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

[37] 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.

[38] 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.

[39] 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.

[40] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(7):256–275, 1950.

[41] H. A. Simon and K. Gilmartin. A simulation of memory for chess positions. *Cognitive Psychology*, 5(1):29–46, 1973.

[42] S. Skiena. An overview of machine learning in computer chess. *Intern. Computer Chess Assoc. Journal*, 9(3):20–28, 1986.

[43] 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.

[44] George Steiner. *Fields of force; Fischer and Spassky at Reykjavik.* Viking Press, New York, 1974.

[45] Richard Sutton, editor. 1991.

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

[47] 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.

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

[49] Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 1991.

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

[51] C. S. Wilcox and R. A. Levinson. A self-organized knowledge base for recall, design, and discovery in organic chemistry. *Artificial Intelligence Applications in Chemistry*, (306), 1986.

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

[53] 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.

[54] A. L. Zobrist and D. R. Carlson. An advice-taking chess computer. *Scientific American*, 228:92–105, June 1973.

[11] J. Gould and R. Levinson. Method integration for experience-based learning. Technical report, University of California, Baskin Center, Santa Cruz, 1991. To appear.

[12] E. Hearst. Man and machine. In P. W. Frey, editor, *Chess Skill in Man and Machine.* Springer-Verlag, 1977.

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

[14] W. James. *The Principles of Psychology*, volume 2. Henry Holt and Co., New York, 1890. Republished by Dover Publications, 1950.

[15] H. Kaindl. Towards a theory of knowledge. In D. Beal, editor, *Advances in Computer Chess 5*, pages 159–185. Pergammon, 1989.

[16] G. Kasparov. *Test of Time.* Pergamon, Oxford, 1986.

[17] G. Kasparov. *Child of Change.* Hutchinson, 1987.

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

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

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

[21] 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.

[22] 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.

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

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

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

[26] 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.

[27] 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.

[28] 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.

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

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

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

[32] 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.

[33] 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.

[34] H. Pfleger and G. Treppner. *Chess: The Mechanics of the Mind.* The Crowood Press, North Pomfret, VT, 1987.

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

  - What happens when Morph is trained on these games? What if the games are presented in reverse order? Can we also explain Bobby Seltzer's development in terms of learned patterns? Can his "creative" moves be explained in terms of past experiences?
  - We have a similar dataset of Bobby Fischer's games? Which Bobby does Morph learn best from: Fischer or Seltzer? A very long term goal is to get Morph to be strong enough to emulate a match between these two (or Fischer and the current World Champion).

- Of critical importance is the determination of more compelling domains beyond chess to which these methods can be applied. One such domain would seem to be organic synthesis [23,51]: through experience a system could learn which types of molecules are more easily or cheaply made and guide synthesis pathways in this direction. Similar ideas may also apply to automatic theorem proving.
- Finally, we believe that the "psychology of Morph" is another direction that is worth pursuing. For example, we have witnessed signs of depression at times: many patterns are evaluated negatively, or Morph is afraid to try new things or to try things that failed early in its training. Likewise, from time to time Morph plays extremely aggressively and sometimes recklessly. Where does this tendency come from? Perhaps, it is through understanding the mathematical principles behind the effect of experience on decision-making and future experience that better therapies can be developed. Can Morph's creativity be "encouraged?".

7 Acknowledgements

We are indebted to those researchers whose methods have been adapted for Morph, Morph's programmers, the encouragement of friends and family, NSF grant IRI-8921291 and to experience itself.

References

[1] A. Avni. *Creative Chess.* Pergamon, Oxford, 1991.

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

[3] M. Botvinnik. *Computers in Chess.* Springer-Verlag, 1984.

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

[5] 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.

[6] A. D. de Groot. *Thought and Choice in Chess.* The Hague, 1965.

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

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

[9] M. Fox and R. James. *The Complete Chess Addict.* Faber and Faber, London, 1987.

[10] 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.

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 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 [7] also applies a semantic approach but to multiple simultaneous game domains.

6 Conclusions and Directions

- As we finish preparing this document, Morph continues to improve. We believe that there are some changes that can make learning significantly faster. The idea is to not allow Morph to use a pattern in an evaluation until it has been updated a certain number (e.g. 25) times. In this, way, Morph's play and evaluation will not be severely hampered by unripe patterns. With this risk out of the way, Morph can be given many more patterns per game than it currently receives and, hence, the learning curve should be steeper. Each pattern continues to live a life of its own, but must reach a certain maturity before influencing learning and decision-making.
- Now that Morph has reached a level in which it is easily trainable, specific training issues can be explored:
  - For instance, is it more useful for the system to train against GnuChess Level II or to first train against GnuChess Level I before moving on to level II? Intuition might suggest that the latter strategy is more efficient, especially given the faster response time of level I, but this needs to be validated empirically.
  - Our research group has obtained the games of (now 16-year old) chessmaster Bobby Seltzer described here by his father:

    > I have a unique set of chess games that might be of value to research in artificial intelligence or to developers of chess-playing software. I have been saving my son Bobby's games as word processing files since the very first rated game he played in Oct. 1984. (I only missed two early scholastic ones, where neither player recorded the moves.) There are now 680 rated games on file, in addition to miscellaneous simul and postal games – a continuous record from 9-year-old raw beginner to 14-year-old master. I believe that analysis of these games could provide valuable information about how one can learn and improve rapidly at chess. [...] In the best of all possible worlds, I could foresee a valuable collaboration. Analysis of Bobby's games, both past and future, could provide insight that helps in the further development of chess-playing software and artificial intelligence in general. [...] And even games played between Bobby and an experimental chess program (perhaps over the Internet) could prove valuable for both.
    >
    > All too often, artificial intelligence/chess research is presented to the public as a battle of man vs. machine. I think collaboration is a much more useful model.

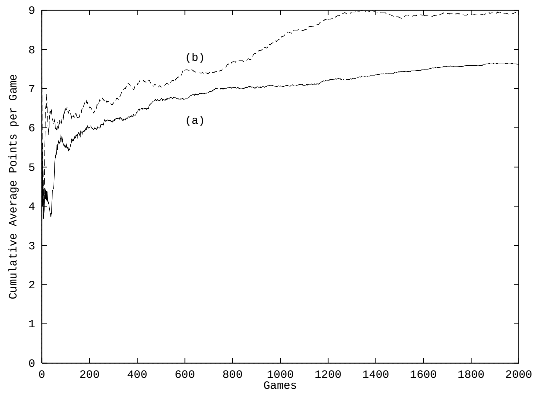


Figure 6: Cumulative Average of two versions of Morph.
Version (a) adds the split evaluation function on top of reverse node ordering Morph. Version (b) adds annealing on top of Version (a).

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.[53] 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 [48]. 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 [19].

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 [29,32] have relied on pre-classified sets of examples or examples that could be classified by a complete game-tree search from the given position [50]. The symbolic learning work by Flann [8] 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 [47] 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 [30], building on Pitrat's work [35], 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 [52], which, also building on Pitrat's work, used pattern knowledge to

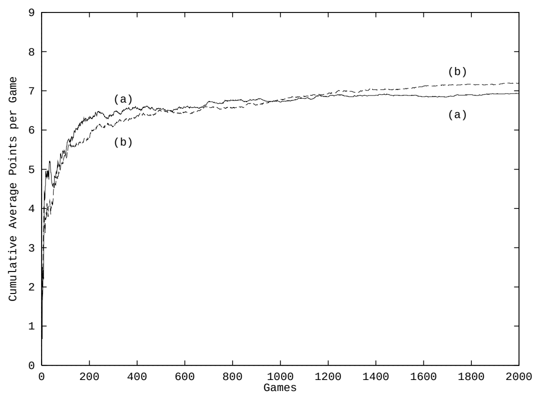


Figure 5: Cumulative Average of two versions of Morph.
Version (a) is the basic Morph in existence three months ago. Version (b) adds the reverse node ordering pattern addition scheme to Version (a).

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.

4.2 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 5a shows a basic Morph, Figure 5b shows the result of adding reverse node ordering, Figure 6a shows the result of splitting the evaluation function, and Figure 6b shows the result of adding annealing.

5 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 [11] 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 [30]. The use of explanation-based learning is one similarity: Morph specifically creates patterns that are "responsible" (as preconditions) for achieving future are consistent and credible [25]. (see Table 1 for a database after 106 games). The weights correspond very well to the traditional values assigned to those patterns. These results reconfirm other efforts with TD learning [4] and perhaps go beyond by providing a finer grain size for material.

Material Pattern					Statistics				Trad.
Pawn	Knight	Bishop	Rook	Queen	Weight	Updates	Variance	Age	Value
0	0	0	0	0	0.455	2485	240.2	106	0
0	0	–1	0	0	0.094	556	7.53	86	–3
0	0	+1	0	0	0.912	653	11.19	88	+3
0	+1	0	0	0	0.910	679	23.59	101	+3
0	–1	0	0	0	0.102	588	17.96	101	–3
0	0	0	–1	0	0.078	667	3.56	103	–5
0	0	0	+1	0	0.916	754	5.74	103	+5
+1	0	0	0	0	0.731	969	22.96	105	+1
–1	0	0	0	0	0.259	861	13.84	105	–1
0	0	0	0	+1	0.903	743	5.68	105	+9
0	0	0	0	–1	0.085	642	3.12	105	–9
0	0	–1	+1	0	0.894	10	0.03	55	+2
0	0	–2	0	0	0.078	146	0.53	71	–6
+1	0	–1	0	0	0.248	26	2.35	73	–2
0	+1	–1	0	0	0.417	81	4.48	82	0
–1	0	–1	0	0	0.081	413	2.14	92	–4
0	–1	+1	0	0	0.478	84	5.72	82	0
+1	0	+1	0	0	0.916	495	3.56	88	+4
0	0	+2	0	0	0.924	168	0.66	91	+6

Table 1: A portion of an actual Morph material database after 106 games.
The columns headed by pieces denote relative quantity. The weight column is the learned weight of the pattern in [0,1]. Updates is the number of times that this weight has been changed. Variance is the sum of the weight changes. Age is how many games this pattern has been in the database. Value is the traditional value assigned to this pattern. Note that a weight of 0.5 corresponds to a traditional value of 0. The entire database contained 575 patterns.

- 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.

4.1 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 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.

3.4.3 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 [30]. 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.

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

4 Performance Results

To date Morph has only one victory against GnuChess, and has obtained over 20 draws via stalemate, repetition of position and the 50-move rule. Despite the relative 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 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 [5].

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.

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

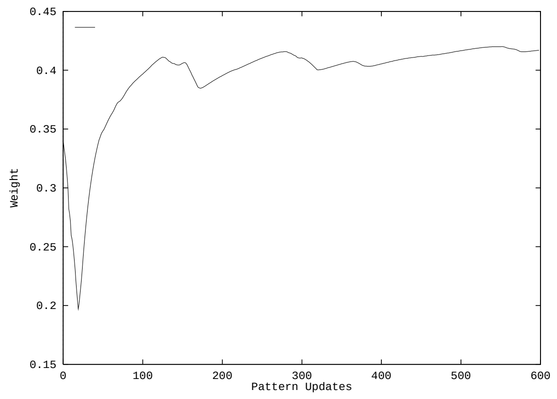


Figure 4: This graph depicts the weight change of the same pattern in Figure 3 in a system that has simulated annealing added.

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 [28,31,33,36] 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.

Genetic Operators

Genetic algorithms [13] are a means of optimizing global and local search [10]. 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

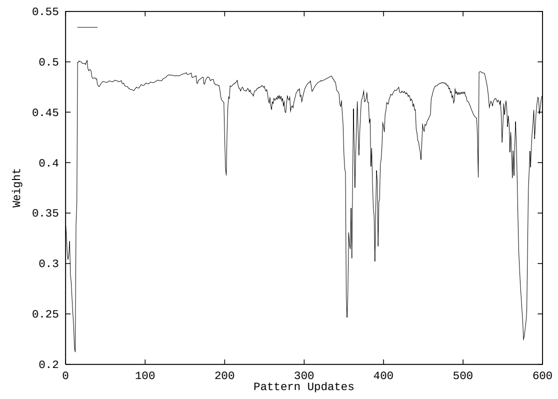


Figure 3: This graph depicts the weight change of the material pattern "down 1 bishop" in a system without simulated annealing.

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 3 and Figure 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 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.

3.4.2 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:

```
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 [37,26,48,49]. 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 in statistical mechanics [5]. 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 [27] 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 [18] adapted annealing to computer science, by finding information analogies for their physical counterparts [5]:

- 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 [46]: 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.

3.4.1 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 [37,38,46]. 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 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 [46]. 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;
```

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

Figure 2: Move selection evaluation function.

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 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 unfavorable 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.

3.4 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.*

whether it should be retained other statistics about patterns are maintained.

3.2 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 [20,22] 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 [24].

3.3 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. 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

In Morph, positions are represented as unique directed graphs in which both nodes and edges are labelled [23]. 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 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 3.1) 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.

A similar representation scheme has successfully been used to represent chess generalizations [54] and to produce a similarity metric for chess positions[23,3].

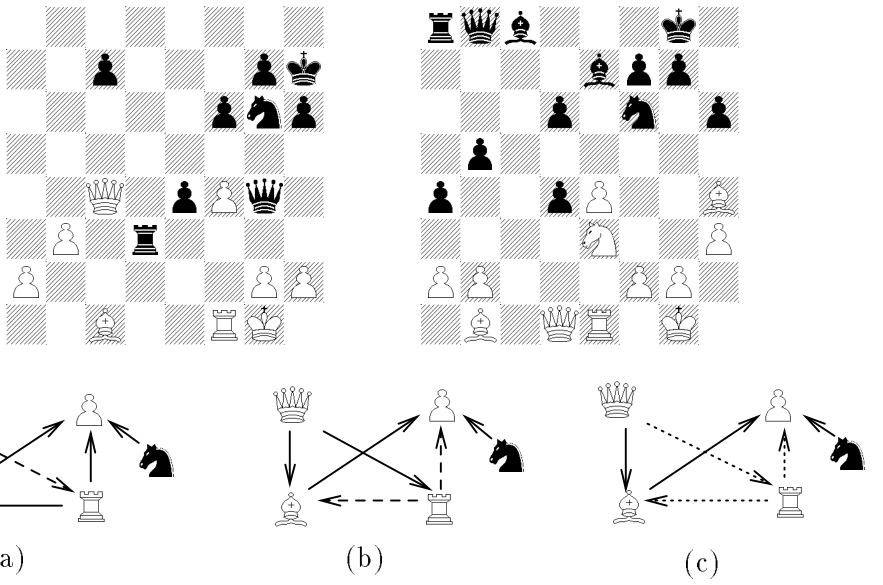


Figure 1: A generalization derived from two different chess positions. (a) is the subgraph derived from the board on the left and (b) is the subgraph from the board on the right. The solid edges correspond to direct edges between pieces and the dashed edges correspond to indirect edges. Graph (c) is the generalized graph derived from (a) and (b) in which the dotted edges have been generalized to generic "attacks."

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 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

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

3.1 Patterns and their Representation

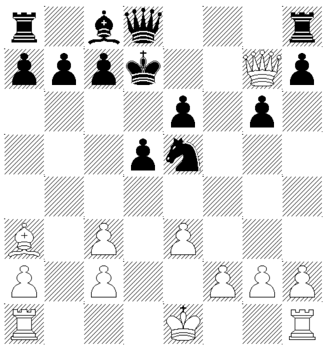
To fully exploit previous experience, the method chosen to represent each experience is critical. An ideal representation is one that uses features that are general enough to occur across many experiences (positions, for chess), but are such that their significance is invariant across these experiences. How to construct such a representation for chess is not obvious. The straightforward approach of using a standard chess board representation is not powerful enough since there are over 10^{40} possible chess positions [40]. In fact, after just three moves for each player, there are over 9 million possible positions [9].Further, a pattern such that "white bishop can capture black rook" has nearly the same significance regardless of where on the board the white bishop and black rook are located, and one would like to choose a representation that exploits this information.

In designing a representation for encoding chess experience, inspiration can be gained from the writings of top chess players who seem to view a chessboard in a manner far different from most people!

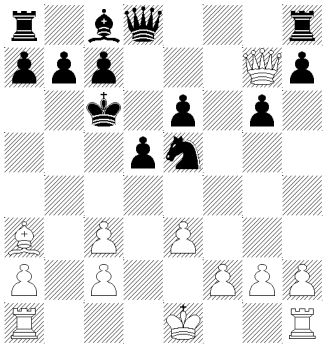
> ...It can happen that the pieces as though receive an invisible impulse from the players, come alive, and begin to live their own lives. And when the energy invested by both sides reaches a critical point, the game begins to develop according to laws unknown to anyone, and it is no longer possible to control its course. The flood of concentrated chess thought washes away the usual conttours of the board, and after twisting the pieces into violent pandemonium, it crashes down into the depths of chess art. And however the game concludes, chess never loses out! It is not easy to understand all the intricacies of such a game even in subsequent analysis and it is difficult to talk about it without disrupting the picture of a grandiose encounter. *World Champion Garry Kasparov [16]*

> What is it that a master 'sees' when he looks at the board that is so different from what other players see? An old French master said: 'I see the chessboard as one sees the street on which one walks without paying much attention to it; when one opens one's wardrobe one knows where all the things are in spite of the fact that one does not see them. The same applies to the moves one makes on the chessboard. *Garry Kasparov [17]*

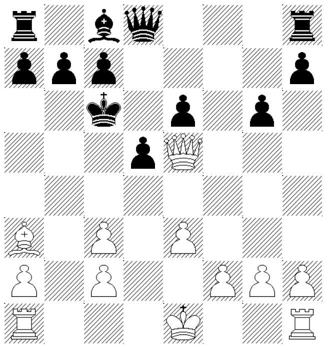
> The great chess player does not see squares and pieces as discrete units, or even as abstract counters. He internalizes a very special sense of "fields of force," regions characterized, differentiated by the fact that certain events can or cannot take place in them. What matters, thus, is not the particular square, or even piece, but a cluster of potential actions, a space of and for evolving events...Like the conductor who keeps in ordered mental grasp the enormous array of notes and tempo markings set out vertically and horizontally on the page of a Wagner or a Mahler score, the chess master experiences and retains relations of motion, "magnetic fields" of conjunction or constraint that wholly transcend the single unit. In memorizing the positions of thirty-two pieces...the chess mind does not photograph rows of single counters. It somehow encodes essential groupings of meaningful force (as we do in scanning a spoken or written text) and commits to memory the articulate sinews beneath the skin. *George Steiner [44]*



Morph follows through - attacking king, rook and unprotected knight simultaneously.



GnuChess must move the king to here (or interpose the knight) or else the rook(and more) is lost.

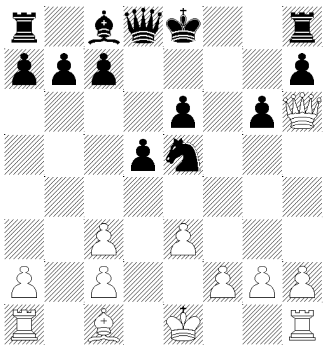


Morph regains the sacrificed piece and continues, with the black king exposed.

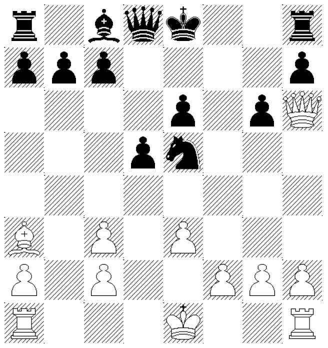
3 System Design

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.

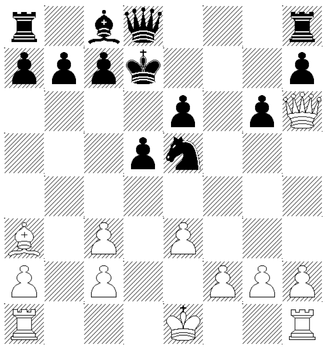
Morph makes a move by generating all legal successors of the current position, evaluating each position using the current pattern database and choosing the most favorable position. After each game patterns are created, deleted and generalized and weights are changed to make evaluations more accurate (in the system's view) based on the outcome of the game. Patterns are deleted periodically if they are not considered useful.



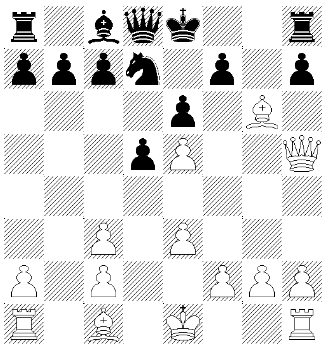
GnuChess captures the king pawn and appears to prepare a nice reply to Q-N7.



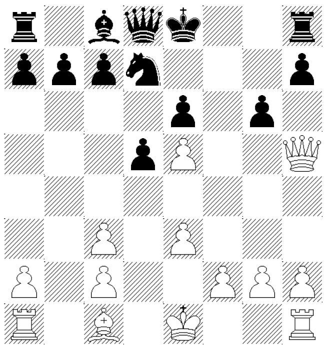
By playing this move that attacks two squares around the black king, Morph threatens to win the white rook with Q-N7.



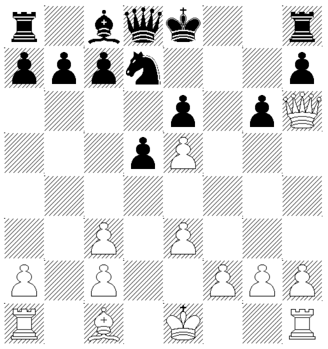
GnuChess's king, feeling the heat, moves. Probably much better was N-B7 attacking the queen and simultaneously protecting knight and rook. Now Q-N7 wins at least the knight.



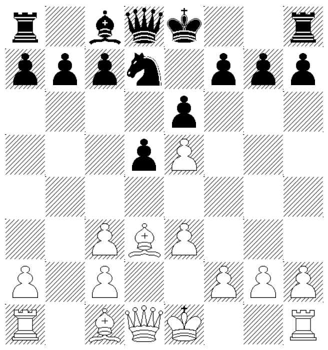
A surprising and risky maneuver by Morph that is probably not necessary for what follows. This move uncovered a subtle bug in the system design: Morph may have played it since the rook pawn is pinned and since it only uses "most-specific patterns" in evaluations the other "pawn can take bishop" edge was ignored.



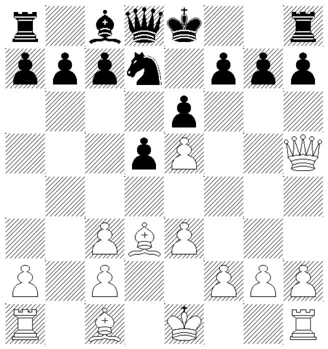
GnuChess captures the bishop and again attacks the queen.



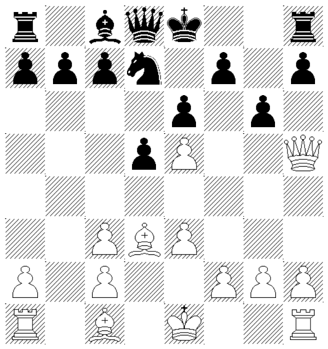
Morph moves the queen up. This may have been in preference to Q-KB3 since the move played also attacks two pawns. Now GnuChess can not castle, and Morph threatens Q-N7 winning a pawn and continuing the attack.



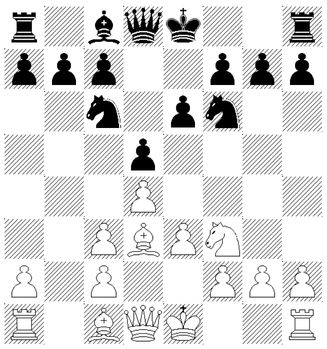
GnuChess retreats the knight.

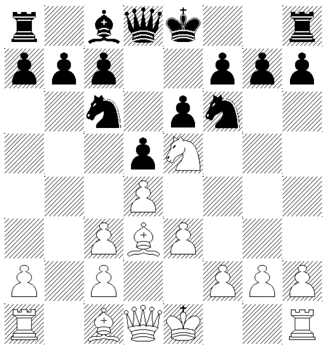


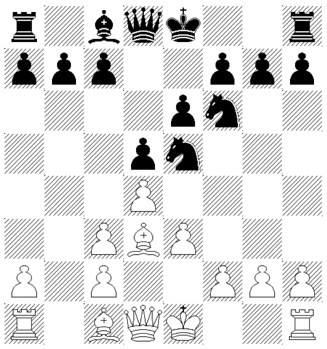
With the knight no longer protecting this square, the queen is in a number of favorable patterns on king rook 5.

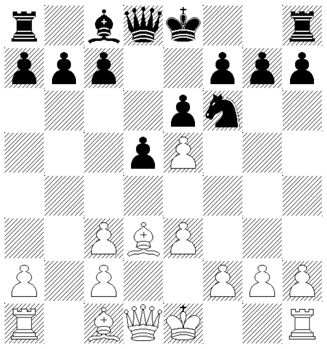


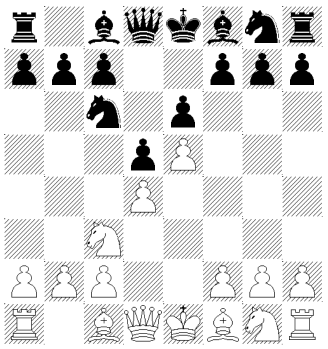
GnuChess forces the queen to move so that he can win the king pawn with his knight. But this pawn move, weakens the kingside...


After 6 moves, Morph has an even game. Now he gets aggressive.

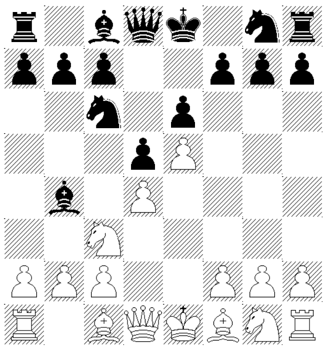

Morph likes this place for the knight - it attacks two squares around the opposing king.


GnuChess captures the knight.

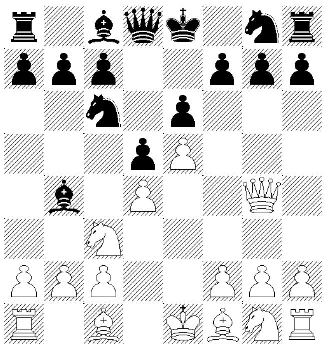

Morph recaptures.



Morph develops his knight toward the center - a pattern that it finds favorable.

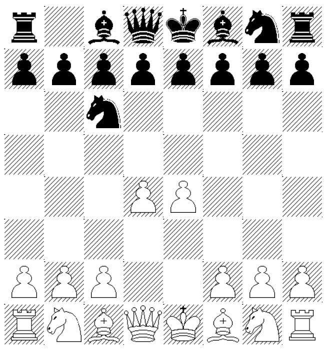


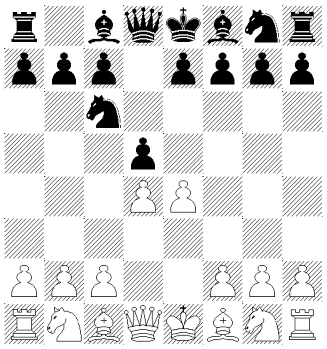
With this, GnuChess initiates the French Defense Winawer Variation.

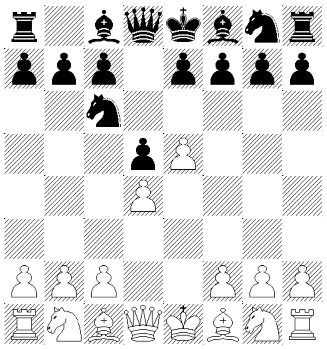


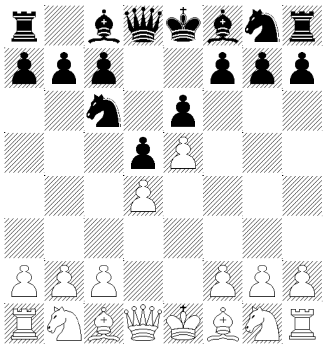
Morph's wild queen sortie puts us in one of the most hotly debated variations in recent master play. Evidently, the opportunity to attack two black pawns (one unprotected) simultaneously looks too good to resist...

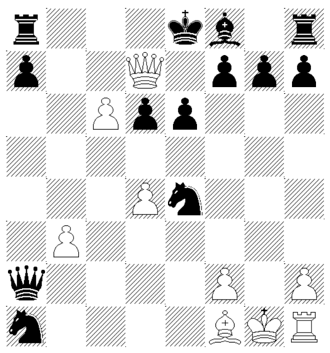
2.4 Morph plays a sacrificial attack.


Morph builds the classic pawn center.


The game is now a Nimzowitsch Defence.


Morph plays the standard move here.

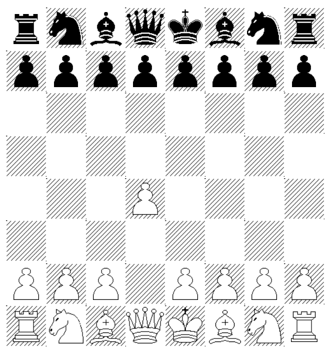

GnuChess now transposes into the French Defense.



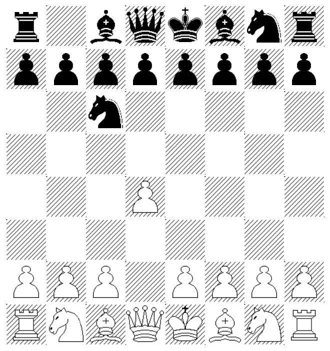
Morph finishes the job and wins its first game ever against its trainer (after 3000 games of training). He places extremely high value on having an unattacked, protected queen adjacent to the opponent's king.

2.3 Morph plays a book variation

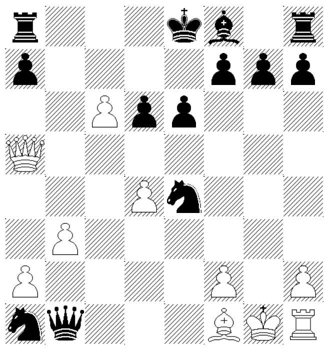
Neither Morph or GnuChess is allowed to use an opening book. Still, Morph sometimes finds his way into a book variation. Surprisingly, the following example occurred after just 27 games of training:



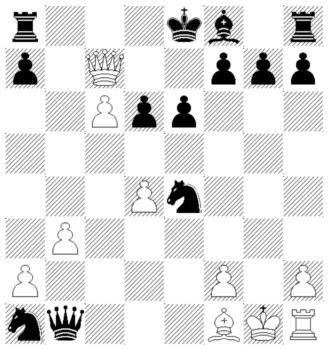
Morph (white) opens with his queen-pawn.



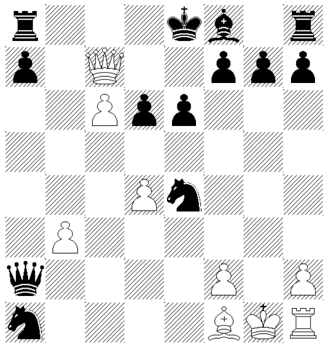
This unusual move is regularly employed by GnuChess.



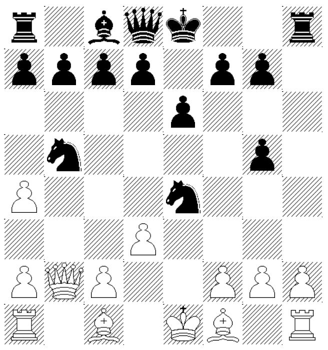
Perhaps feeling overconfident, GnuChess grabs one more pawn.



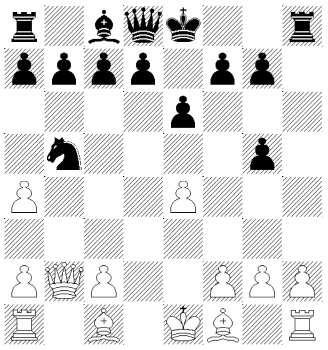
Morph improves the position of his queen and threatens mate. He selected this move because of patterns that encourage attacking squares around the opposing king. From here the queen attacks four such squares. Note that it correctly considers this move to be more valuable then protecting the queen rook pawn.



In an unusually poor move for a computer, GnuChess fails to prevent the mate, but instead threatens mate of its own. (Although, by preventing mate with B-K2,N-KB3 or R-Q1 GnuChess can go on to win, it is worth noting that each of these moves probably loses material to Q-N7, followed by P-QB7 for white)

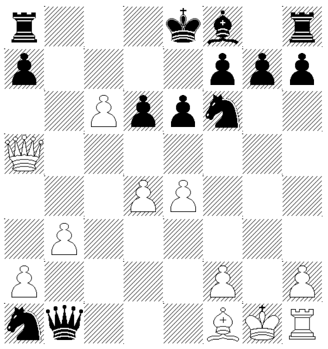


GnuChess realizing now that it must lose a piece, takes the king pawn to salvage something.

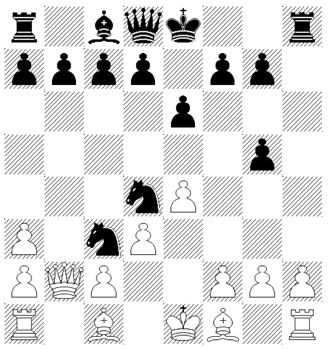


Morph recaptures the knight and continues the attack...

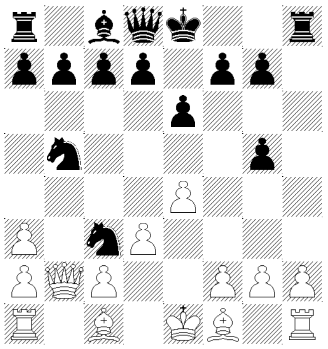
2.2 Morph snatches victory from certain defeat



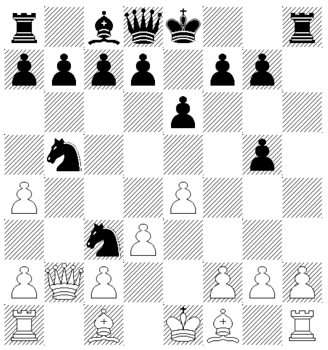
In this position, GnuChess (black) to move, has an overwhelming material advantage.



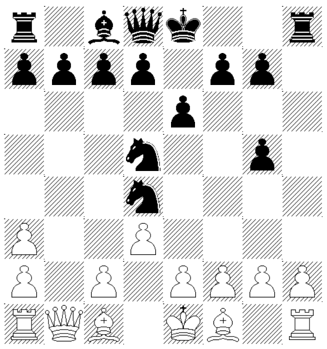
Morph capitalizes. He moves his queen out of the attack and creates a pattern in which he is attacking one knight directly and the other one through "discovery" as well as preventing the potential fork on his King-Bishop-2.



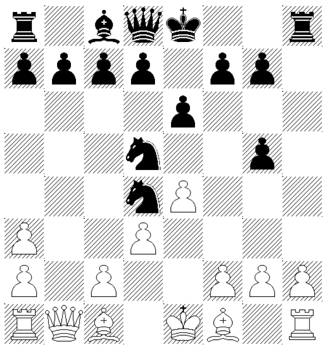
Perhaps GnuChess thinks he is "out of the water". Now both knights protect each other. (Better, perhaps, was Q-KB3 since QxN loses to N-KB6ch, but then B-Q2 apparently leads to the same result).



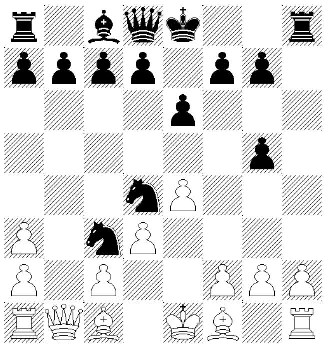
But Morph has learned to attack defenders! The queen simultaneously attacks both knights, and the pawn adds its own attack.



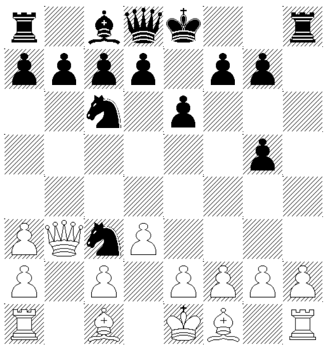
GnuChess moves in the other knight.



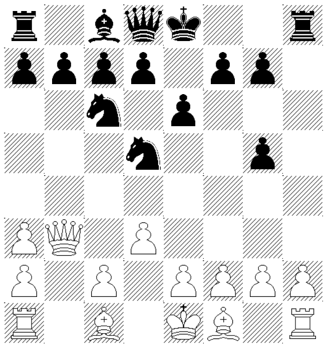
Morph moves his king pawn to attack the knight on queen 5. Note that he could instead have attacked the knight on queen 4 with this pawn, but favors this move because he has learned that it is desirable to have a pawn on king 4.



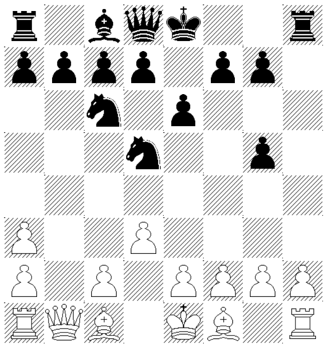
The crucial mistake. GnuChess again brings this knight in to attack the queen. Unfortunately, against optimal play one of the knights will be lost.



Morph moves his queen and simultaneously attacks the undefended knight as well as directly and indirectly attacking two more black pawns than before.



The knight retreats to its only safe square.



Morph's queen returns to its original square. This is probably a weak move. Its one redeeming feature is that it invites black to repeat the position and perhaps draw the game (if any position repeats three times with the same player to move). However, Morph is unfamiliar with this rule.

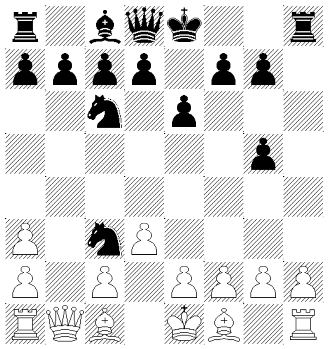
1.5 Outline of paper

In Section 2 specific examples of creativity displayed by Morph are given. Section 3 describes the design of the Morph system. In particular, the focus of Section 3 is on the variety of learning methods that are being used in Morph. To adapt these methods for Morph some design constraints usually associated with them have been relaxed while retaining the essential conceptual framework of each method. It is the learning components and their integration [11] that allow the system to compile its experience into a form that can be used to produce good chess moves without search. In essence, Morph takes advantage of the many games (searches) that have been played previously. achieved. Section 4 describes the relationship of Morph to other research in machine learning, chess, and adaptive game-playing systems. Section 5 gives general performance results with Morph. Finally, Section 6 makes some concluding remarks and gives future directions.

2 Creative Moves Made by Morph

In this section several examples of creative play by the Morph chess system are described. All moves were made against GnuChess, a traditional chess program that is better than at least 60 percent of tournament players. On most moves GnuChess searches 4 or 5 ply deep. We classify the following moves by Morph as creative because they seem to reveal a deeper insight into the dynamics of the position than can normally be found in 1-ply of search. We acknowledge that much deeper instances of creativity than these are possible in chess [1] and hope to see Morph's moves in the coming years grow in that direction.

2.1 Morph wins a piece



Morph (white), to move, finds himself down a minor piece with his queen attacked. What his queen is doing on that unusual square, is perhaps, already an indication that Morph does not view the game as most of us do!

Or from G.Steiner:

> ...The tournament player will draw on an ever growing ever more analytically detailed body of annotated precedent. He must have in orderly mental reach an inventory of previous end games in which analogous positions, analogous combinations of pieces have turned up. Under time pressure, he cannot hope to find optimal strategies across the board but must rely for his tactical choices on a recognition of previously explored lines and configurations. From the middle game onward, the expert player is projecting, by inner vision, the terminal situations he must aim for or avoid. He knows, without even thinking, that bishops of opposite colors lead to a draw, that the side whose king first reaches the central squares has a large advantage, that split pawns are a nagging weakness, that a posture on the enemy's seventh rank is often paralyzing, that major pieces ought normally to be behind advancing pawns. He knows in what position a lone king can force a draw against king and pawn. But, above all, he remembers the denouements of previous master encounters. [44]

Finally, from A.D. de Groot [6]:

> Incorporating experience into the program, in one degree or another [...] is the very core of the simulation problem.

1.4 Is Morph creative?

In what sense, can a move generated by the pattern-oriented method be considered creative? First of all note that the description of the move itself can not be considered creative since the legal moves in chess positions can be generated by a straightforward algorithm. A move can only be considered creative if it is not only good, but shows in some sense imagination and originality. Now since Morph does not perform game-tree search the reason for its moves can not be written as a concrete variation or subtree, but are based on the creation of a pattern or set of patterns that are considered favorable. It is the difference of these patterns from those that would normally be recognized in the position that can give the move its creative character. The imaginative act is the recognition of this pattern in the future position and the right assessment of its significance. Such a move, if actually good, would be considered by humans to show *great insight.* The graph-based representation scheme used by Morph coupled with its associative recall mechanism may allow it to bring to bear past experience in a way that is missed by humans. Further, since little knowledge of the game and semantics of the graphs has been given the system it is not far-fetched to say that the machine is playing *intuitively* - for it seems to *know* without reasoning or having been explicitly taught.

It is useful to consider the history and influence of a pattern in the Morph system. A pattern is used whenever it is a "most-specific" pattern in a move considered by Morph. A pattern is updated whenever it is in a position moved to by Morph. Each pattern then is a unique store of knowledge specific to those positions it is used in. Any given new position, then, is evaluated based on a set of patterns that themselves may not be unique, but their combination for the given position and the move suggested very well may be. In Morph, each pattern lives a life of its own - with its own learning rate. This is accomplished through simulated annealing (see Section 3.) and allows Morph to adapt first to frequently occurring situations before pursuing finer details.

these previous experiences that have contributed to the patterns and weights that are being used to evaluate P.

Once a chess graph is constructed from a game board, the semantics of the nodes and edges in the graphs are unknown to the system. The only information a pattern contains as far as the system is concerned is the significance (weight) that has been attached to the pattern , in no place after pattern creation do we special case pieces or edges. Such a syntactic approach to the learning of search knowledge is substantially different from many of the traditional symbolic AI approaches to chess and to the learning of control knowledge.

To determine its next move, Morph performs only one ply of search: From the current position it generates all legal successors, evaluates these positions using its pattern database and moves to the position that is considered most favorable.

1.3 Morph and cognitive models

The system, being pattern-oriented , adaptive and not relying heavily on search is more consistent with psychological models of human chess performance than existing chess systems. We quote directly from S. Skiena [42]:

> The first psychological study of chess and perception dates back to Binet in 1894, but the major work in the field is Thought and Choice in Chess by Adrian de Groot(1965), who analyzed the thought processes for chess players of various levels of ability on difficult positions.
>
> The experiments surprisingly show that grandmasters do not search the game tree deeper than less talented players, but invariably select only good moves for further study. This is evidenced by performance on blitz games, where the player does not have time to explore the tree to any depth. Play is still at a very high level, suggesting that, in human chess, pattern recognition plays a more important role than search. [These experiments have since been repeated by de Groot and verified by others as well, [12,34]] Masters explicitly search a tree of about 50 nodes before selecting a move, using perceptual mechanisms to prune the tree to eliminate the need to consider other possibilities. Even with search, the evaluation of leaf nodes is performed as a perceptual process, with players rating them subjectively rather than explicitly assigning a score.
>
> Instead of some form of chess aptitude, [1], what distinguishes a player as a master is the accumulation of problem-specific knowledge. At least five years of intense chess study is needed to become a grandmaster, and even the cases of child prodigies show the periods of total dedication necessary to achieving excellence. This suggests that learning plays an important role in chess play, and that attempts to codify this knowledge into a few expert-system rules are probably doomed to failure.
>
> Clearly, chess masters perform on the level they do because of an internal library of patterns which they apply to suggest promising moves. Simon and Gilmartin [41] estimated the number of patterns for a grandmaster at about 50000. When then compared to the astronomical number of possible board positions or nodes in the complete game tree this is a tractable number, and thus suggests an avenue through which computers can improve their play.

[1]This clause seems false at the high echelon of play where exceptional creativity is clearly evident.

Given this hypothesis of the power and generality of the experience-based problem-solving mechanism, how do we go about implementing or simulating it on a computer? First there must be a means of encoding experience. For this we choose "pattern-weight pairs" ("pw"s) [21] where patterns are boolean features that can occur in one or more states (experiences) and weights reflect the desirableness of that pattern. The system will then be designed so that given a set of options (future states) it will choose that state with the most favorable patterns. Now note that the two crucial processes: a. learning weights (importance) of patterns and b. choosing favorable states can be done independently of the specific content of the patterns. All that matters is whether each pattern matches and its weight. of It is this domain-independent syntactic process that we believe is an integral component of creativity. For this process to succeed in any domain it must be coupled with a very powerful pattern representation language that represents only the relevant factors in a given problem or else the system will suffer from the combinatorial explosion that arises if too many potential patterns (features) need be considered. For now, we will place the burden of constructing this representation on humans (and allow it to be domain-specific), though in principle we believe that computers should be able to do this as well.

1.2 The Morph Adaptive Pattern-Oriented Chess System

To study computational intuition or creativity we have built "Morph" – an adaptive, pattern-oriented chess program. In contrast to traditional chess programs, Morph's moves are not based on search of the game tree, but rather on patterns that have been developed and learned from experience(using a combination of machine learning methods including genetic algorithms, weight-updating, simulated annealing, temporal-difference learning and explanation-based structure learning). Little effort has been applied to incorporating experience into a chess program:

> Although it is apparently effective to discover tactical issues by searches, isn't it dull to "forget" them immediately after use instead of "learning" something for a later re-use? There is no serious attempt known to make a chess program really learn from its experience for future games itself. Moreover, the rediscovery of the "same" motif over and over again in many subtrees is obviously the most effective means. *H. Kaindl [15]*

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! There do now exist some systems that recall positions that they found promising, but from which they later lost material [39,43]. This is certainly a step in the right direction, but much more important than dealing with exact replication of positions is to handle situations that are analogous, but not identical, to previous situations.

Morph's patterns come in two varieties: graph patterns represent attacks and defends relationships between pieces and squares on the chessboard and material patterns are vectors that give the relative material difference between the two players ("up a pawn", "even material","down a bishop, up a knight", etc. With each pattern is attached a weight or significance - a number in [0,1] that is an estimate of the expected true minmax evaluation of positions that contain the pattern as a subpattern. Morph uses a uniform combining rule to combine the significances of the most specific applicable patterns for a given position P to reach an heuristic evaluation of P. Thus the evaluation of any given position P is indirectly built out of experiences in similar positions, for it is new combinations of concepts: symmetry, parsimony, complementarity, etc. It is our thesis that the creative act can be implemented and viewed as a syntactic, rather than a semantic process and, furthermore, from the syntactic point of view creativity can be seen as a largely domain-independent process. As a domain independent process some of the "magic" returns, creativity not being a function of knowledge but a method of combining experience that is universal.

What is it that remains constant in our day-to-day moment-to-moment lives? Is it not experience in one form or another! Do not many of our decisions and actions arise from our past experiences in similar situations to the ones we currently find ourselves in? Isn't it odd then that until recently (say the last five years) experience has been largely ignored in Artificial Intelligence research? Most AI work has been on building static knowledge bases or domain theories from which inferences could be drawn and queries answered. In most heuristic search systems the system explores a state space on its own using a fixed heuristic, but little of this experience is compiled for later use. In machine learning systems, most effort has been on "supervised learning" where examples have been pre-classified by an expert or an oracle is available for such information. In other cases, the domain pursued has been so small that experience could be stored directly as full or partial state descriptions without facing memory limitations. Only recently has "reinforcement learning" [45,2,11] been pursued seriously. In this learning paradigm, little feedback is provided to the system except for occasional reward or punishment from the environment.

There seems to be one question in our lives that no one can answer for us: "am I happy?" In this area of inner subjective experience we are responsible for our own feedback and then, perhaps, consciously or unconsciously attempt to mold circumstances , attitudes and interpretations to re-create a feeling of happiness that we have experienced in the past. Perhaps, our past experiences are encoded with our relative experience of happiness at that time (say in the range [0,1] - though no doubt such a scale should be multi-dimensional) and the idea is to create experiences that are most similar to happy experiences in the past? We suggest that this same inner process by which humans attempt to achieve happiness may be used in other problem-solving tasks as well. In this case experiences are not encoded with respect to happiness but with respect to some domain-specific goal (such as reaching one's destination safely when driving a car) and patterns and associations (such as a red stop light, or snow on the pavement) are built up around this goal. Then, maybe, it should be no wonder that various emotional states such as tension and breakthrough are tied up with creative processes? Possibly, each pattern brings with it its own nuance of emotion and the painter is really mixing these nuances within and that what appears on the canvas is merely a manifestation of this inner process? Maybe, there is little difference between the scientist, the sculptor or the chessmaster, each manipulating their individual pattern-emotion associations, but perhaps with the same inner process:

> [An expert] instinctively knows that in a novel case, this and not that will be the promising course of action. The well-known story of the old judge advising the new one never to give reasons for his decisions, "the decisions will probably be right, the reasons will surely be wrong" illustrates this. The doctor will feel that the patient is doomed, the dentist will have a premonition that the tooth will break, though neither can articulate a reason for his foreboding. The reason lies embedded, but not yet laid bare, in all the countless previous cases dimly suggested by the actual one, all calling up the same conclusion, which the adept thus finds himself swept on to, he knows not how or why. *William James [14].*

Experience-Based Creativity

Robert Levinson
Department of Computer and Information Sciences
University of California Santa Cruz
Santa Cruz, CA 95064 U.S.A.
(408)459-2087
ARPANET:levinson@cis.ucsc.edu

Abstract

This paper argues for a computational model of creativity as the unique and favorable combining of past experiences. It is hypothesized that the inner process that combines the experiences is universal and largely syntactic, i.e. it is only the emotional associations that are attached to encoded experience and not the semantic content of experience that is used by the inner creative process. A chess system, "Morph", has been developed that can be viewed as exploring this model. Morph is limited to 1-ply of search, little domain knowledge and no supervised training. It is only through playing another program, and encoding its experience (as "pattern-weight" pairs) that it can improve. Morph's learning system is a combination of machine learning methods that have been successful in other settings - and is covered in depth. Part of Morph's strength comes from a powerful representation for chess patterns that allows generalizations across positions (experiences). Performance results, including displays of creativity by Morph, are also presented.

1 Introduction

Calculation is only one side of it. In chess no less important is intuition, inspiration, or, if you prefer, mood. I for example cannot always explain why in one position this move is good, and in another bad. In my games I have sometimes found a combination intuitively simply feeling that it must be there. Yet I was not able to translate my thought processes into normal human language. *Former World Champion Mikhal Tal.*

1.1 Creativity and Experience (some speculation)

Creativity is the discovery or structuring of information into a new form that is very useful and that was not at all obvious before. We shall not deal with that form of creativity that may be regarded as magic(i.e. the discovery of a new form from virtually nothing) but shall study that creativity that results from a unique structuring of experience.

How can creativity (a new expression) arise from experience (the old)? It seems that there must be a medium by which experiences *can be combined* to create the new form. But the combination rule should be based on simple principles - else the result has been influenced too much by the system builder - contradicting the term "creative". Humans also use simple principles in creating