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

[Tesauro, 1991] Gerald Tesauro. Practical issues in temporal difference learning. IBM Thomas J. Watson Research Center, 1991.

[Thompson and Roycroft, 1983] K. Thompson and A. J. Roycroft. A prophesy fulfilled. *EndGame*, 5(74):217–220, 1983.

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

[Yee *et al.*, 1990] R. C. Yee, Sharad Saxena, Paul E. Utgoff, and Andrew G. Barto. Explaining temporal differences to create useful concepts for evaluating states. In *Proceedings of the Eighth National Conference on AI*, Menlo Park, 1990. American Association for Artificial Intelligence, AAAI Press/The MIT Press.

[Levinson, 1991a] R. Levinson. Pattern associativity and the retrieval of semantic networks. *Computers and Mathematics with Applications*, 1991. To appear in Special Issue on Semantic Networks in Artificial Intelligence, Fritz Lehmann, editor.

[Levinson, 1991b] R. Levinson. A self-organizing pattern retrieval system and its applications. *Internation Journal of Intelligent Systems*, 1991. To appear.

[Metropolis *et al.*, 1953] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[Michalski, 1983] 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.

[Michie and Bratko, 1987] D. Michie and I. Bratko. Ideas on knowledge synthesis stemming from the KBBKN endgame. *Intern. Computer Chess Assoc. Journal*, 10(1):3–13, 1987.

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

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

[Muggleton, 1988] S. H. Muggleton. Inductive acquisition of chess strategies. In D. Michie J. E. Hayes and J. Richards, editors, *Machine Intelligence 11*, pages 375–389. Oxford University Press, Oxford, 1988.

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

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

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

[Samuel, 1959] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959. Reprinted in CT, pp. 71–105.

[Samuel, 1967] A. L. Samuel. Some studies in machine learning using the game of checkers–ii recent progress. *IBM Journal of Research and Development*, 11(6):601–617, 1967.

[Scherzer *et al.*, 1990] T. Scherzer, L. Scherzer, and D. Tjaden. Learning in Bebe. In T. A. Marsland and J. Schaeffer, editors, *Computer, Chess and Cognition*, chapter 12, pages 197–216. Springer-Verlag, 1990.

[Slate, 1987] D. J. Slate. A chess program that uses its transposition table to learn from experience. *Intern. Computer Chess Assoc. Journal*, 10(2):59–71, 1987.

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

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

[Tadepalli, 1989] P. Tadepalli. Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989. Morgan Kaufmann.

References

[Christensen and Korf, 1986] J. Christensen and R. Korf. A unified theory of heuristic evaluation. In *AAAI-86*, 1986.

[Davis and Steenstrup, 1987] Lawrence Davis and Martha Steenstrup. *Genetic Algorithms and Simulated Annealing*, chapter Chapter 1, Genetic Algorithms and Simulated Annealing: An Overview. Research Notes in Artificial Intelligence. Morgan Kaufmann Publishers, 1987.

[DeJong and Smith, 1981] K.A. DeJong and T. Smith. Genetic algorithms applied to information driven models of us migration patterns. In *12th Annual Pittsburgh Conference on Modeling and Simulation*, April 1981.

[DeJong, 1980] K.A. DeJong. Adaptive system design: a genetic approach. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10(9), September 1980.

[Epstein, 1990] S. L. Epstein. Learning plans for competitive domains. In *Proceedings of the Seventh International Conference on Machine Learning*, June 1990.

[Fitzpatrick *et al.*, 1984] J.M. Fitzpatrick, J. J. Grefenstette, and D. Van Gucht. Image registration by genetic search. In *IEEE Southeastcon '84*, pages 460–464, April 1984.

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

[Glover, 1987] David E. Glover. *Genetic Algorithms and Simulated Annealing*, chapter Chapter 1, Solving a Complex Keyboaard Configuation Problem Through Generalized Adaptive Search. Research Notes in Artificial Intelligence. Morgan Kaufmann Publishers, 1987.

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

[J. H. Holland, 1978] J. R. Reitman J. H. Holland. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed Inference Systems*. Academic Press, New York, 1978.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

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

[Levinson and Ellis, 1991] R. Levinson and G Ellis. Multilevel hierarchical retrieval. *Knowledge-Based Systems*, 1991. To appear.

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

[Levinson *et al.*, 1990] R. Levinson, R. Snyder, B. Beach, T. Dayan, and K. Sohn. Adaptive-predictive game-playing programs. Technical Report UCSC-CRL-90-12, University of California at Santa Cruz, 1990. Submitted to Journal of Experimental and Theoretical AI.

[Levinson, 1984] R. Levinson. A self-organizing retrieval system for graphs. In *AAAI-84*, 1984.

[Levinson, 1989a] R. Levinson. A pattern-weight formulation of search knowledge. Technical Report UCSC-CRL-89-05, University of California at Santa Cruz, 1989. Submitted to Computational Intelligence.

[Levinson, 1989b] R. Levinson. A self-learning pattern-oriented chess program. *ICCA*, 12(4):207–215, December 1989.

6. Conclusions and Ongoing Directions

Getting systems to develop their knowledge bases from experience is a difficult but important challenge. It is hoped that the example the Morph project provides of how several learning methods can be combined to exploit experience in the form of pws will provide at least one framework on which research in experiential learning may proceed. The Morph project will be continuing over a number of years as we strive to bring its playing level up to the current brute-force chess machines. To do this the effort will be to make the learning mechanisms and their mutual cooperation as sharp as possible. Hopefully, this will prepare the way for more compelling applications of these methods beyond chess. For example, it may be possible in organic synthesis systems to improve search time with experience using similar graph methods [Levinson, 1991b].

The following points are worth remembering:

- In combining the many learning methods in the Morph system we have not taken the methods as they are normally used but have extracted their essence and combined them beneficially.
- Guided by appropriate performance measures, modification and testing of the system proceeds systematically.
- Interesting ideas arise directly as a result of taking the multi-strategy view. Some examples:
  1. *The genetic inversion operator described in Section 2.4.2.*
  2. *Optimal pattern population.* Just as we are trying to get the learning methods to work in harmony, we are attempting the same coordination with Morph's patterns. The idea is to get a set of patterns that are good predictors as a whole rather than to find strong individual patterns (though the latter may be part of the former).
  3. *Higher level concepts via hidden units.* Once a good set of patterns has been obtained we may be able to introduce a more sophisticated evaluation function. This function, patterned after neural nets, would have hidden units that extract higher level interactions between the patterns. For example, conjunctions and disjunctions may be realized and given weights different from that implied by their components.

Acknowledgments

Thank you to Jeff Keller for constructing the new move selection evaluation function, to Paul Zola and Kamal Mostafa for the initial Morph implementation, and to Richard Sutton for sharing our enthusiasm for reinforcement learning.

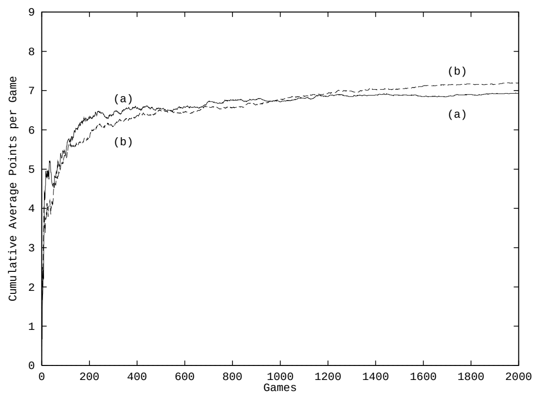

Figure 5.1: Cumulative Average of two versions of Morph.

Version (a) is the basic Morph in existence three months ago. Version (b) adds the reverse node ordering pattern addition scheme to Version (a).

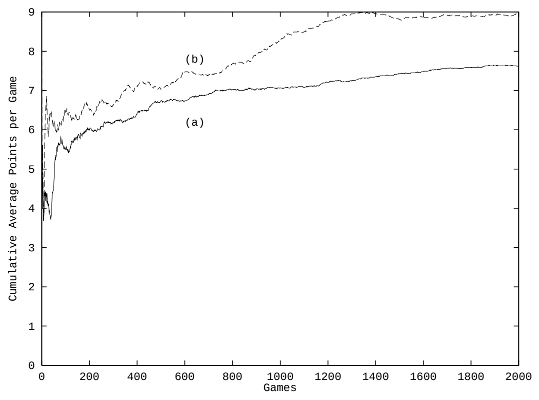

Figure 5.2: Cumulative Average of two versions of Morph.

Version (a) adds the split evaluation function on top of reverse node ordering Morph. Version (b) adds annealing on top of Version (a).

5. Performance Results

To date Morph has not defeated GnuChess, though it has obtained over 20 draws via stalemate, repetition of position and the 50-move rule. Despite the lack of success against GnuChess there have been many encouraging signs in the nine months since Morph was fully implemented:

- Even though no information about the relative values of the pieces (or that pieces are valuable) have been supplied to the system, after 30 or more games of training Morph's material patterns are consistent and credible [Levinson and Snyder, 1991]. The weights correspond very well to the traditional values assigned to those patterns. These results reconfirm other efforts with TD learning [Christensen and Korf, 1986] and perhaps go beyond by providing a finer grain size for material.
- After 50 games of training, Morph begins to play reasonable sequences of opening moves and even the beginnings of book variations. This is encouraging because no information about development, center control and king safety have been directly given the system and since neither Morph or GnuChess uses an opening book. It is not rare for Morph to reach the middlegame or sometimes the endgame with equal chances before making a crucial mistake due to lack of appropriate knowledge.
- Morph's database contains many patterns that are recognizable by human players and has given most of these reasonable values. The patterns include mating patterns, mates-in-one, castled king and related defenses and attacks on this position, pawn structures in the center, doubled rooks, developed knights, attacked and/or defended pieces and more.

5.1 Performance Evaluation

To explore new additions to Morph, one implementation is compared with another by using the average number of traditional chess points per game as the metric. Each implementation is run until the metric is no longer increasing this. (Most Morphs stop learning at between 1500 and 2000 games of play). The one with the higher rating is considered the better. We have concluded that only one such comparison is sufficient because the same version of Morph usually reaches the same average.

The "meaning" of the hidden units to which weights are associated in neural nets is usually not clear, whereas in Morph it is specific structures that are given weights. The resulting transparency of Morph's knowledge has allowed us to fine tune ts learning mechanisms - with various system utilities it is possible to ascertain exactly why Morph is selecting one move over another.

5.2 Improvement through Adding New Learning Methods

Adding learning strategies is a gradual process. Each method must be added one at a time to see if it increases performance. If it does than it is kept. Since Morph's initial implementation significant performance increases have occurred due to such additions. The following two graphs show Morph's cumulative average over time. These graphs compare the performance of four versions of the system. Each version is an extension of the previous one. Figure 5.1a shows a basic Morph, Figure 5.1b shows the result of adding reverse node ordering, Figure 5.2a shows the result of splitting the evaluation function, and Figure 5.2b shows the result of adding annealing.

search to check plans and the restriction to tactical positions distinguish it from Morph. Also, Paradise is not a learning program: patterns and planning knowledge are supplied by the programmer. Epstein's Hoyle system [Epstein, 1990] also applies a semantic approach but to multiple simultaneous game domains.

Of course, the novel aspects of the Morph system could not have been achieved without the unique combination of learning methods described here.

4. Relationship to Other Approaches

Above, we have described how the chess system combines threads of a variety of machine-learning techniques that have been successful in other settings. To produce this combination, design constraints usually associated with these methods have been relaxed.

We feel the integration of these diverse techniques would not be possible without the uniform, syntactic processing provided by the pattern-weight formulation of search knowledge. To appreciate this, it is useful to understand the similarities and differences between Morph and other systems for learning control or problem-solving knowledge. For example, consider Minton's explanation-based Prodigy system [Minton, 1984]. The use of explanation-based learning is one similarity: Morph specifically creates patterns that are "responsible" (as preconditions) for achieving future favorable or unfavorable patterns. Also similar is the use of "utility" by Morph's deletion routine to determine if it is worthwhile to continue to store a pattern. The decision is based on accuracy and significance of the pattern versus matching or retrieval costs. A major difference between the two approaches is the simplicity and uniformity of Morph's control structure: no "meta-level control" rules are constructed or used nor are goals or subgoals explicitly reasoned about. Another difference is that actions are never explicitly mentioned in the system. Yee et al.[Yee *et al.*, 1990] have combined explanation-based learning and TD learning in a manner similar to Morph. They apply the technique to Tic-Tac-Toe.

It is also interesting to compare Morph to other adaptive-game playing systems. Most other systems are given a set of features and asked to determine the weights that go with them. These weights are usually learned through some form of TD learning [Tesauro and Sejnowski, 1989]. Morph extends the TD approaches by exploring and selecting from a very large set of possible features in a manner similar to genetic algorithms. It is also possible to improve on these approaches by using Bayesian learning to determine inter-feature correlation [Lee and Mahajan, 1988].

A small number of AI and machine learning techniques in addition to heuristic search have been applied directly to chess, and then, usually to a small sub-domain.The inductive-learning endgame systems [Michie and Bratko, 1987; Muggleton, 1988] have relied on pre-classified sets of examples or examples that could be classified by a complete game-tree search from the given position [Thompson and Roycroft, 1983]. The symbolic learning work by Flann [Flann and Dietterich, 1989] has occurred on only a very small sub-domain of chess. The concepts capable of being learned by this system are graphs of two or three nodes in Morph. Such concepts are learned naturally by Morph's generalization mechanism.

Tadepalli's work [Tadepalli, 1989] on hierarchical goal structures for chess is promising. We suspect that such high-level strategic understanding may be necessary in the long run to bring Morph beyond an intermediate level (the goal of the current project) to an expert or master level. Minton [Minton, 1984], building on Pitrat's work [Pitrat, 1976], applied constraint-based generalization to learning forced mating plans. This method can be viewed as a special case of our pattern creation system. Perhaps the most successful application of AI to chess was Wilkin's Paradise (PAttern Recognition Applied to DIrecting Search) system [Wilkins, 1980], which, also building on Pitrat's work, used pattern knowledge to guide search in tactical situations. Paradise was able to find combinations as deep as 19-ply. It made liberal use of planning knowledge in the form of a rich set of primitives for reasoning and thus can be characterized as a "semantic approach." This difference plus the use of

3. Integration

It is useful to reflect on how the different methods support each other synergistically within Morph:

1. TD learning requires a set of features for evaluating states, these features are constantly being created, generalized and deleted by other learning modules.
2. The patterns created by some of the creation modules, while potentially useful have not been designed to specifically handle the mis-evaluations that occur during play. They also do not produce the macros that are required to reduce search. Specific patterns for these purposes are created by backing up extreme patterns through goal regression(EBG).
3. Because the feature set is changing dynamically, convergence of TD learning can not be guaranteed. However, the cooling process of simulated annealing is used to give each pattern a decreasing learning rate thus guaranteeing local convergence for that pattern. Experience has shown that the weights to which these patterns converge are appropriate ones. Once parts of the system have converged properly they provide a sound basis for the slower learning patterns in the system to converge to appropriate weights as well.
4. Bit strings are usually used to represent patterns in genetic algorithms. Unfortunately, in chess a bit string representation of chess positions is probably not an appropriate one for learning, as features (substrings) are probably too specific to be useful. Thus, we use the structured graph patterns described in Section 2.1, since they may be invariant across many board positions.
5. TD learning and weight-updating are only as strong as the heuristic evaluation function that supports them. It is this function that determines how the many weights arising from a given position are combined to provide a single value for that position. Recall that we have opted to use a different function for weight-updating than that used during play.

Explanation Based Generalization (EBG)

In order for Morph to compete using 1-ply of search, a means must exist by which combinational (or macro) knowledge is given to the system. Macros can be represented as pws by constructing a sequence of them such that each pattern is a precondition of the following one. With successive weights gaining in extremeness the system is then motivated to move in this direction. But here it is worth mentioning the advantage of pws over macros: while executing one macro, the system has the potential to switch into another more favorable macro rather than being committed to the former.

To construct such sequences of pws in the Morph system a form of EBG or goal regression is used. The idea is to take an extreme pattern in one position and back it up to get its preconditions in the preceding position. If this new pattern is also most extreme the process can be continued etc. We like to call this technique "reverse engineering" as the pw-sequence is discovered through retrograde analysis. The advantages of this technique are more than just learning "one more macro": each of the patterns can be used to improve the evaluation of many future positions and/or to start up the macro at any point in the sequence.

Node ordered induced subgraphs

A simple and rapid mechanism for getting useful patterns in Morph proceeds as follows: Take the graph of a position, number the nodes in a connected fashion using a heuristic rule, choose a random size n, and return the induced subgraph formed by the first n nodes in the node ordering (and the edges between them). Morph uses two relatively game-independent node ordering rules: In forward node ordering, nodes are ordered by most recently moved piece while maintaining connectivity of the nodes. In reverse node ordering nodes are ordered by which piece is next to move while maintaining connectivity of the nodes. In both schemes captured pieces and kings in check are placed high in the list and ties (for squares and unmoved pieces, for instance) are broken randomly. The inclusion of random factors in the above scheme also fall well within the genetic algorithm viewpoint, since the system is then capable of generating and exploring a large set of possibilities.

Pattern Deletion

Also, as in genetic algorithms there must be a mechanism for insignificant, incorrect or redundant patterns to be deleted (forgotten) by the system. A pattern should contribute to making the evaluations of positions it is part of more accurate. The utility of a pattern can be measured as a function of many factors including age, number of updates, uses, size, extremeness and variance. We are exploring a variety of utility functions [Minton, 1984]. Using the utility function, patterns below a certain level of utility can be deleted. Deletion is also necessary for efficiency considerations: the larger the database the slower the system learns.

Genetic Operators

Genetic algorithms [Holland, 1975] are a means of optimizing global and local search [Glover, 1987]. In these algorithms solutions to a problem are encoded in bit strings that are made to evolve over time into better solutions in a manner analogous to the evolution of species of animals. The bits in the solution representation are analogous to genes and the entire solution to a chromosome. In such systems there are a fixed number of solutions at any one time (a generation). Members of each generation inter-breed to form the next generation. Each genetic algorithm has a fitness function that rates the quality of the solution in a particular generation. When it is time for a new generation to be created from the current generation the solutions that are more fit are allowed to be the more active breeders. Breeding usually involves three processes: crossover, inversion, and mutation.

The three methods work as follows. Crossover involves randomly selecting a point the same distance along both parents and splitting the solutions in half. Then the tails of each parent are swapped producing two hybrid children. Inversion involves taking a single solution in the current generation, selecting two points along the bit string and then inverting the bits between the two points. The resulting bit string is a solution in the new generation. Finally, mutation involves flipping a bit in a solution of the new generation. The new generation will be of the same size as the old generation [Davis and Steenstrup, 1987].

Adapting genetic algorithms for Morph

Morph uses patterns to reason about positions that it encounters. These patterns have countless variations; far too many to be stored in a computer. Thus, it is necessary to pick the best set of patterns to generate the most accurate evaluations of positions. Likewise, it is desirable to weed out those patterns that lead to erroneous evaluations or are too specific to be used often. Thus, Morph's search takes place in the space of possible pattern sets.

In Morph:

- Patterns are the basic elements of a given population. These patterns do not represent solutions but instead represent important elements of positions to be looked for.
- The initial population is created via a method called seeding where we introduce all two node graph patterns (such as white bishop attacks black queen) into the database.
- The fitness function that Morph uses favors those patterns that are found in positions often, have low variance and have extreme values.
- Parameters exist for keeping the number of patterns below a fixed number. This number balances the desire to have many patterns with the desire to play and learn quickly. Thus, although we are currently measuring Morph's performance over a number of games, it is probably more appropriate to measure it in terms of computing time - in which the size of the database becomes an important factor.
- Morph has operators that are analogous to those used in genetic algorithms but differ in some important respects because graphs are not bitstrings. In particular, they are not ordered nor do they have a fixed length. Morph's generalization and specialization operators (see above) are similar to crossover and mutation, respectively. While we currently do not have an inversion operator in Morph, one could imagine the utility of swapping white and black color designations in all or part of a graph pattern.

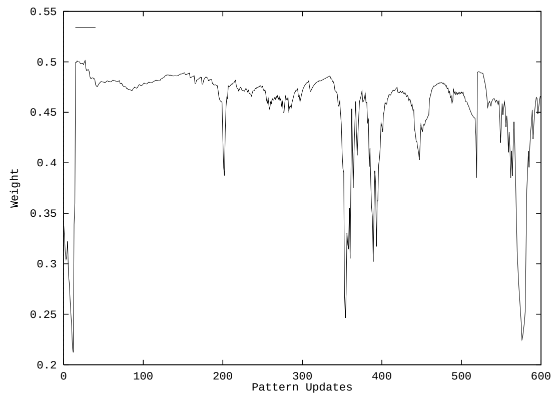

Figure 2.3: This graph depicts the weight change of the material pattern “down 1 bishop” in a system without simulated annealing.

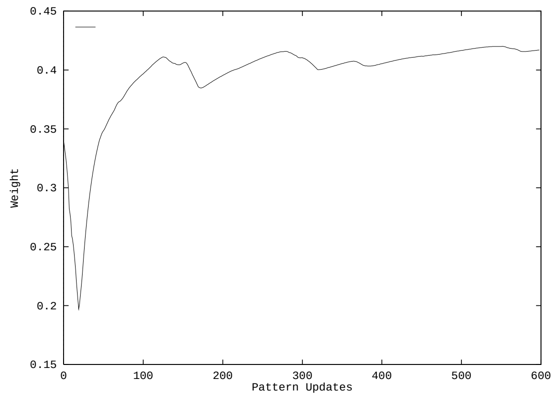

Figure 2.4: This graph depicts the weight change of the same pattern in Figure 3 in a system that has simulated annealing added.

the same pattern when it resided within a Morph that used simulated annealing. Here the weight of the pattern tends to home in on a specific value.

Pattern Creation

The system must have a method for developing and storing new patterns. To reduce complexity, key subpatterns, rather than full position graphs, are used. Within Morph, four main methods for extracting or creating these subpatterns are used:

1. generalization and specialization of existing patterns as in concept induction
2. pattern extraction and mutation operators as in genetic algorithms
3. goal and subgoal regression as in explanation-based generalization.
4. node ordered induced subgraphs

The proper mix of these methods is an important issue currently being explored.

Generalization and Specialization

In concept induction schemes [Michalski, 1983; Mitchell *et al.*, 1986; Niblett and Shapiro, 1981; Quinlan, 1986] the goal is to find a concept description to correctly classify a set of positive and negative examples. In general, the smaller description that does the job, the better. Sometimes the concept description needs to be made more specific to make a further distinction whereas at other times it can be simplified without loss of discriminative power.

In Morph, of course, positive and negative examples are not supplied. Still, one can view the evaluations arising from TD learning as serving this purpose. Morph creates generalizations of learned patterns by taking maximum common subgraphs of two patterns that have similar weights and similar structures. Whereas in a standard concept induction scheme the more specific patterns may be deleted, in Morph we keep them around for the moment because they can lead to further important distinctions. The regular deletion module may delete them later, if they should no longer prove useful.

Morph will *specialize* a pattern if its weight must be updated a large amount (indicating inaccuracy). The pattern is specialized by adding a connected edge (and node) to the most recently moved or captured piece in the graph.

in statistical mechanics [Davis and Steenstrup, 1987]. Statistical mechanics attempts to describe the properties of complex systems. It is often desirable to take systems of molecules and reduce them to the lowest possible energy by lowering the temperature of the system. Through experience it has been found that if the temperature is reduced too quickly the system will have a small probability of being at an optimally low temperature. Metropolis et al [Metropolis *et al.*, 1953] developed a technique for lowering the temperature gradually to produce (on the average) very low energy systems at the lowest temperature: Cool the system at a particular temperature, let the system reach equilibrium, then adjust the temperature to a nearby (usually lower) value. Continue until the final temperature is reached.

Kirkpatrick et al [Kirkpatrick *et al.*, 1983] adapted annealing to computer science, by finding information analogies for their physical counterparts [Davis and Steenstrup, 1987]:

- The energy of the system became an objective function that describes how good of a state the system is currently in.
- Moving a physical system to its lowest energy state is then analogous to finding the state that optimizes the objective function.
- The state of the system is then the different informational parameters that the system can have.
- The temperature is a mechanism for changing the parameters.

Our situation is similar to that of the statistical physicist. Morph is a complex system of many particles (patterns). The goal is to reach an optimal configuration, i.e. one in which each weight has its proper value. The average error (i.e. the difference of Morph's prediction of a position's value and that of temporal-difference) serves as the objective evaluation function. Intuitively, the average error is an acceptable performance evaluation metric if one accepts the empirical and analytic arguments that TD learning is a convergent process [Sutton, 1988a]: since TD learning will produce evaluations close to the true ones the error will be high or low depending on Morph's degree of accuracy.

The state of Morph's system is made up of the patterns' weights. Temperature corresponds to the rate at which a pattern moves towards the value recommended by TD learning. In addition to using a (global) temperature that applies to all patterns, each patterns has its own independent temperature. The reason this is done is because each pattern has its own learning curve depending on the number of boards it has occurred in. A pattern that has occurred in many boards has its temperature reduced more quickly than a pattern that occurred in only a few boards. This is because the first pattern has more examples to learn from and hence early convergence is appropriate. Each pattern's learning rate is affected by its number of updates and the global temperature as follows:

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

Weight_i is the weight after the ith update, new is the what TD recommends that the weight should be, n is there number of the current update and k is the global temperature. When $k = 0$ the system only considers previous experience, when $k = 1$ the system averages all experience, and when $k > 1$ the present recommendation is weighted more heavily then previous experience. Thus, raising k creates faster moving patterns.

Performance Example of Annealing

Figure 2.3 and Figure 2.4 show two update histories for a given pattern: how its weight changes over time. The first figure shows a pattern in a version that did not use simulated annealing. It tends to fluctuate within a certain range. The second figure displays of updates. TD learning and simulated annealing and their use in Morph are described in the following two subsections.

Temporal-Difference Learning

TD learning was designed for situations where the learner does not get immediate feedback for his predictions; instead, the learner makes a sequence of predictions and then is given the true value of the last prediction only. The learner is usually trying to predict the value of a particular phenomenon given a set of input values. In most TD systems a subset of the input is used. Still, in some simple domains the entire state can be used as input to a evaluation function, e.g. the markov model learning [Sutton, 1988b]. Associated with each input feature is a real valued weight, it is precisely this weight that gets updated and is used to make the next evaluation.

After the learning system makes a sequence of predictions and receives feedback for the final prediction, TD learning proceeds to modify the weights of the input features on each state working back from the final state in the sequence to the first state. The weights of the features associated with the final state are updated in the direction of the true value supplied by the environment. For each of the other states, weights are updated in the direction of the new evaluation for the succeeding state.

An Example of TD Learning

Assume that there is an election and a TD system is trying to predict the percentage of the votes to be received by candidate A. Suppose, there exist two newspapers that publish polls, S and T, on the percentage of votes each candidate will receive. The TD system will use S and T to predict A's percentage as follows: $P = (sS + tT)/(s + t)$. s and t are real-valued weights denoting the credibility of each poll. Lets say that the newspapers come out with three polls before the election. The system will make three predictions before the election and then will update the credibility weights s and t, according to the results of the election.

```
Initially s = 1, t = 1

Poll 1:   S = .5, T = .4;   P = .45
Poll 2:   S = .2, T = .35;  P = .275
Poll 3:   S = .3, T = .5;   P = .4

Election Result = .45

Updating at Poll 3 -- move s, and t so that P
tends (say half way)
towards .45
s = 1.08, t = 1.8, giving P = .425;

Updating at Poll 2 -- move toward .425
s = 0,  t = 1, giving P = .35

Updating at Poll 1 -- move toward .35
s = 0,  t = 1, giving P = .4
```

Since Morph must make a sequence of predictions (board evaluations) but only receives feedback for the last one (0 – lose, 1 – win, .5 draw) TD learning is appropriate. This has been true for other adaptive game playing systems in which the credit assignment task is difficult and critical [Samuel, 1959; Levinson *et al.*, 1990; Tesauro and Sejnowski, 1989; Tesauro, 1991]. In Morph, TD learning is implemented close to the standard way. It deviates in one important point though: while most systems use a fixed set of features determined before learning begins, Morph's feature set changes over time. As we wish to keep Morph's human supplied knowledge to a minimum, it is left on its own to determine the proper feature set. Morph's material patterns greatly enhance the rate of TD learning since they provide useful subgoals that may occur anywhere in a game. This is especially so since Morph tends to learn these values early on in training.

Simulated Annealing

Simulated annealing is a learning procedure that has been derived from a practice

$$f(x,y) = \begin{cases} -.5(2-2x)(2-2y)+1 & \text{if } x \geq \frac{1}{2} \text{ and } y \geq \frac{1}{2} \\ \frac{x-.5+y-.5(2x-1)(2y-1)((2x-1)^2-(2y-1)^2)}{(2x-1)^2(2y-1)^2} & \text{if } x \geq \frac{1}{2} \text{ and } y < \frac{1}{2} \\ f(y,x) & \text{if } x < \frac{1}{2} \text{ and } y \geq \frac{1}{2} \\ 2xy & \text{otherwise} \end{cases}$$

Figure 2.2: Move selection evaluation function.

that the current board is a losing position and patterns with weight 1 suggest that the current board is a winning position.

The second function is a weighted average:

$$\text{Eval} = \frac{\sum_{i=1}^{n} w_i * (|w_i - .5|^{\beta})}{\sum_{i=1}^{n} |w_i - .5|^{\beta}}$$

Where $\{w_i\}$, $1 \leq i \leq n$, are the weights of the patterns matching the current board and β is a configurable power, usually between 1 and 5.

The reason this function is useful for updating is that, being an average, it causes weights to move minimally to include the new data point; thus the system state remains relatively stable.

Performance Example

The following example demonstrates the necessity for using the first evaluation during play. Lets say that Morph is in a game where he is losing and he knows it. This would mean that all evaluations for the next possible move are below .5. From experience we have found that this is usually caused by one extremely bad pattern. For instance it could be a material pattern that says Morph is down a queen with weight .2. Further lets say that there are only two possible alternatives for Morph: Position A with just the .2 pattern and Position B with the .2 pattern and a .3 pattern (such as pawn attacking Morph's rook). Obviously, position A should be considered better since it only has one unfavorble pattern. Unfortunately, function 2 will choose position B since it returns a weighted average of .22. Further trouble with using function 2 for move selection can be seen by considering a new position C with a .2 and a .4 pattern. Because of the use of extremeness, function 2 will give position C an evaluation of .21 and incorrectly prefer B. Note that function 1 evaluates A, B, and C as .20, .12, and .16, respectively and thus maintains the proper order.

2.4 Learning System

The learning system has three parts defined in the following three subsections. *For each main learning method used we will emphasize the main concept and structure behind the method and how they have been adapted for utilization in Morph.*

Positional Credit Assignment and Weight-Updating

Each game provides feedback to the system about the accuracy of its evaluations. The first step is to use the outcome of the game to improve the evaluations assigned to positions during the game. This is done using temporal-difference (TD) learning [Samuel, 1959; Samuel, 1967; Sutton, 1988a]. Once new positions have been given evaluations, the weights of patterns that were used to evaluate the positions are moved in the direction of the desired or "target" evaluation. Using a temperature as in simulated annealing allows each pattern to move at its own rate, based on the number players, e.g. "up 2 pawns and down 1 rook," "even material," etc.

Much of the following discussion will refer to graph patterns alone. But it should be understood that material patterns and graph patterns are processed identically by the system.

Along with each pattern is stored a weight in [0,1] as an estimate of the expected true minimax evaluation of states that satisfy the pattern. In order to determine the utility of a pattern and whether it should be retained other statistics about patterns are maintained.

2.2 Associative Pattern Database

The pattern database expedites associative retrieval by storing the patterns in a partially-ordered hierarchy based on the relation "subpattern-of" ("more-general-than"). Thus, at one end of the hierarchy are simple and general patterns such as "white bishop can take black pawn" and at the other end are complex and specific patterns such as those associated with full positions.

Empirically, it has been shown that on typical databases using a simple associative retrieval algorithm [Levinson, 1984; Levinson, 1991a] only a small fraction of the patterns in the database need be considered to evaluate a position. An even more powerful retrieval algorithm is being developed [Levinson and Ellis, 1991].

2.3 Evaluation Function

The evaluation function takes a position and returns a value in [0,1] that represents an estimate of the expected outcome of the game from that position (0=loss, .5=draw, 1.0=win). Although heuristic evaluation is not directly a learning technique it it is accessed by virtually every learning strategy in the system; thus having an integral role in the learning process.

Morph uses two different evaluation functions. One is used for evaluating boards during play (for move selection), and the other one is used for evaluating boards during updating (as a basis for learning). Both evaluation mechanisms apply the following procedures:

1. Take a position as input.
2. Determine all of the most specific patterns that match the position.
3. Apply a specific function to all the patterns.
4. Return the result of step three.

Thus, the two evaluation functions differ only in the third step, the function applied to the set of matched patterns.

The evaluation function that evaluates boards during play uses a function for step three that has the following properties, where w_1 and w_2 are weights of patterns:

1. if $w_1 > .5$ and $w_2 > .5$ then $f(w_1, w_2) > max(w_1, w_2)$ unless either w_1 or w_2 is 1 then $f = 1$.
2. if $w_1 = .5$ then $f(w_1, w_2) = w_2$
3. if $w_1 = \alpha$ and $w_2 = 1 - \alpha$ then $f = .5$
4. if $w_1 < .5$ and $w_2 < .5$ then $f < min(w_1, w_2)$ unless either w_1 or w_2 is 0 then $f = 0$.
5. if $w_1 > .5$ and $w_2 < .5$ then $w_2 < f < w_1$. f is more towards the most extreme weight.

The entire function is displayed in Figure 2.2. This binary function is applied iteratively to all matching patterns.

These mathematical constraints to the evaluation function have a strong intuitive backing. For instance, rule 1 states that if two patterns suggest that a position is good ($> .5$) the board should then be considered better than either of them alone. .5 is the weight that is assigned to a pattern that does not have any positive or negative connotation. Patterns with weight 0 suggest strongly

2. System Design

Here we give a description of the Morph playing and learning system. In particular we will emphasize how diverse learning methods are being applied for experience-based learning.

Morph makes a move by generating all legal successors of the current position, evaluating each position using the current pattern database and choosing the most favorable position. After each game patterns are created, deleted and generalized and weights are changed to make evaluations more accurate (in the system's view) based on the outcome of the game. Patterns are deleted periodically if they are not considered useful. All performance results in Section 5 are based on training against GnuChess Level I, a program that is stronger than at least 60% of tournament players.

In this section the chess system design is described in detail. The model has four basic parts, which are described in the following subsections.

2.1 Patterns and their Representation

The basic unit of knowledge to be stored is a pattern. Patterns may represent an entire position or represent a boolean feature that has occurred in one or more positions and is expected to occur again in the future. We purposely choose a uniform representation scheme to maximize the potential cross-fertilization of pattern knowledge and to simplify the implementation and processing requirements. Positions are represented as unique directed graphs in which both nodes and edges are labelled [Levinson, 1991b]. Nodes are created for all pieces that occur in a position and for all squares that are immediately adjacent to the kings. The nodes are labelled with the type and color of the piece (or square) they represent. For kings and pawns (and also pieces that

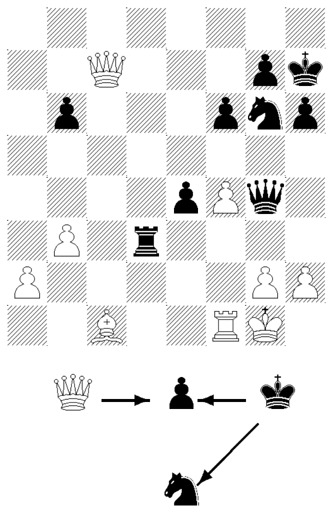

Figure 2.1: A board and subgraph (pattern) that resides within it.

would otherwise be disconnected from the graph) the exact rank and file on the board in which they occur is also stored. The exact squares of kings and pawns allows the system to generate specific endgame patterns and patterns related to pawn structure. Edges represent attack and defend relationships between pieces and pawns: Direct attack, indirect attack, or discovered attack (a defense is simply an attack on one's own piece). At most one directed edge is assigned from one node to another and the graph is oriented with black to move. Patterns come from subgraphs of position graphs (see Figure 2.1) and hence are represented the same way except that they may label any node with an exact or partial square designation. The representation has recently been extended to include other squares as nodes besides those adjacent to kings.

There are actually two types of patterns stored in the Morph system. In addition to the above mentioned graph patterns , Morph stores "material" patterns: vectors that give the relative material difference between the

1. Introduction

Currently a tremendous amount of computation is wasted because most computers do not learn from their experience. Consider a system that after 10 hours of calculation designs a synthesis plan for an organic molecule X [Levinson, 1991b] . Now isn't it silly that if we give it molecule X again or more to the point something similar to X another 10 hours of computing are required? Thus, we can expect in the future that increasing responsibility for integrating experience will be given to systems. We have argued elsewhere that one way that search experience can be compiled is as a set of patterns coupled with weights("pw"s) [Levinson, 1989a]. Patterns are features used for evaluating states and weights indicate the significance of the patterns (positive or negative) and by how much. To give the system responsibility for creating and applying its own pws, pattern learning methods must be coupled with weight updating methods and heuristic evaluation. Furthermore, with little supervised feedback from the environment, methods for assigning values to states (credit assignment) must also be present. The coordination of such methods in a beneficial way is the topic of this paper.

Interestingly, despite the recognition of the criticality of search and the high-costs that are paid to achieve it,only a little effort has been applied to getting chess systems to utilize previous search experiences in future searches [Scherzer *et al.*, 1990; Slate, 1987]. Thus, excluding random factors from the system (or human intervention), one can expect a chess system to play exactly the same way against the same series of moves, whether it has won or lost, and take the same amount of time to do so!

We have built a chess system, Morph [Levinson, 1989b; Levinson and Snyder, 1991] that is required to learn from its experience with little supervised training, little domain knowledge and little search. Obviously, with such little assistance, Morph requires a powerful learning component. Indeed, Morph uses a number of machine learning techniques that have been useful in other settings to learn patterns and their significance. Patterns denote configurations of interaction between squares and pieces. A uniform heuristic evaluation method combines the significances in a given position to reach a final evaluation for that position. The learning mechanism combines weight-updating, genetic algorithms, explanation-based generalization, structural induction, annealing and temporal-difference learning modules to create, delete, generalize and evaluate graph patterns. An associative pattern retrieval system organizes the database of patterns for efficient processing.

Method Integration for Experience-Based Learning

Jeffrey Gould and Robert Levinson

UCSC-CRL-91-27
August 28, 1991

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

This paper describes how a variety of machine learning methods can be combined synergistically to produce an adaptive pattern-oriented chess program. Major aspects of the following machine learning methods are used: temporal-difference learning, simulated annealing, genetic algorithms, explanation-based generalization, structured concept induction and heuristic evaluation. The need for these methods comes from the research constraints placed on the chess system. The system, "Morph", is limited to using just 1-ply of search, little domain knowledge and no supervised training. Thus, the system is responsible for finding a useful set of features (patterns) for evaluating states and for determining their significance (in the form of a weight). To get the learning methods to cooperate effectively some design constraints normally associated with these methods need to be relaxed. The paper also argues that a benefit of a multi-strategy viewpoint is that new research ideas arise from the multiple perspectives.

Keywords: machine learning, integrated learning systems, temporal-difference learning, genetic algorithms, pattern induction.