-tac-toe, and the 8-puzzle. It is also possible to test the power of different search algorithms and human supplied heuristics in these domains.

Acknowledgements

Barney Pell provided useful criticism and encouragement during the writing of the paper. John Amenta developed much of the MorphII software, Yuxia Zhang assisted in developing the declarative definitions of games.

References

[1] 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.

[2] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA, October 1992.

[3] G. Ellis. Efficient retrieval from hierarchies of objects using lattice operations. In G. Mineau and B. Moulin, editors, *Proceedings of First International Conference on Conceptual Structures (ICCS-93*, Montreal, 1993. To Appear.

[4] G. Ellis and R. A. Levinson. The birth of peirce. In *Conceptual Structures: Theory and Implementation.* Springer-Verlag, 1992.

[5] S. Epstein. Deep forks in strategic maps – playing to win. In *Heuristic Programming in Artificial Intelligence 2*, pages 189–203. Ellis Horwood, 1991.

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

[7] 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.

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

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

[10] R. Levinson. Towards domain-independent machine intelligence. In *Conceptual Graphs for Knowledge Representation*, volume 699 of *Lecture Notes in Computer Science*, pages 254–273. Springer-Verlag, 1993.

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

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

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

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

[15] 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.

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

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

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