From: AAAI-94 Proceedings. Copyright © 1994, AAAI (www.aaai.org). All rights reserved.

A Strategic Metagame Player for General Chess-Like Games

Barney Pell
RIACS, NASA Ames Research Center
AI Research Branch, Mail Stop: 269-2
Moffett Field, CA 94035–1000
pell@ptolemy.arc.nasa.gov

Abstract

This paper introduces METAGAMER, the first program designed within the paradigm of Meta-Game Playing (Metagame) (Pell 1992a). This program plays Metagame in the class of symmetric chess-like games (Pell 1992b), which includes chess, Chinese-chess, checkers, draughts, and Shogi. METAGAMER takes as input the *rules* of a specific game and analyses those rules to construct for that game an efficient representation and an evaluation function, for use by a generic search engine. The strategic analysis performed by METAGAMER relates a set of general knowledge sources to the details of the particular game. Among other properties, this analysis determines the relative value of the different pieces in a given game. Although METAGAMER does not learn from experience, the values resulting from its analysis are qualitatively similar to values used by experts on known games, and are sufficient to produce competitive performance the first time METAGAMER actually plays each new game. Besides being the first Metagame-playing program, this is the first program to have derived useful piece values directly from analysis of the rules of different games. This paper describes the knowledge implemented in METAGAMER, illustrates the piece values METAGAMER derives for chess and checkers, and discusses experiments with METAGAMER on both existing and newly generated games.

1 Introduction

Virtually all past research in computer game-playing has attempted to develop computer programs which could play existing games at a reasonable standard. While some researchers consider the development of a game-specific expert for some game to be a sufficient end in itself, many scientists in AI are motivated by a desire for generality. Their emphasis is not on achieving strong performance on a particular game, but rather on understanding the general ability to produce such strength on a wider variety of games (or problems in general). Hence additional evaluation criteria are typically placed on the playing programs beyond mere performance in competition: criteria intended to ensure that methods used to achieve strength on a specific game will transfer also to new games. Such criteria include the use of learning and planning, and the ability to play more than one game.

However, even this generality-oriented research is subject to a potential methodological bias. The human researchers know at the time of program-development which specific game or games the program will be tested on, and therefore it is possible that they import the results of their own understanding of the game directly into their program. In this case, it is difficult to determine whether the subsequent performance of the program is due to the general theory it implements, or merely to the insightful observations of its developer about the characteristics necessary for strong performance on this particular game. An instance of this problem is the *fixed representation trick* (Flann & Dietterich 1989), in which many developers of learning systems spend much of their time finding a representation of the game which will allow their systems to learn how to play it well.

This problem is seen more easily when computer game-playing with known games is viewed schematically, as in Figure 1. Here, the human researcher or programmer is aware of the rules and specific knowledge for the game to be programmed, as well as the resource bounds within which the program must play. Given this information, the human then constructs a playing program to play that game. The program then plays in competition, and is modified based on the outcome of this experience, either by the human, or perhaps by itself in the case of experience-based learning programs. In all cases, what is significant about this picture is that the human stands in the centre, and mediates the relation between the program and the game it plays.

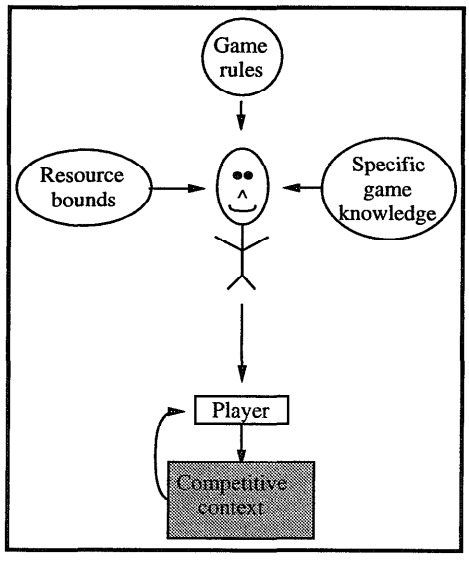

Figure 1: Computer Game-Playing with existing games: the human programmer mediates the relation between the program and the game it plays.

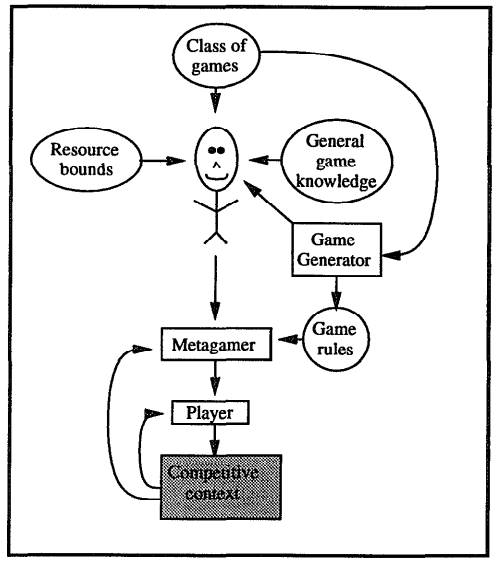

Figure 2: Metagame-playing with new games.

1.1 Metagame

Most AI games researchers, when pressed, will confess that their real interest is not in writing an expert-level chess program or checkers program. Their interest is in understanding general principles that are best tested by play at these well-understood games. This introduces the possibility of experimental bias. An experimenter can design a program that takes advantage of peculiar features of a single game, or uses a tuned representation to permit the program to learn features that the experimenter is well aware of.

The concept of *Meta-Game Playing* (Pell 1992a), shown schematically in Figure 2, is an attempt to reduce the possibility of such bias. Rather than designing a program to play an existing game known in advance, we design a *metagamer* to play a large but well-defined *class* of games. A metagamer takes as input only the rules of a new game (as produced by an automatic game generator and encoded in a well-specified language), and then produces a specialised program (a *player*) to play that game. Different metagamers are evaluated in the context of a *Metagame-tournament*, in which a set of new games are generated, the game rules are provided directly to the programs, and the programs then play the games against each other without human intervention. As only the *class* of games is known to the human developer in advance, metagamers are required to perform any game-specific optimisations without human assistance. The challenge is to produce the metagamer which receives the highest overall score across all game instances and opponents in the tournament. In contrast with the discussion on existing games above, the human no longer mediates the relation between the program and the games it plays, instead she mediates the relation between the program and the *class* of games it plays. By making the class explicit, we are able to quantify the level of generality achieved. Moreover, we can begin with classes which represent only moderate generalisations over tasks we have already looked at, and gradually move to more general classes of problems as scientific understanding develops.

1.2 SCL-Metagame

SCL-Metagame (Pell 1992b) is a Metagame research problem based around the class of symmetric chess-like games. The class includes the games of chess, checkers, noughts and crosses, Chinese-chess, and Shogi. An implemented game generator produces new games in this class (some of which are objects of interest in their own right (Pell 1992b)).

Symmetric Chess-Like Games A *symmetric chess-like game* is a two-player game of perfect information, in which the two players move pieces along specified directions, across rectangular boards. Different pieces have different powers of movement, capture, and promotion, and interact with other pieces based on ownership and piece type. Goals involve eliminating certain types

of pieces (*eradicate goals*), driving a player out of moves (*stalemate goals*), or getting certain pieces to occupy specific squares (*arrival goals*). Most importantly, the games are *symmetric* between the two players, in that all the rules can be presented from the perspective of one player only, and the differences in goals and movements are solely determined by the direction from which the different players view the board. For a formal definition and analysis of this class of games, see Pell (1992b).

Game Generation The goal of game generation is to produce a wide variety of games, all of which fall in the class of games as described by a *grammar*. Pell (1992b) implemented a statistical game generator for this class, and Pell (1993c) analysed the complexity and variety of games it produces.

Structure of Paper With the provision of the class definition and generator, Pell (1992b) formally defined the Metagame research problem of SCL-Metagame. This paper extends that work by constructing the first program to play games in this class, and by evaluating this program within the Metagame paradigm. The rest of this paper is organised as follows. Section 2 discusses the general architecture and the class-specific knowledge implemented in METAGAMER. Section 3 illustrates the analysis performed by METAGAMER to determine piece values for the games of chess and checkers, when presented with only the rules of those games. Section 4 discusses experiments with METAGAMER on those existing games and on newly generated games. Section 5 compares METAGAMER to other work on general game-playing programs and on automatic methods for determining feature values in games. Section 6 concludes the paper.

2 Constructing a Metagame-player

The main intention of Metagame is to serve as a test-bed for learning and planning. One obvious test of a testbed is to compute lower bounds: how well do existing techniques perform on the challenges of the testbed. In this section, we engineer a metagamer using existing game-playing techniques.

Search Engine To this end, the search engine used incorporates many standard game-playing search techniques (see Levy & Newborn (1991)). It is based on the *minimax* algorithm with *alpha-beta pruning*, *iterative deepening*, and the *principal continuation heuristic*. More details of the Metagame search engine are given by Pell (1993c).

Automated Efficiency Optimisation This powerful search engine should allow a playing program to search deeply. However, search speed is linked to the speed with which the primitive search operations of move-generation and goal detection can be performed. For a game-specific program, these issues can be easily hand-engineered, but for a program which is to play an entire class of games, the excess overhead of such generality initially caused the search to be unbearably slow. This problem was overcome by using a *knowledge-compilation* approach. We represent the semantics of the class of games in an extremely general but inefficient manner, and after receiving the rules of a given game the program automatically *partially evaluates* the general theory with respect to it, thus compiling away both unnecessary conditionality and the overhead of interpretation (Pell 1993a). In addition to the partial evaluation techniques discussed by Pell (1993a), METAGAMER contains a set of pre-computation methods which produce *static-analysis tables*. These tables speed up position evaluation by caching compute-intensive properties, in some cases providing approximate versions of knowledge which would be too expensive to compute correctly at runtime (Pell 1993c).

2.1 Meta-Level Evaluation Function

With the search engine in place, using the optimised primitive operations, we have a program which can search as deeply as resources permit, in any position in any game in this class. The remaining task is to develop an evaluation function which will be useful across many known and unknown games.

Following the approach used in HOYLE (Epstein 1989), we view each feature as an *advisor*, which encapsulates a piece of advice about why some aspect of a position may be favourable or unfavourable to one of the players. This section briefly explains some of the advisors currently implemented for METAGAMER. It should be noted that this list is incomplete due to space limitations, the set is still growing, and there are several important general heuristics which are not yet incorporated (such as *distance* and *control* (Snyder 1993)). Motivation and more detailed descriptions of all advisors were provided by Pell (1993c; 1993b). The advisors can be categorised into four groups, based on the general concepts from which they derive.

Mobility Advisors The first group is concerned with different indicators of *mobility*. These advisors were inspired in part by Church & Church (1979) and Botvinnik (1970), and by generalising features used for game-specific programs (Pell 1993c).

- `capturing-mobility`:counts the captures each piece could make in the current position.

• `dynamic-mobility`: counts the squares to which a piece can move directly from its current square on the current board, using a *moving* ability.[1]

• `static-mobility`: a static version of the above, this counts the squares to which a piece could move directly from its current square on an otherwise empty board.

• `eventual-mobility`: measures the total value of all squares to which a piece could move eventually from its current square on an otherwise empty board, using a *moving* ability. The value of each square decreases with the number of moves required for the piece to get there.[2]

Threats The second group of advisors deals with threats and conditions enabling threats. The advisors come in two types, *local* and *global*. Local advisors assess each threat separately, and return the sum of the values of all threats in a position. Global advisors determine the most important of these threats, and return only the maximum value of the local advisors. To determine the value of a threat, these advisors make use of the other advisors to determine, for example, the contribution a threatened piece makes to the present position.

Goals and Step Functions The third group of advisors is concerned with goals and regressed goals for this class of games.

• `vital`: Measures dynamic progress by both players on goals to eradicate sets of pieces.

• `arrival-distance`: this is a decreasing function of the abstract number of moves it would take a piece to *move* (i.e. without capturing) from its current square to a goal *destination* on an otherwise empty board, where this abstract number is based on the minimum distance to the destination plus the cost/difficulty of clearing the path.

• `promote-distance`: for each *target-piece* that a piece could promote into, this measures both the value and difficulty of achieving such a promotion.

• `possess`: This advisor handles games involving *placements*, in which a player can place a piece down on any of a set of squares. The value in such a situation is related to value the piece would have on the maximum available square.[3]

[1] Pieces in this class (e.g. checkers pieces) may move and capture in different ways (see Section 1.2).

[2] Thus while a bishop has 32 eventual moves and a knight has 64 from any square, the bishop can reach most of its squares more quickly, a fact captured by this advisor.

[3] Examples of such placement games are Shogi and Nine-Men's Morris.

Material Value The final group of advisors are used for assigning a fixed material value to each type of piece, which is later awarded to a player for each piece of that type he owns in a given position. This value is a weighted sum of the values returned by the advisors listed in this section, and does not depend on the position of the piece or of the other pieces on the board.

• `max-static-mob`: The maximum static-mobility for this piece over all board squares.

• `avg-static-mob`: The average static-mobility for this piece over all board squares.

• `max-eventual-mob`: The maximum eventual–mobility for this piece over all board squares.

• `avg-eventual-mob`: The average eventual-mobility for this piece over all board squares.

• `victims`: Awards 1 point for each type of piece this piece has the ability to capture (i.e. the number of pieces matching one of its *capture-types*).[4]

• `eradicate`: Awards 1 point for each opponent goal to eradicate this piece, and minus one point for each player goal to eradicate this piece.

• `stalemate`: This views the goal to stalemate a player as if it were a goal to eradicate all of the player's pieces, and performs the same computation as *eradicate* above.

• `promote`: This is computed in a separate pass after all the other material values. It awards a piece a fraction of the material value (computed so far) of each piece it can promote into. This advisor is not fully implemented yet, and was not used in the work discussed in this paper.

Section 3 provides concrete examples of the application of these advisors to the rules of different games discussed in this paper.

2.2 Weights for Advisors

The last major issue concerning the construction of the strategic evaluation function involves assigning weights to each advisor. While this issue is already difficult in the case of existing games, it is correspondingly more difficult when we move to unknown games, where we are not even assured of the presence of a strong opponent to learn from. However, the construction of some advisors provides one significant constraint on their possible values. For advisors which anticipate goal-achievement (such as `promote-distance` and the threat advisors), it would seem that their weight should not exceed 1. The reason is that they return some fraction of the

[4] A more sophisticated version of this feature, not fully implemented yet, takes into account the value of each victim, as determined by other static advisors.

value derived from achieving their anticipated goal. If such an advisor were weighted double, for example, the value of the threat would exceed the anticipated value of its execution, and the program would not in general choose to execute its threats.

Beyond the above constraint on such advisors, this issue of weight assignment for Metagame is an open problem. One idea for future research would be to apply temporal-difference learning and self-play (Tesauro 1994) to this problem. It would be interesting to investigate whether the "knowledge-free" approach which is so successful in learning backgammon transfers to these different games, or whether it depends for its success on properties specific to backgammon. In the meantime, we have been using METAGAMER with all weights set to 1.[5]

3 Examples of Material Analysis

One important aspect of METAGAMER's game analysis, which was discussed in Section 2.1, is concerned with determining relative values for each type of piece in a given game. This type of analysis is called *material analysis*, and the resulting values are called *material values* or *static piece values*. This section demonstrates METAGAMER's material analysis when applied to chess and checkers. In both cases, METAGAMER took as input only the rules of the games. In conducting this analysis, METAGAMER used the material advisors (Section 2.1) all with equal weight of one point each.

Checkers Table 1 lists material values determined by METAGAMER for the game of checkers, given only an encoding of the rules as an instance of the class of symmetric chess-like games (Pell 1992b).[6] In the table, K stands for *king*, and M stands for *man*. For compactness, advisors which do not apply to a game (and thus have value of 0 for all pieces) are not listed in material analysis tables for that game.

METAGAMER concludes that a king is worth almost two men. According to expert knowledge,[7] this is a gross underestimate of the value of a man. The reason that men are undervalued here is that METAGAMER does not yet consider the static value of a piece based on its possibility to promote into other pieces (see Section 2.1). When actually playing a game, METAGAMER does consider this, using the dynamic **promote-distance** advisor.

Material Analysis: checkers		
Advisor	Piece	
	K	M
max-static-mob	4	2
max-eventual-mob	6.94	3.72
avg-static-mob	3.06	1.53
avg-eventual-mob	5.19	2.64
eradicate	1	1
victims	2	2
stalemate	1	1
Total	23.2	13.9

Table 1: Material value analysis for checkers.

Chess Table 2 lists material values determined by METAGAMER for the game of chess, given only an encoding of the rules similar to that for checkers (Pell 1993c). In the table, the names of the pieces are just the first letters of the standard piece names, except that N refers to a knight.

Material Analysis: chess						
Advisor	Piece					
	B	K	N	P	Q	R
max-static	13	8	8	1	27	14
max-eventual	12	12.9	14.8	1.99	23.5	20.2
avg-static	8.75	6.56	5.25	0.875	22.8	14
avg-eventual	10.9	9.65	11.8	1.75	22.4	20.2
eradicate	0	1	0	0	0	0
victims	6	6	6	6	6	6
stalemate	1	1	1	1	1	1
Total	51.7	45.1	46.9	12.6	103	75.5

Table 2: Material value analysis for chess.

As discussed for checkers above, pawns are here undervalued because METAGAMER does not consider their potential to promote into queens, rooks, bishops, or knights. According to its present analysis, a pawn has increasingly less eventual-mobility as it gets closer to the promotion rank. Beyond this, the relative value of the pieces is surprisingly close to the values used in conventional chess programs (queen=9, rook=5, bishop=3.25, knight=3, and pawn=1) (Botvinnik 1970; Abramson 1990), given that the analysis was so simplistic.

[5] It should be noted that this is not the same as setting all piece values for a given game to equal value. The general knowledge still imposes constraints on the relative values of individual pieces in a game. For example, even a random setting of weights will cause METAGAMER to value queens above rooks in chess (Pell 1993c).

[6] The game definition for the entire rules of checkers as an instance of this class requires under 250 words and fits on one column of a page. It has been omitted due to space limitations.

[7] I am thankful to Nick Flann for serving as a checkers expert.

4 Summary of Results

An evaluation of the game-playing performance of METAGAMER was carried out by Pell (1993c) across a set of existing and generated games.

4.1 Known Games

Performance in the games of chess and checkers demonstrated that the knowledge encoded in METAGAMER endows it with a modest level of competence against highly specialised programs, which was still impressive given that METAGAMER plays these games in effect from first-principles.

Checkers The performance of METAGAMER in checkers was assessed by playing it against Chinook (Schaeffer *et al.* 1991). Chinook is the world's strongest computer checkers player, and the second strongest checkers player in general. As it is a highly optimised and specialised program, it is not surprising that METAGAMER always loses to it (at checkers, of course!). However, to get a baseline for METAGAMER's performance relative to other possible programs when playing against Chinook,[8] we have evaluated the programs when given various handicaps (number of men taken from Chinook at the start of the game).

The primary result from the checkers experiments was that METAGAMER is around even to Chinook, when given a handicap of one man. This is compared to a deep-searching greedy material program which requires a handicap of 4 men, and a random player, which requires a handicap of 8. In fact, in the 1-man handicap positions, METAGAMER generally achieves what is technically a winning position, but it is unable to win against Chinook's defensive strategy of hiding in the double-corner.

On observation of METAGAMER's play of checkers, it was interesting to see that METAGAMER "rediscovered" the checkers strategy of not moving its back men until late in the game. It turned out that this strategy emerged from the `promote-distance` advisor, operating defensively instead of in its "intended" offensive function. In effect, METAGAMER realized from more general principles that by moving its back men, it made the promotion square more accessible to the opponent, thus increasing the opponent's value, and decreasing its own.

Chess In chess, METAGAMER played against GnuChess, a very strong publicly available chess program.[9] GnuChess is vastly superior to METAGAMER (at chess, of course!), unless it is handicapped severely in time and moderately in material. The overall result of the experiments was that METAGAMER is around even to GnuChess on its easiest level,[10] when given a handicap of one knight. For comparison, a version of METAGAMER with only a standard hand-encoded material evaluation function (queen=9, rook=5, bishop=3.25, knight=3, and pawn=1) (Botvinnik 1970; Abramson 1990) played against METAGAMER with all its advisors and against the version of GnuChess used above. The result was that the material program lost every game at knight's handicap against GnuChess, and lost every game at even material against METAGAMER with all its advisors. This showed that METAGAMER's performance was not due to its search abilities, but rather to the knowledge in its evaluation function.

On observation of METAGAMER's play of chess, we have seen the program develop its pieces quickly, place them on active central squares, put pressure on enemy pieces, make favourable exchanges while avoiding bad ones, and restrict the freedom of its opponent. In all, it is clear that METAGAMER's knowledge gives it a reasonable *positional* sense and enables it to achieve longer-term strategic goals while searching only one or two-ply deep. This is actually quite impressive, given that none of the knowledge encoded in METAGAMER's advisors or static analyser makes reference to any properties specific to the game of chess—METAGAMER worked out its own set of material values for each of the pieces (see Section 3), and its own concept of the value of each piece on each square. On the other hand, the most obvious immediate limitation of METAGAMER revealed in these games is a weakness in *tactics* caused in part by an inability to search more deeply within the time constraints, in part by a lack of quiescence search, and also by the reliance on full-width tree-search. These are all important areas for future work.

4.2 New Games

Pell (1993c) carried out an experiment in the form of a *Metagame tournament.* In the experiment, several versions of METAGAMER with different settings of weights for their advisors played against each

[8] In our experiments, Chinook played on its easiest level. It also played without access to its opening book or endgame database, although it is unlikely that the experimental results would have been much altered had it been using them.

[9] GnuChess was the winner of the C Language division of the 1992 Uniform Platform Chess Competition.

[10] In the experiments, GnuChess played on level 1 with depth 1. This means it searches 1-ply in general but can still search deeply in quiescence search. METAGAMER played with one minute per move, and occasionally searched into the second-ply.

other and against baseline[11] players on a set of generated games which were *unknown* to the human designer in advance of the competition. The rules were provided directly to the programs, and they played the games without further human intervention. The most significant result of the experiment was that the version of METAGAMER which made use of the most knowledge clearly outperformed all opponents in terms of total score on the tournament. This was true despite the added evaluation cost incurred when using more knowledge. This result is evidence that the knowledge implemented in METAGAMER provides it with competitive strength across the entire class of games, at least relative to more limited versions of itself and to a set of baseline programs.

5 Related Work

This section compares METAGAMER to other work on general game-playing programs and on automatic methods for determining feature values in games. There have been many efforts to develop general game-playing programs which have been tested on more than one game (Epstein 1989; Williams 1972; Tadepalli 1989; Collins *et al.* 1991; Callan, Fawcett, & Rissland 1991; Gherrity 1993). However, none of these programs have been evaluated within the context of Metagame, which poses several unique challenges. First, the human designer cannot influence the representation of specific games (as they are input directed to the metagamers). Second, there is no existing body of knowledge or game records for newly generated games, which poses difficulties for approaches which rely on learning concepts or weights from textbooks or large amounts of existing games. Finally, there are no existing experts against which programs can train, which suggests that stronger programs will be based on self-play (Tesauro 1994) or more active forms of rule analysis (such as that performed by METAGAMER).

Despite its simplicity, METAGAMER's analysis produced useful piece values for a wide variety of games, which agree qualitatively with the assessment of experts on some of these games. This appears to be the first instance of a game-playing program automatically deriving material values based on active analysis when given only the rules of different games. It also appears to be the first instance of a program capable of deriving useful piece values for games unknown to the developer of the program (Pell 1993c). The remainder of this section compares METAGAMER to previous work with respect to determination of feature values.

Expected Outcome and Self-Play Abramson (1990) developed a technique for determining feature values based on predicting the *expected-outcome* of a position in which particular features (not only piece values) were present. The expected-outcome of a position is the fraction of games a player expects to win from a position if the rest of the game after that position were played randomly. He suggested that this method was an *indirect* means of measuring the mobility afforded by certain pieces. The method is statistical, computationally intensive, and requires playing out many thousands of games. Similar considerations apply to work on self-play (Tesauro 1994; Epstein 1992). On the other hand, the analysis performed by METAGAMER is a *direct* means of determining piece values, which follows from the application of general principles to the rules of a game. It took METAGAMER under one minute to derive piece values for each of the games discussed in this section, and it conducted the analysis without playing out even a single contest.

Automatic Feature Generation There has recently been much progress in developing programs which generate features automatically from the rules of games (de Grey 1985; Callan & Utgoff 1991; Fawcett & Utgoff 1992). When applied to chess such programs produce features which count the number of chess pieces of each type, and when applied to Othello they produce features which measure different aspects of positions which are correlated with mobility. The methods operate on any problems encoded in an extended logical representation, and are more general than the methods currently used by METAGAMER. However, these methods do not generate the *weights* of these features, and instead serve as input to systems which may learn their weights from experience or through observation of expert problem-solving. While METAGAMER's analysis is specialised to the class of symmetric chess-like games, and thus less general than these other methods, it produces piece values which are immediately useful, even for a program which does not perform any learning.

Evaluation Function Learning There has been much work on learning feature values by experience or by observation of strong players (e.g. (Samuels 1967; Lee & Mahajan 1988; Levinson & Snyder 1991; Callan, Fawcett, & Rissland 1991; Tunstall-Pedoe 1991)). These are all examples of passive analysis (Pell 1993c), and would not seem

[11] The baseline players consisted of a random player and a player that conducted a 2-ply search using a random evaluation function at non-terminal positions.

likely to produce a strong program in a Metagame-tournament until later rounds, after which the program would have had significant experience with stronger players.

6 Conclusion

The results on existing and generated games establish METAGAMER as a competent competitor for SCL-Metagame. At present, a number of challenger Metagame-playing programs are under development by independent researchers in the games and learning communities. This raises the possibility of a large-scale Metagame-tournament in the near future, which is an exciting prospect.

7 Acknowledgements

Thanks to Jonathan Schaeffer for permitting me to use Chinook, and to Stuart Cracraft for making GnuChess available. Thanks also to Othar Hansson, Robert Levinson, Pandu Nayak, Manny Rayner, and the anonymous reviewers for useful comments on drafts of this work. Parts of this work have been supported by the British Marshall Scholarship, the American Friends of Cambridge University, Trinity College (Cambridge), and the Cambridge University Computer Laboratory.

References

Abramson, B. 1990. Expected-outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12(2).

Botvinnik, M. M. 1970. *Computers, chess and long-range planning.* Springer-Verlag New York, Inc.

Callan, J. P., and Utgoff, P. 1991. Constructive induction on domain knowledge. In *Proceedings of AAAI-91.*

Callan, J. P.; Fawcett, T. E.; and Rissland, E. L. 1991. Adaptive case-based reasoning. In *Proceedings of IJCAI-91.*

Church, R. M., and Church, K. W. 1979. Plans, goals, and search strategies for the selection of a move in chess. In Frey, P. W., ed., *Chess Skill in Man and Machine.* Springer-Verlag.

Collins, G.; Birnbaum, L.; Krulwich, B.; and Freed, M. 1991. Plan debugging in an intentional system. In *Proceedings of IJCAI-91.*

de Grey, A. 1985. Towards a versatile self-learning board game program. Final Project, Tripos in Computer Science, University of Cambridge.

Epstein, S. 1989. The Intelligent Novice - Learning to Play Better. In Levy, D., and Beal, D., eds., *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad.* Ellis Horwood.

Epstein, S. 1992. Learning Expertise from the Opposition: The role of the trainer in a competitive environment. In *Proceedings of AI 92.*

Fawcett, T. E., and Utgoff, P. E. 1992. Automatic feature generation for problem solving systems. In Sleeman, D., and Edwards, P., eds., *Proceedings of the Ninth International Workshop on Machine Learning.*

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

Gherrity, M. 1993. *A Game-Learning Machine.* Ph.D. Dissertation, University of California, San Diego.

Lee, K.-F., and Mahajan, S. 1988. A pattern classification approach to evaluation function learning. *Artificial Intelligence* 36.

Levinson, R. A., and Snyder, R. 1991. Adaptive, pattern-oriented chess. In *Proceedings of AAAI-91.*

Levy, D., and Newborn, M. 1991. *How Computers Play Chess.* W.H. Freeman and Company.

Pell, B. 1992a. Metagame: A New Challenge for Games and Learning. In van den Herik, H., and Allis, L., eds., *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad.* Ellis Horwood.

Pell, B. 1992b. Metagame in Symmetric, Chess-Like Games. In van den Herik, H., and Allis, L., eds., *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad.* Ellis Horwood.

Pell, B. 1993a. Logic Programming for General Game Playing. In *Proceedings of the ML93 Workshop on Knowledge Compilation and Speedup Learning.*

Pell, B. 1993b. A Strategic Metagame Player for General Chess-Like Games. In *Proceedings of the AAAI Fall Symposium on Games: Planning and Learning.*

Pell, B. 1993c. *Strategy Generation and Evaluation for Meta-Game Playing.* Ph.D. Dissertation, Computer Laboratory, University of Cambridge.

Samuels, A. L. 1967. Some studies in machine learning using the game of Checkers. ii. *IBM Journal* 11.

Schaeffer, J.; Culberson, J.; Treloar, N.; Knight, B.; Lu, P.; and Szafron, D. 1991. Reviving the game of checkers. In Levy, D., and Beal, D., eds., *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad.* Ellis Horwood.

Snyder, R. 1993. Distance: Toward the unification of chess knowledge. Master's thesis, University of California, Santa Cruz.

Tadepalli, P. 1989. Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of IJCAI-89.*

Tesauro, G. 1994. TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation* 6(2).

Tunstall-Pedoe, W. 1991. Genetic Algorithms Optimizing Evaluation Functions. *ICCA-Journal* 14(3).

Williams, T. G. 1972. Some studies in game playing with a digital computer. In Siklossy, and Simon, H., eds., *Representation and Meaning.* Prentice-Hall.