A Strategic Metagame Player for General Chess-Like Games

Barney Pell*
Computer Laboratory
University of Cambridge
Cambridge, CB2 3QG, UK
bdp@cl.cam.ac.uk

Abstract

This paper reviews the concept of Metagame and discusses the implementation of METAGAMER, a program which plays Metagame in the class of symmetric chess-like games, which includes chess, Chinese-chess, checkers, draughts, and Shogi. The program 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, both for use with a generic search engine. The strategic analysis performed by the program 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 the program actually plays each game it is given. This appears to be the first program to have derived useful piece values directly from analysis of the rules of different games.

1 The Problem

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.

1.1 Bias when using Existing Games

However, even this generality-oriented research is subject to a potential methodological bias. As the human researchers know at the time of program-development which specific game or games the program will be tested on, it is possible that they import the results of their own understanding of the game directly into their program. In this case, it becomes 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 and 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 (or at least an acceptable encoding of the rules if the program is already fairly general, like HOYLE [Epstein, 1989b]). 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.

1.2 Metagame

The preceding discussion only serves to emphasize a point commonly acknowledged in AI: whenever we design a program to solve a set of problems, we build in our own *bias* about how the program should go about solving them. However, while we cannot eliminate bias

*Parts of this work have been supported by RIACS, NASA Ames Research Center [FIA], the American Friends of Cambridge University, Trinity College (Cambridge), and the Cambridge University Computer Laboratory.

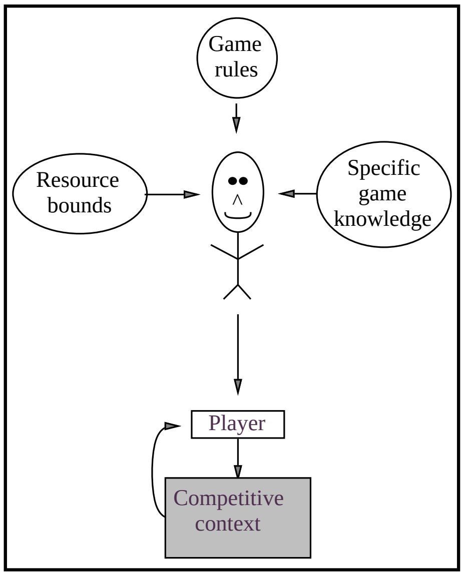

Figure 1: Computer Game-Playing with existing games: the human programmer mediates the relation between the program and the game it plays.

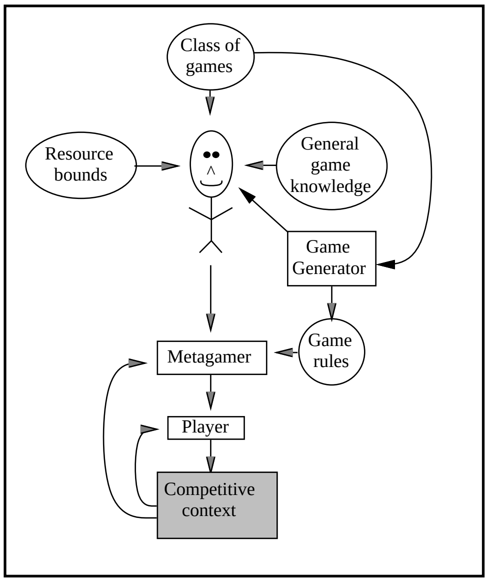

Figure 2: Metagame-playing with new games.

totally from AI research (or any other for that matter), we can to some degree *control for human bias*, and thus eliminate a degree of it.

This motivates the idea of *Meta-Game Playing* [Pell, 1992a], shown schematically in Figure 2. Rather than designing programs to play an existing game known in advance, we design programs to play (or themselves produce other programs to play) *new* games, from a well-defined class, taking as input only the rules of the games as output by an automatic game generator. As only the *class* of games is known in advance, a degree of human bias is eliminated, and playing programs are required to perform any game-specific optimisations without human assistance. 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.[1]

1.3 SCL-Metagame

SCL-Metagame [Pell, 1992b] is a concrete 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.

Symmetric Chess-Like Games Informally, 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.

As an illustration of how chess-like games are defined in this class, Figure 3 presents a grammatical representation of the complete rules for American Checkers as a *symmetric chess-like game*.

Game Generation The goal of game generation, as shown in Figure 4, is to produce a wide variety of games, all of which fall in the same class of games as described by a *grammar*. We also want to be able to modify the distribution of games generated by changing *parameters*, often in the form of *probabilities*. Finally, the combination of grammar and probabilities may not be enough to constrain the class according to our needs, in which case the generator should be able to handle a set of *constraints*.

The game generator for this class of games is illustrated schematically in Figure 4. The generator begins by generating a board and a set of piece names, and then generates goals and definitions for the pieces subject to the grammar and constraints, making choices using probabilities (parameters) attached to each choicepoint in the grammar. Details of the generator, its parameters, and example generated games are provided in [Pell, 1992b].

With the provision of the class definition and genera-

[1]Instead of playing a game, the program now plays a game about game-playing, hence the name *meta*-game playing.

```
GAME              american_checkers
GOALS             stalemate opponent
BOARD_SIZE        8 BY 8
BOARD_TYPE        planar
PROMOTE_RANK 8
SETUP  man AT {(1,1) (3,1) (5,1) (7,1) (2,2) (4,2)
               (6,2) (8,2) (1,3) (3,3) (5,3) (7,3)}
CONSTRAINTS       must_capture

DEFINE man
  MOVING
    MOVEMENT
      LEAP ⟨1,1⟩ SYMMETRY {side}
    END MOVEMENT
  END MOVING
  CAPTURING
    CAPTURE
      BY {hop}
      TYPE [{opponent} any_piece]
      EFFECT remove
      MOVEMENT
        HOP BEFORE [X = 0]
            OVER   [X = 1]
            AFTER  [X = 0]
        HOP_OVER [{opponent} any_piece]
        ⟨1,1⟩ SYMMETRY {side}
      END MOVEMENT
    END CAPTURE
  END CAPTURING
  PROMOTING
    PROMOTE_TO king
  END PROMOTING
  CONSTRAINTS continue_captures
END DEFINE

DEFINE king
  MOVING
    MOVEMENT
      LEAP ⟨1,1⟩ SYMMETRY {forward side}
    END MOVEMENT
  END MOVING
  CAPTURING
    CAPTURE
      BY {hop}
      TYPE [{opponent} any_piece]
      EFFECT remove
      MOVEMENT
        HOP BEFORE [X = 0]
            OVER   [X = 1]
            AFTER  [X = 0]
        HOP_OVER [{opponent} any_piece]
        ⟨1,1⟩ SYMMETRY {forward side}
      END MOVEMENT
    END CAPTURE
  END CAPTURING
  CONSTRAINTS continue_captures
END DEFINE

END GAME.
```

Figure 3: Definition of *American Checkers* as a symmetric chess-like game.

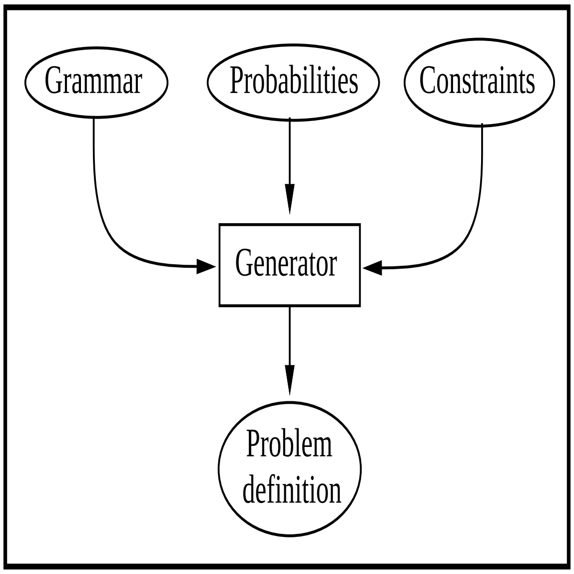

Figure 4: Components of a Game Generator.

tor (as well as various communications protocols to enable communication between programs, also discussed in [Pell, 1992b]), the problem of Metagame in symmetric chess-like games is thus fully defined. The rest of this paper is organised as follows. Section 2 discusses the challenge of representing knowledge for unknown games and the construction of METAGAMER. Section 3 discusses 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, and compares this to previous work on automatic methods for determining feature values in games. Finally, Section 4 presents the results of preliminary evaluations of METAGAMER on the games of chess and checkers.

2 Constructing a Metagame-player

While a main intention of Metagame is to serve as a test-bed for learning and planning systems, a logical first approach to constructing a Metagame-player is to push the limits of what can be done with standard game-tree search and knowledge-engineering. At the very least such an engineered player could serve as a baseline against which to measure the performance of more general systems.

2.1 Search Engine

To this end, the search engine used incorporates many standard search techniques from the game-playing literature (see [Levy and 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 in [Pell, 1993b].

2.2 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. This is discussed in detail in [Pell, 1993a].

2.3 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, 1989b], 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. But as the class of games to be played is different from that played by Epstein's HOYLE, we had to construct our advisors mostly from scratch.

In terms of the representation of the advisors, we follow an approach similar to that used in Zenith [Fawcett and Utgoff, 1992], in which each advisor is defined by a non-deterministic rule for assigning additional value to a position. The total contribution (value) of the advisor is the sum of the values for each solution of the rule. This method of representation is extremely general and flexible, and facilitates the entry and modification of knowledge sources. We have also tried when possible to derive advisors manually following the transformations used by Zenith [Fawcett and Utgoff, 1992] and CINDI [Callan and Utgoff, 1991], two systems designed for automatic feature generation.

2.4 Advisors

This section briefly explains the advisors currently implemented for METAGAMER. It should be noted that this set is not final, and there are several important general heuristics which are not yet incorporated (such as *distance* and *control* [Snyder, 1993]).

The advisors can be categorised into four groups, based on the general concept from which they derive.

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

- `dynamic-mobility`:counts the number of squares to which a piece can move directly from its current square on the current board, using a *moving* power.[2]
- `static-mobility`:a static version of immediate-mobility, this counts the number of squares to which a piece could move directly from its current square on an otherwise empty board, using a *moving* power.
- `capturing-mobility`:counts the number of captures each piece could make in the current position, regardless of the usefulness of the capture. It does not even distinguish whether the victim is a friendly or enemy piece. There is no static version of this advisor. For future work, one way to approximate the potential capturing ability of a piece might be to play random games and count the pieces attacked by each piece in each position.
- `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* power. The value of each square decreases (by a parameter-controlled function) with the number of moves required for the piece to get there. 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.

Threats and Capturing The second group of advisors deals with capturing interactions (in general, threats and conditions enabling threats):

- `local-threat`:for each target piece which a piece could capture in the current position, this measures the value of executing this threat (based on the other advisors), but reduces this value based on which player is to move. Thus a threat in a position where a player is to move is almost as valuable for that player as the value derived from executing it (i.e. he can capture if he likes), but if it is the opponent's move the threat is less valuable, as it is less likely to happen. Thus attacking an enemy piece while leaving one's own piece attacked (if the pieces are of equal value) is a losing proposition, but if the threatened enemy piece is worth much more this may be a good idea. The value of these threats are also augmented based on the *effect* of the capture.
- `potent-threat`:this extracts from the local-threat analysis just those threats which are obviously potent. A threat is *potent* for the player on move if the target is either undefended or more valuable (based on the other advisors) than the threatening piece. A threat is potent for the non-moving player only if the attacker is less valuable than the target and the moving-player does not already have a potent threat against the attacker.
- `global-threat`:The two threat advisors above exist in both *local* and *global* versions. The local version credits a player for each threat she has in a position, while the global version credits the player with only the *maximum* of those local threat values.
- `possess`:In a position where a player has a piece *in-*

[2]Recall from Section 1.3 that pieces in this class may move and capture in different ways.

hand, the player is awarded the dynamic value (using the local advisors) that the piece would receive, *averaged* over all empty board squares. Note that if the maximum value were used instead of the average, a program searching one-ply would never choose to place a piece on the board once possessed.[3]

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

• `vital:`this measures dynamic progress by both players on their goals to eradicate some set of piece types. As a given goal becomes closer to achievement, exponentially more points are awarded to that player. In addition, if the number of such remaining pieces is below some threshold, the remaining pieces are considered *vital*, in which case any potential threats against them automatically become potent.

• `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. This applies only to destinations for which the game has a defined arrival-goal for a piece-type consistent with the piece, and succeeds separately for each way in which this is true.

• `promote-distance:`for each *target-piece* that a piece could promote into (by the player's choice), this measures the value of achieving this promotion (using the other advisors), but reduces this value based on the difficulty of achieving this promotion, as discussed for arrival-distance above. The result is the maximum net value (to a player) of the discounted promotion options, or the minimum value if the opponent gets to promote the piece instead.[4]

• `init-promote:`This applies in a position where a player is about to promote the opponent's piece (on some square) before making a normal move. For each promoting-option defined for that piece, this calculates the value that option would have on the given square in the current position. The value returned is the maximum of the values of the options, from the perspective of the promoting player (and thus the minimum from the perspective of the player who owned the piece originally).

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. Note that *white king* and *black king* are viewed as different types of pieces during the computations below.[5]

• `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.

• `eradicate:`Awards 1 point for each opponent goal to eradicate pieces which match this type, and minus one point for each player goal to eradicate our own piece matching this type.

• `victims:`Awards 1 point for each type of piece this piece has a power to capture (i.e. the number of pieces matching one of its *capture-types*). A bonus is provided for the effects of each capture, as discussed for the local-threat advisor above. It would be interesting to have a dynamic version of this which gave preference to pieces which could capture other pieces actually present in a given position.[6]

• `immunity:`Awards 1 point for each type of enemy piece that *cannot* capture this piece.

• `giveaway:`Awards 1 point for each type of friendly piece that *can* capture 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.

• `arrive:`Awards a piece 1 point if the player has an arrival goal predicated on that type of piece. It awards $1/n$ points if the piece can promote to an arrival-goal piece in n promotions. Values are negated for opponent arrival goals.[7]

• `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 different games discussed in this thesis.

2.5 Static Analysis Tables

Most of these advisors draw on information which would be extremely slow to compute dynamically at

[3]This analysis would have to be improved substantially to capture the complexities of possessed pieces in games like Shogi.

[4]If we take the sum instead of the maximum here, a piece with many promotion options could be more valuable than its combined options, and thus it would never be desirable to promote it. For example, a pawn on the seventh rank would sit there forever enjoying all its options, but never caching them in.

[5]Thus if a piece gets one point for each piece it can possibly capture, and there are 5 distinct piece names, it is possible to score 10 points if the piece can capture all pieces.

[6]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.

[7]The net effect of this is that pieces which help a player to achieve arrival goals receive positive points, and those which help the opponent receive negative points.

each position to be evaluated. We overcome this problem in a similar manner as with the automated efficiency optimisation for the primitive search components, by having the program compile a set of tables after it receives the rules of each game, and then use those tables thereafter. Another advantage of this game-specialisation beyond efficiency is that after receiving the game rules, the program can take advantage of the rules to fold some of the goal tests directly into the tables.

As an example, one of the most frequently used tables is the `Constrained-Transition Table`. This table (used by all the static mobility advisors) records for each piece and square, the other squares the piece could move to directly on an otherwise empty board. However, it *excludes* all transitions from this set which can easily be shown either to be impossible or immediately losing for the player moving that piece. A transition is *impossible* if the starting square is one in which an arrival goal would already have been achieved when the piece first arrived at the square. A transition is *immediately losing* if the player would necessarily lose the game whenever his piece arrives on the destination square. While these considerations do not obviously apply to any known games, they prove highly relevant in generated games. Further details on static analysis tables are provided in [Pell, 1993b].

2.6 Weights for Advisors

The last major issue concerning the construction of the strategic evaluation function involves assigning weights to each advisor, or more generally, developing a function for *mediation among advisors* [Epstein, 1989a]. 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, by the construction of some of the advisors, we do have 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 always be at most 1. The reason is that the value they return is some fraction of the value which would be derived if the goal they anticipate were to be achieved. If such an advisor were weighted double, for example, the value of the threat would be greater than the anticipated value of its execution, and the program would not in general choose to execute its threats.

Beyond the 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, 1993] to this problem. It would be interesting to investigate whether the "knowledge-free" approach which was so successful in learning backgammon also 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.[8]

[8]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 [Pell, 1993b].

3 Examples of Material Analysis

One important aspect of METAGAMER's game analysis, which was discussed in Section 2.4, 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 illustrates the results of 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 material analysis, METAGAMER used the material advisors shown in Table 1, all with equal weight of one point each.

Advisor	Weight
dynamic-mobility	1
capture-mobility	1
global-threat	1
eventual-mobility	1
promote-distance	1
eradicate	1
vital	1
material	1
victims	1
max-static-mob	1
max-eventual-mob	1
eradicate	1
stalemate	1
arrive	1
giveaway	1
immunity	1

Table 1: Advisor weights for material analysis examples.

3.1 Checkers

Table 2 lists material values determined by METAGAMER for the game of checkers, given only the encoding of the rules as presented in Figure 3. In the table, K stands for *king*, and M stands for *man*.

METAGAMER concludes that a king is worth almost two men. According to expert knowledge,[9] 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 2.4). When actually playing a game, METAGAMER does take this into consideration, based on the dynamic **promote-distance** advisor.

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

Material Analysis: checkers		
	Piece	
Advisor	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
immunity	0	0
giveaway	0	0
stalemate	1	1
arrive	0	0
Total	23.2	13.9

Table 2: Material value analysis for checkers.

3.2 Chess

Table 3 lists material values determined by METAGAMER for the game of chess, given only an encoding of the rules similar to that for checkers [Pell, 1993b]. 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						
	Piece					
Advisor	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
immunity	0	0	0	0	0	0
giveaway	0	0	0	0	0	0
stalemate	1	1	1	1	1	1
arrive	0	0	0	0	0	0
Total	51.7	45.1	46.9	12.6	103	75.5

Table 3: 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, given that the analysis was so simplistic.

3.3 Discussion

Despite its simplicity, the 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 illustrates that the general knowledge encoded in METAGAMER's advisors and analysis methods is an appropriate generalisation of game-specific knowledge. 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, 1993b]. The following sections compare METAGAMER to previous work with respect to determination of feature values.

Expected Outcome Abramson [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. 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 and Utgoff, 1991; Fawcett and 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 *values* 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 observation (e.g. [Samuels, 1967; Lee and Mahajan, 1988; Levinson and Snyder, 1991; Callan *et al.*, 1991; Tunstall-Pedoe, 1991; van Tiggelen, 1991]). These are all examples of passive analysis [Pell, 1993b], and are not of use to a program until it has had significant experience with strong players.

Hoyle HOYLE [Epstein, 1989b] is a program, similar in spirit to METAGAMER, in which general knowledge is encapsulated using the metaphor of *advisors*. HOYLE has an advisor responsible for guiding the program into positions in which it has high mobility. However, HOYLE does not analyse the rules of the games it plays, and instead uses the naive notion of immediate-mobility as the number of moves available to a player in a particular position. The power of material values is that they abstract away from the mobility a piece has in a particular position, and characterise the potential options and goals which are furthered by the existence of each type of piece, whether or not these options are realised in a particular position. As HOYLE does not perform any analysis of the rules or construct analysis tables as does METAGAMER, it is unable to benefit from this important source of power.

4 Preliminary Results

The experimental evaluation of METAGAMER has only recently begun, but the preliminary results are exciting. To assess the strength of the program on known games, we are comparing it against specialised game-playing programs in the games of chess and checkers.

4.1 Checkers

We are comparing the performance of METAGAMER in checkers 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 (on checkers, of course!). However, to get a baseline for METAGAMER's performance relative to other possible programs when playing against Chinook,[10] we have evaluated the programs when given various handicaps (number of men taken from Chinook at the start of the game).

The preliminary results from the experiments are 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 "re-discovered" 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 defencively 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. Of course, as discussed in Section 1.1, we cannot make an unbiased claim about originality of the program, as this game was known to METAGAMER's designer beforehand, but this development can still be taken as an indication of the generality of the advisors across a variety of games.

[10]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.

4.2 Chess

In chess, we are playing METAGAMER against GnuChess, a very strong publicly available chess program, winner of the C Language division of the 1992 Uniform Platform Chess Competition. GnuChess is vastly superior to METAGAMER (at chess, of course!), unless it is handicapped severely in time and moderately in material.

The preliminary results from the experiments are that METAGAMER is around even to GnuChess on its easiest level,[11] when given a handicap of one knight. For purposes of 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.

4.3 New Games

The experiments above allow us to assess the application of the same general encoded knowledge to two

[11]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.

different games. However, the test of METAGAMER's ability on the task for which it was designed (i.e. SCL-Metagame) comes when we compare different programs on a tournament of newly-generated games. The results of such experiments are encouraging [Pell, 1993b], and will be reported in future work.

5 Acknowledgements

Thanks to Jonathan Schaeffer for permitting me to use Chinook and to Stuart Cracraft for contributing GnuChess to the research community. Thanks also to Siani Baker, Mark Humphrys, Rozella Oliver, Manny Rayner, and Eike Ritter for careful reading of drafts of this work.

References

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

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

[Callan and Utgoff, 1991] J. P. Callan and P.E. Utgoff. Constructive induction on domain knowledge. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 614–619, 1991.

[Callan *et al.*, 1991] James P. Callan, Tom E. Fawcett, and Edwina L. Rissland. Adaptive case-based reasoning. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991.

[Church and Church, 1979] Russell M. Church and Kenneth W. Church. Plans, goals, and search strategies for the selection of a move in chess. In Peter W. Frey, editor, *Chess Skill in Man and Machine.* Springer-Verlag, 1979.

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

[Epstein, 1989a] Susan Epstein. Mediation among Advisors. In *Proceedings on AI and Limited Rationality*, pages 35–39. AAAI Spring Symposium Series, 1989.

[Epstein, 1989b] Susan Epstein. The Intelligent Novice - Learning to Play Better. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad.* Ellis Horwood, 1989.

[Fawcett and Utgoff, 1992] Tom E. Fawcett and Paul E. Utgoff. Automatic feature generation for problem solving systems. In Derek Sleeman and Peter Edwards, editors, *Proceedings of the Ninth Interntational Workshop on Machine Learning*, Aberdeen, Scotland, 1992.

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

[Lee and Mahajan, 1988] Kai-Fu Lee and Sanjoy Mahajan. A pattern classification approach to evaluation function learning. *Artificial Intelligence*, 36:1–25, 1988.

[Levinson and Snyder, 1991] Robert A. Levinson and R. Snyder. Adaptive, pattern-oriented chess. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 1991.

[Levy and Newborn, 1991] David Levy and Monty Newborn. *How Computers Play Chess.* W.H. Freeman and Company, 1991.

[Pell, 1992a] Barney Pell. Metagame: A New Challenge for Games and Learning. In H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad.* Ellis Horwood, 1992. Also appears as University of Cambridge Computer Laboratory Technical Report No. 276.

[Pell, 1992b] Barney Pell. Metagame in Symmetric, Chess-Like Games. In H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad.* Ellis Horwood, 1992. Also appears as University of Cambridge Computer Laboratory Technical Report No. 277.

[Pell, 1993a] Barney Pell. Logic Programming for General Game Playing. In *Proceedings of the ML93 Workshop on Knowledge Compilation and Speedup Learning*, Amherst, Mass., June 1993. Machine Learning Conference. Also appears as University of Cambridge Computer Laboratory Technical Report No. 302.

[Pell, 1993b] Barney Pell. *Strategy Generation and Evaluation for Meta Game-Playing.* PhD thesis, Computer Laboratory, University of Cambridge, 1993. Forthcoming.

[Samuels, 1967] A. L. Samuels. Some studies in machine learning using the game of Checkers. ii. *IBM Journal*, 11:601–617, 1967.

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

[Snyder, 1993] Richard Snyder. *Distance*: Toward the unification of chess knowledge. Master's thesis, University of California, Santa Cruz, March 1993.

[Tesauro, 1993] G. Tesauro. TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, 1993. To Appear.

[Tunstall-Pedoe, 1991] William Tunstall-Pedoe. Genetic Algorithms Optimizing Evaluation Functions. *ICCA-Journal*, 14(3):119–128, September 1991.

[van Tiggelen, 1991] A. van Tiggelen. Neural Networks as a Guide to Optimization. *ICCA-Journal*, 14(3):115–118, September 1991.