Player Modeling, Search Algorithms and Strategies in Multi-player Games

Ulf Lorenz and Tobias Tscheuschner

Department of Computer Science,
Paderborn, Germany
{flulo, chessy}@upb.de

Abstract. For a long period of time, two person zero-sum games have been in the focus of researchers of various communities. The efforts were mainly driven by the fascination of special competitions such as Deep Blue vs. Kasparov, and of the beauty of parlor games such as Checkers, Backgammon, Othello, and Go.

Multi-player games, however, have been investigated considerably less, and although literature of game theory fills books about equilibrium strategies in such games, practical experiences are rare. Recently, Korf, Sturtevant and a few others started highly interesting research activities. We focused on investigating a four-person chess variant, in order to understand the peculiarities of multi-player games without chance components. In this contribution, we present player models and search algorithms that we tested in the four-player chess world. As a result, we may state that the more successful player models can benefit from more efficient algorithms and speed, because searching more deeply leads to better results. Moreover, we present a meta-strategy, which beats a paranoid α-β player, the best known player in multi-player games.

1 Introduction

We start with a short overview about two-person games (1.1), and about recent developments in the area of multi-player games (1.2).

1.1 Two-Person Games

In two-person zero-sum games, game-tree search is the core of most attempts to make computers play games. Typically, a game-playing program consists of three parts: a move generator, which computes all possible moves in a given position; an evaluation procedure which implements a human expert's knowledge about the value of a given position or an automatically generated heuristic evaluation function (in both cases, the values are quite heuristic, fuzzy, and limited), and a search algorithm which organizes a forecast.

For most of the interesting board games, we do not know the correct evaluations of all positions. Therefore, we are forced to base our decisions on heuristic or vague knowledge. At some level of branching, the complete game tree (as defined by the rules of the game) is therefore cut, the artificial leaves of the

H.J. van den Herik et al. (Eds.): ACG11, LNCS 4250, pp. 210–224, 2006.

resulting subtree are evaluated with the heuristic evaluations, and these values are propagated to the root [5,11,1] of the game tree as if they were real ones. For two-person zero-sum games, computing this heuristic minimax value is by far the most successful approach in computer-games history, and when Shannon [13] proposed a design for a chess program in 1949 — which is in its core still used by all modern game- playing programs — it seemed quite reasonable that deeper searches lead to better results. Indeed, the important observation over the last 40 years in the chess game and some other games is: *the game tree acts as an error filter*. Therefore, the faster and the more sophisticated the search algorithm, the better the search results! This is not self-evident, as some theoretical analyses show [2,10,4,6], but is the most crucial point in the success story of the forecast-based game- playing programs.

1.2 Multi-player Games

Most of the material concerning general strategic games comes from mathematical game theory. For an excellent introduction, we refer to [8]. In order to describe a strategic game, you need

1. the players (Who is involved?),
2. the rules (Who moves when? What do players know, when they move? What can they do?),
3. the outcomes (for each possible set of actions by the players, what is the outcome?), and
4. the payoffs (What are the players' utilities over the outcomes?).

In contrast to two-person zero-sum games, we cannot determine an optimal strategy in n-person zero-sum games independently from our opponent. In general, no so-called *dominating* strategy exists. Instead, we must be satisfied with some kind of equilibrium, mostly the so-called Nash- or User-equilibrium [9]: A set of n strategies $S_1, \ldots, S_n$ (one for each player) will be called in Nash-equilibrium, if none of the n players can improve his profit by changing his strategy S_i to S_i' under the assumption that all other $n-1$ players keep their strategies as they are. Such an equilibrium is called a *pure Nash equilibrium*. If we assume that the players do not select just one strategy, but that they select a random distribution over all their possible strategies, such a set of distributions will be called to be in equilibrium if none of the players can rise his expected profit by changing his probability distribution under the assumption that all the others keep their distributions. This kind of equilibrium is called a *mixed Nash-equilibrium*. For every finite game, at least one mixed equilibrium does exist.

The game that we inspect within this paper is an n-person game with complete information and alternating right to move. This means, at each point of time exactly one player has the opportunity to move, and all players can observe all other players' actions. Therefore, also a pure equilibrium does always exist [8]. For the special case that all outcomes at the leaves of the game tree are different from each other, the $\max^n$-algorithm finds such an equilibrium [3]. Otherwise, it

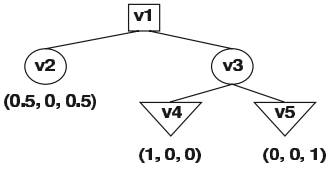

Fig. 1. Tree example with non-unique profit for player 1 at node v_1

cannot be predicted what is most rational for a player, respectively, the outcome is unstable. You can see this in Fig. 1.

Let us assume that we have three players, A,B, and C. A, to move at node v_1, can achieve 0.5, when he moves to the left to node v_2. He can also achieve either 1 or 0 when he goes to the right, but in the given example, B at node v_3 can decide whether it will be 0 or 1. It is not possible to assign unique numbers to positions in order to evaluate them. Instead, we need either sets of numbers in order to describe the value of a position, or we need a function that combines the numbers to one value.

Therefore, opponent models become an important factor. Sturtevant and Korf [16] examined the α-β algorithm in the paranoid model in that each player assumes that all the others play against him, as well as the $\max^n$-algorithm in the model in that each player believes that all players try to maximize their own profit.

2 Organization of the Paper

We are going to describe the application of our interest in Section 2. In Section 3, we present the four player types which we used for our experiments: (1) the paranoid player, (2) the $\max^n$ player, (3) the careful $\max^n$ player, (4) the coalition player, and in particular (5) the coalition-mixer player. We were mainly interested in the following questions.

- Which opponent model is strong in multi-player games without dice? Can we acknowledge Sturtevant's and Korf's observations?
- Most importantly: can the error-filter effect of game trees (as we described above) be observed in n-person games? We are convinced that only player models which can benefit from Moore's law have a chance to survive in the long run. Are there differences between the known player models concerning this issue?
- How can we use our knowledge about error filtering forecasts and our observations about the previously mentioned players, in order to construct stronger players? Are there efficient algorithms for the strong player models?
- One important step in the progress of two-person chess was the introduction of quiescence searches. Are quiescence searches useful in four-person chess?

In Section 4 we discuss the results of our experiments and we also report about some attempts which failed. Section 5 ends the paper with some conclusions.

2.1 The Four-Person Chess Game

A four-person chess game is the game of interest in this paper. It has the following four interesting properties.

(1) The fact that four players fight against each other instead only two, changes the character of the game completely, and offers a wide research field. Is α-β search still useful? Is forecasting still useful? What is a good playing strategy?
(2) The board and the pieces of the game are similar to the traditional chess ones. Therefore, we may assume that the heuristic board evaluation can be kept similar to the chess evaluation: piece values, mobility, king safety... We see good chances that we can carry over our expertise from two-person chess.
(3) For players already familiar with chess, it is not difficult to understand the extra rules for four-player chess.
(4) Four player chess has no dice and contains all the problems of market models usually inspected in game theory. Therefore, we are optimistic that results that we achieve in this little game will have impact on the wide field of economics.

Fig. 2. The four-person chess board

All participants — usually called White, Black, Blue, and Yellow, respectively South, West, North and East — play against all others, fighting for one full point. A draw between four players brings a quarter point, between three players a third

of a point etc. The chessboard is a Chessapeak Challenge(R)[1] design, but we tried to keep the rules as near to normal chess as possible.

As you see in Fig. 2, the board consists of an 8×8 chess board with 4 additional 3×8 partial boards connected to the four sides of the middle chess board. In the initial position, the Kings are always placed on the right of their Queens. All pieces move like in traditional Chess, only the Pawns have some extra opportunities. The squares on the main diagonals of the 8×8-mid-board are marked. There, Pawns can change their direction such that their distance from their original square to a promotion square remains 12 steps. Each Pawn can do that only once per game.

The game is artificially made finite with the help of the draw-by-repetition rule and the 50-moves rule, which are defined analogously to the classic chess rules. A move consists of 2, 3, or 4 partial moves, depending on how many players are on the board. If a player is mated or his king can be taken, his pieces are taken from the board.[2]

3 The Player Types

The evaluation procedure is for all player-types the same. It is measured by a relative game portion $S(i)$, which means that the piece values $M(i)$, the static and dynamic piece-square-values $SPT(i)$, and $DPT(i)$, plus the king safety $KS(i)$ of player i is divided through the sum of the values of all players:

$$S(i) = \frac{M(i) + SPT(i) + DPT(i) + KS(i)}{\sum_{j=1}^{n} M(j) + SPT(j) + DPT(j) + KS(j)}\,. \tag{1}$$

3.1 The Paranoid Player Type

Let $G = (V, E, H)$ be a game tree with the set of nodes V and edges E. The nodes shall correspond to game positions and the edges to moves from one position to the next. The α-β algorithm is a depth-first search algorithm, which runs into the search tree down to a predetermined level d. The leaves of the tree are evaluated by some (heuristic) evaluation function $H : V \to [0, 1]$, that assigns values (here for the sake of simplicity $[0, 1]$) to positions of the game. An appropriate game tree for the α-β algorithm consists of two disjoint subsets of nodes, so called min-nodes and max-nodes. At max-nodes, the max-player must move and he[3] builds values of inner nodes by computing the maximum over its successor values. The analog is valid for the min-player who minimizes over successor values. Values of inner nodes of the tree are called minimax values. The α-β standard algorithm can easily be extended to n-person games if the paranoid model is used.

[1] http://chessapeak.com/chess.html

[2] The detailed rules can be found at http://wwwcs.uni-paderborn.de/fachbereich/AG/monien/PERSONAL/FLULO/4PChess.html

[3] In this paper we use 'he' and 'his' whenever 'he or she' and 'his or her' are meant.

```
value alphabeta (Position v, value α, value β, remaining depth d, player i)
  // Let player 1 be the max-player, let the others be min-players.
  compute the feasible successor positions v_1, ..., v_b of v.
  // Let H(v) be the evaluation function for leaves.
  if (d == 0 or  b == 0) return H(v);
  for j := 1 to b
    if (max-player has to move) {
      α := maximum(α, alphabeta(v_j, α, β, d − 1, (i + 1) mod n));
      if α ≥ β return α;
      if j == b return α;
    } else  {
      β := minimum(alphabeta(v_j, α, β, d − 1, (i + 1) mod n), β);
      if α ≥ β return β;
      if j == b return β;
    }
```

At the best, the algorithm finds out the minimax value of the root position and needs to examine only $O(b^{t\cdot(n-1)/n})$ leaves [16], n being the number of players, b a uniform branching factor, and t the search depth.

Hash tables, history heuristic, killer moves, null moves, zero-window search, and iterative deepening are important additional techniques that may enhance an α-β algorithm [12], such that the best case can be nearly achieved in practice.

3.2 The max^n-Player Type [7]

Again, let (V, E, H) be a game tree, but V being partitioned into n disjoint subsets for n different players. Let H be the heuristic evaluation function. In contrast to what we saw before, let it return a vector of profits. $H : V \to [0,1]^n$, with $\sum_{i=1}^{b} H_i(v) = 1$. Let $F_i(v)$, the i^{th} component of a current max^n-vector of a node v, describe the profit of player i in position v.

```
profit-vector max^n (position v, lower bound on predecessor's profit
        m = F_{i−1}(predecessor(v)), remaining depth d, player i)
  compute the feasible successor positions v_1, ..., v_b of v.
  // Let H(v) be the evaluation function for leaves.
  if (d == 0 or  b == 0) return H(v);
  profit-vector a := (−1, ..., −1);
  for j := 1 to b
    F(v_j) := max^n (v_j, a_i, d − 1, (i + 1) mod n));
    if (F_i(v_j) > a_i) a := F(v_j) ;
    if a_i > 1 − m return a; // shallow pruning
    if j == b return a;
```

The algorithm performs a so-called shallow pruning, i.e., pruning which is caused by the predecessor of a cutoff node. Without speculation, this is the only possible pruning [15]. The following example shows that so-called deep pruning is not possible.

In Fig. 3, the third component of the profit vector of node (e) has the value 0.5. This is larger than $1 - (F_1(a)) = 1 - 0.6 = 0.4$, and therefore, player 1 will

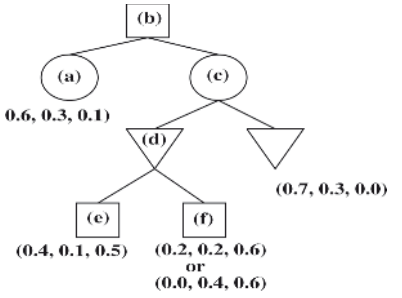

Fig. 3. Example for dangerous deep pruning

avoid that node (e) occurs on the board. Therefore, we might be willing to cutoff the node (f). This procedure is called 'deep pruning', because the grandfather node (b) of the node (d) causes the cutoff.

However, although the profit vector of the node (f) cannot reach the root (b) when the max^n-algorithm is used, (f) can change the root profit vector. First, let us assume that (f) has the vector $(0.2, 0.2, 0.6)$. In this case, player 3 will propagate the vector to node (d). As a consequence, player 2 will choose the vector $(0.7, 0.3, 0.0)$ for node (c). After all, player 1 assigns this vector $(0.7, 0.3, 0.0)$ to the root.

Now, let the value of node (f) be $(0.0, 0.4, 0.6)$. Player 3 will choose this for node (d) as well, but player 2 will propagate $(0.0, 0.4, 0.6)$ instead of $(0.7, 0.3, 0.0)$. Player 1 will see that node (a) is better for him than node (c), and the value vector of the root becomes $(0.6, 0.3, 0.1)$.

The paranoid player can also be interpreted as a special case of the $\max^n$-player with an appropriate evaluation function. In this sense, our distinction between paranoid and $\max^n$-player may look a bit artificial. We believe, however, that the strength of the 'special case' is severe enough that we feel encouraged to deal with it as a separate case.

3.3 The Careful $\max^n$ (cmax^n) Player Type

As we will see in the experimental section, the paranoid player does quite a good job. We tried to find out, what goes wrong with the $\max^n$ player, and we tried to find other updating rules for inner node values in order to beat the paranoid player.

The intuition behind the *careful* max^n *(*$cmax^n$*) player type* is the conjecture that the weakness of the $\max^n$ player comes from the fact that the players do not exactly know, what a real value vector of a position is, and that they do not know, how the other players estimate a position. Therefore, we assume that a player p, who has to move at node v, examines all successors, takes the observed profit vectors of the successors, and computes a weighted average over the successor profit vectors. We assumed it to be reasonable that a successor's weight becomes higher, the better it looks for player p. We assumed two advantages over the $\max^n$ updating rule. First, every position gets a unique value vector. Second, it

is modeled that all players want to maximize their own profit, but that they are not sure about the opponents incentives. One possibility to realize this concept is as follows.

Let v be a node and $v_1, \ldots, v_b$ its successors, n the number of players. Let $p_1, \ldots, p_b = (p_1^1, \ldots p_1^n), \ldots, (p_b^1, \ldots, p_b^n)$ be the profit vectors of the nodes $v_1, \ldots, v_b$. Let us furthermore assume that player P has to move. Then we define weights $w_1^P, \ldots, w_b^P$ for the b successors with

$$w_i^P := \frac{(p_i^P)^2}{\sum_{j=1}^{b} (p_j^P)^2}\,. \tag{2}$$

The new profit vector for node v is then $(p_1, \ldots, p_b)$ with $p_i := \sum_{k=1}^{b} w_i^P \cdot p_i^k$. As the experimental section will show, this player type is a misconception.

3.4 Coalition Players

The idea of the *coalition player* is as follows. During the game, either all players are more or less equally strong, or one gets the best position, one player the worst, and two are somewhere in between. The three non-leading players have a certain interest to stop the strongest player, but the weakest cannot sacrifice anything. He has the least interest to attack the leading player. The strongest player knows that the others want to bring him back, and must therefore play against all three. He should especially play against the weakest player, in order to mate him, before his strength is reduced to an average level. The strongest player plays against all others. The weakest player plays against the strongest in order to defend himself. The other two players can start coordinated attacks against the strongest player, they can even sacrifice some game portion. We created a player, who assumes this coalition scheme and who tries to benefit from the coalition by examining fewer moves than the normal paranoid player. The player with the lowest game portion ignored moves of the coaliting players and the coaliting players ignored the moves of the player with the lowest game portion. This attempt is too restrictive and neither successful.

3.5 Coalition-Mixer Player Type

The basic idea of the *comixer player* and the resulting comixer algorithm is to improve the behavior of the paranoid α-β player. The paranoid player is a good defender of a position, but shows some lacks in the offense.

In order to add the idea of cooperative attacks to the paranoid player, we provide the comixer player with minimax values of various coalitions. All the help searches for minimax values were made with the help of the α-β algorithm, of course. Although the many small help searches can only be performed with a relative small search depth, this player type beats an efficient paranoid player type.

The Algorithm. For a rough description of the algorithm, we present the following pseudo code.

(value of position v, move (v, w)) comixer (position v, player M, depth parameter d)
generate moves from v to the successors of v, named $v_1, \ldots, v_b$
// Let b be the number of moves.
// Let a paranoid value $p_i(v)$ be the minimax value that arises when player i
// plays against all the others, preserving the move rights.
1 compute the paranoid values $p_i(v)$ for all players, using α-β searches with depth d
2 Let T be the player with best $p_T(v)$ with $M \neq T$.
Select a set C of plausible coalitions. (Coalitions against T only.)
3 For all coalitions $c \in C$
4 For all moves (v, w_i) from v to successor $w_i, i \in \{1 \ldots b\}$
5 compute minimax value $e_c(v, w_i) :=$ alphabeta$(w_i, 0, 1,$ d)
// Needs modification of p.4 algorithm.
// This leads to an total search depth of $d + 1$.
6 For all moves (v, w_i) from v to successor $w_i, i \in \{1 \ldots b\}$
7 compute the benefit value of the move (v, w_i): comix(v, w_i).
//Comix is a piecewise linear function which combines the move values of the
//various coalitions to one move value.
8 return the best move with its value.

Explanation of the Algorithm. In the case of three remaining players, there are three possible coalitions, in which two players can cooperate with each other. In the case of four players we have four coalitions in which one single player fights against the remaining players and three two-by-two coalitions. For our four-person Chess game this is sufficient, but you can easily generalize the approach for a n-player game by calculating the corresponding possible coalitions.

In order to reduce the number of coalitions to be examined, the comixer algorithm does not compute the values of all possible coalitions. Instead, we concentrate on coalitions against the player with the best position. Let M be the player, who has the right to move. The comixer selects that player $T \neq M$ as a target, who possesses the greatest portion of the game (see line 2 of the algorithm) except M itself. This decision is made by performing the paranoid α-β procedure with a low searching depth (see line 1 of the algorithm). After this, the comixer calculates all coalitions with M and T not simultaneously being a member (see line 3 of the algorithm). For example, in the four-player game it calculates the paranoid coalition against himself (all against M), the coalition against player T (all against T) and the two remaining two-by-two coalitions, in which T is not in the team of M. Note, that these coalitions can also be computed by the α-β algorithm, because the game is reduced to a two-player game.

For getting minimax values for the moves of player M at the present node v, the α-β algorithm is not started at the root, but at its successors (see line 4-5 in the algorithm).

The Mixing Functions. The goal is to make the paranoid player cooperate with other players when it seems necessary. For instance, in the situation that one player gets a relatively high game value, he may be able to dominate all other players and win the game. To prevent this, the cooperating coalitions get higher weights, the more player T's game value reaches the 50% border.

When we reach line 6 of the comixer algorithm, we know how good the moves, starting from v, are in the specific coalitions. In order to make the benefits of the moves comparable to each other, we divide the value of a move by the value of the best move in the specific coalition. Note, that player M minimizes in the paranoid coalition against T. Thus, the best move has a value, that is not greater than the value of the considered move. So here you have to divide the value of the best move by the value of the given move. Altogether you get the following definition.

Let $c \in C$ be the observed coalition and $h \in \{1, \ldots, b\}$ be the index, for which $e_c(v, w_h) \geq e_c(v, w_j)\ \forall j \in \{1, \ldots, b\}$. Then the relative strength of a move (v, w_i) in a coalition c is defined as

$$r_c(v, w_i) := min\{\frac{e_c(v, w_i)}{e_c(v, w_h)}, \frac{e_c(v, w_h)}{e_c(v, w_i)}\}\,. \tag{3}$$

Now we can mix the values of the different coalitions (this is where the name of the algorithm comes from). The total value comix(v, w) of a given move (v, w) is calculated with the help of a set of k weight functions $f_1, \ldots, f_k$, which depend on the game portion of the player S with the highest game portion ($S = T$ or $S = M$), computed in line 1 of the comixer algorithm, whereas k is the number of the observed coalitions. A property of the f_i is, that

$$\sum_{c=1}^{k} f_c(x) = 1, \quad \forall x \in [\frac{1}{n}, 1]\,. \tag{4}$$

Then the comix function for a move with index i is defined as

$$comix(v, w_i) := \sum_{c=1}^{k} f_c(p_S(v)) \cdot r_c(v, w_i)\,. \tag{5}$$

We split the description of the functions in two important parts.

(1) M is determined to be the player with the highest value in line 1 of the algorithm. In this case, the mixing functions are very easy. The paranoid coalition gets a continously very high weight of between 80% and 90% depending on the number of the remaining players. The more players there are, the less is the importance of the paranoid coalition, because more coalitions participate. The residual 10-20% are shared by the other coalitions.

(2) The more interesting case will occur, when M is determined to be not the player with the highest value. Here we use piecewise linear functions to combine the weights of the coalitions. If there are three remaining players (see Fig. 4) we only need to combine two coalitions, as mentioned above.

When player T has a value of 33% of the game, the paranoid coalition against M has 90% weight and the paranoid coalition against T has 10% weight. This ratio switches linearly until player T has 50% of the game. From this rate on up to 100%, the weights are constant.

If there are four players in the game, the ratios between the paranoid coalitions are similar (see Fig. 5).

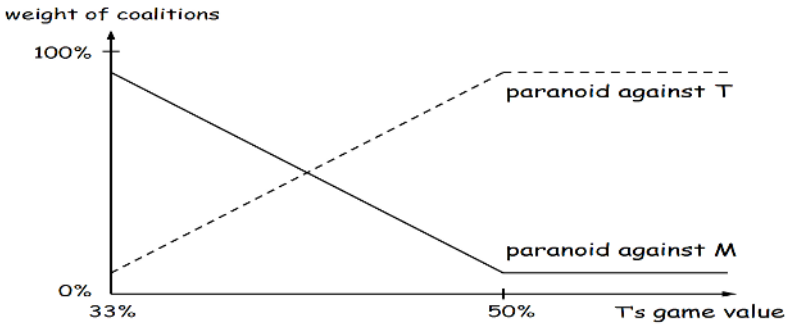

Fig. 4. Comix weight function for 3 players

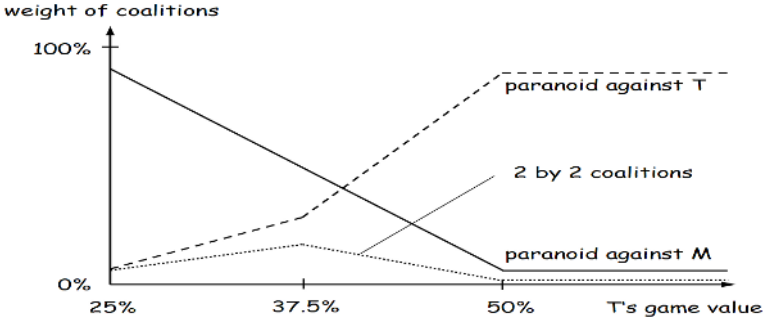

Fig. 5. Comix weight function for 4 players

Only the two two-by-two coalitions are added. Both begin with a value of 5%, when T's game value is 25% and reach their peak of 15% weight at 37.5% of T's game value. After that they fall down to 2% each at 50% of T's value. After this 50% line the functions again keep their values constant. Reasoned by the addition of the values of the two-by-two coalitions the values of the two paranoid functions (in particular the one against T) are decreased.

4 Experiments

Our generic four-person chess program consists of a move generator, a heuristic evaluation function which assigns a profit-vector to each position, and a search procedure. The evaluation function uses piece values for all pieces, similar as used in classic computer chess. A Knight, however, is not as much worth as a Bishop or three Pawns. Its value is only two Pawns. Moreover, we use mobility as a feature, and piece square tables in order to provide the players with a long-term idea where to place the pieces. The king cover consists of the existence/non-existence of the pawn shield, as well as of the distance between the pawn-shield and the King. The values are in relation to a player's portion of the game, i.e., the sum over all components of a profit vector is 1.

The search procedure depends on the player model used. In the case of the paranoid player, we use the α-β algorithm, enhanced with the killer heuristic, hash tables and iterative deepening. In all other cases, we use plain versions of the algorithms. Typical search data are the following. On a 2.4 GHz Pentium processor, the α-β algorithm searches about 1.1 million nodes per second without quiescence search and about 0.6 million nodes per second with quiescence search. Within 3 minutes computing time, it traverses search trees with depths between 7 and 8 without quiescence search, about 6 partial moves with quiescence search.

In order to play games, we set up 16 starting positions by hand. Six games were played with each of these 16 opening positions. Four players of two different types (i.e., two groups of two are identical players each) lead to six different orders, how the players can sit at the table. The two players of one group can sit at NE (North and East), SE, NW, SW, EW, or NS. The South-player starts the game. Thus, each contest consists of 96 games. In order to shorten the games, we adjudicate a game as soon as one player reaches more than 65% of the game portion. The semantic change of the players' actions can certainly be negotiated, and with this modification, a typical game takes between 200 and 300 quarter-moves. Mostly, at the end two players remain on board with one of them having more than 65% of the game portion.

4.1 Results

Table 1 presents some pairings with the paranoid player type and the max^n-player type involved, and the results. '+Qs' means quiescence search was used, '−Qs' means that it was not used. 'm/M' means minutes per move and tells us how many minutes time each side had for each move. We played 96 games per pairing, starting on a collection of 16 start positions. When a player type A scores more than 64 percent of the points against player type B, we may assume that this is not the effect of randomness.

Quiescence Search. Quiescence search is an important feature in traditional computer chess. The idea is that you should evaluate only so called quiet positions with the help of the static evaluation function. A position will be quiet if the player to move has no further taking moves. Quiescence search seems to be

Table 1. Results of pairings between the paranoid and the max^n-player type

Player 1	Player 2	Result
α-β +Qs , $3m/M$	α-β −Qs , $3m/M$	$46\frac{1}{6} : 46\frac{5}{6}$
max^n +Qs	max^n −Qs	$27\frac{5}{6} : 62\frac{1}{6}$
α-β +Qs , $3m/M$	max^n −Qs , $3m/M$	$86 : 10$
α-β +Qs, $3m/M$	max^n −Qs, $2m/M$	$84\frac{1}{2} : 11\frac{1}{2}$
α-β +Qs, $2m/M$	max^n −Qs, $3m/M$	$81\frac{5}{6} : 14\frac{1}{6}$
α-β +Qs, $3m/M$	α-β +Qs, $30s/M$	$64\frac{1}{2} : 31\frac{1}{2}$
max^n −Qs, $3m/M$	max^n −Qs, $30s/M$	$54 : 42$

less important in the 4-player chess game, because the α-β with quiescence search (+Qs) lost against the α-β without quiescence search (−Qs) with $46\frac{1}{6} : 46\frac{5}{6}$, and more distinct, the max^n +Qs lost against the max^n −Qs $27\frac{5}{6} : 62\frac{1}{6}$. For further experiments we use the quiescence search paranoid players, but not for the max^n players.

Paranoid α-β vs. max^n-Player. Although Sturtevant [14,15] already stated that the paranoid player was strong in several other games, the result of 86 : 10 is astonishingly clear. The following example is shown in order to show the different behavior of the different player types.

In the position of Fig. 6, only three quarters of a move are played, but Yellow (East) is in danger already. If East continues with a careless move (e.g. if he moves his queen-bishop pawn one step forward), White (South) can play Bg3 (South's bishop on the file of the South's queen) and Blue (North) can already mate East. Only the max^n player moves South's bishop to g3, because the paranoid player assumes that North will not take East out of the game as from the paranoia's point of view North and East are in a coalition against South. If a paranoid player leads East's pieces, he will interestingly handle East's position correctly. If he leads North's pieces, he will also correctly eliminate East. In simple words, the paranoid player participates in cooperative attacks only in the terminator role, but he never initiates such attacks as we will see below.

We repeated the games with a predetermined search depth instead of restricting the time per move, in order to find out whether structural reasons or the

Fig. 6. Early elimination of a careless player

superior efficiency of the α-β algorithm are responsible for the max^n disaster. The result was 88 : 8 for the paranoid player, and therefore, we may conclude that the real reason for the weaknesses of the max^n player lies in the fact that he relies on the other players' actions to be performed as he does expect them. Let us again inspect the position of Fig. 2. Assume in Fig. 2 the max^n player has South's pieces and is to move, he may initiate a cooperative attack by playing Qxl8+ (South's Queen takes East's Pawn). Assume now that (1) West helps South and plays Qxm9+ (West's Queen takes East's Pawn) and that (2) North would not help East by playing Bxl8 (North's Bishop takes South's Queen), but helps South and West, East is mated and may leave the board.

If West is a max^n player he will indeed help South to mate East. Although West's Queen will vanish (because North will not (neither if it is a max^n nor if it is a paranoid player) help East with Bxl8, and South will thereafter take West's Queen), West has a certain advantage, because East is completely out of the game. If, however, West is a paranoid player, he will never participate in this attack, because he believes North prevents the elimination of East after Qxm9+ by moving Bxl8, whereafter he would simply lose his Queen, because the only move East has, is taking West's Queen with his Knight Nxm9. But if West does not participate in the attack, South will just lose his Queen, because East will take it in the next move. In other words: it is the existence of the paranoid player which makes the game of the max^n player wrong.

Differences in Thinking Times. In the way as we expected it, the α-β player type with 3 minutes per move wins against the α-β player type with 30 seconds per move *clearly* with $64\frac{1}{2} : 31\frac{1}{2}$. Again, the max^n player faces problems. The player who has more time per move available wins only by 54 : 42.

The Careful max^n (cmax^n) Players and the Coalition Player. Both player types could not gain any benefit from their models. The careful max^n player lost $81\frac{5}{6}$ to $14\frac{1}{6}$, and the coalition player $87\frac{5}{6}$ to $8\frac{1}{6}$ against the paranoid players.

The Comixer Algorithm. The comixer played 96 games against the paranoid α-β as well. The depth parameter of the comixer was fixed at 4, and the time that the paranoid player had available depended on the time which the comixer needed for its calculation. The α-β player then received the arithmetic middle of the last two times the comixer needed. With the help of this rule, the paranoid players reached searching depths between 7 and 9. The result of the match was $58\frac{1}{3}$ to $37\frac{2}{3}$ for the comixer. The main disadvantage with this approach is that we do not yet know how its performance scales with increasing machine power.

5 Conclusion

We made experiments with various player models in the area of the four-person chess game. We created a couple of candidate player models, and examined their

relative strengths. Moreover, we investigated in how far computing power plays a role for the playing strength of the player models, and we investigated the effect of quiescence searches.

The paranoid player is quite strong and for a long time, we did not find any competitive other player model. After all, however, we were able to present the so-called comixer player type which performs better than the paranoid standard player. This proves that a simple worst-case analysis of the position is not the best choice, although being astonishingly effective. Moreover, some player models can better benefit from increasing computing power than others.

References

1. A. de Bruin, A. Plaat, J. Schaeffer, and W. Pijls. A Minimax Algorithm Better than SSS*. *Artificial Intelligence*, 87:255–293, 1999.
2. I. Althöfer. Root Evaluation Errors: How They Arise and Propagate. *ICCA Journal*, 11(3):55–63, 1988.
3. A.J. Jones. Game theory: Mathematical Models of Conflict. *West Sussex, England: Ellis Horwood*, 1980.
4. H. Kaindl and A. Scheucher. The Reason for the Benefits of Minmax Search. In *Proc. of the 11 th IJCAI*, pages 322–327, Detroit, MI, 1989.
5. D.E. Knuth and R.W. Moore. An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, 6(4):293–326, 1975.
6. U. Lorenz and B. Monien. The Secret of Selective Game Tree Search, When Using Random-Error Evaluations. *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS), (H. Alt, A. Ferreira), Springer LNCS*, pages 203–214, 2002.
7. C.A. Luckhardt and K.B. Irani. An Algorithmic Solution of n-Person Games. *Proceedings AAAI-86, Philadelphia, PA*, pages 158–162, 1986.
8. A. Mas-Colell, M.D. Whinston, and J.R. Green. Microeconomic Theory. *Oxford University Press*, 1995.
9. J.F. Nash. Non-Cooperative Games. *Annals of Mathematics, 54*, pages 286–295, 1951.
10. D.S. Nau. Pathology on Game Trees Revisited, and an Alternative to Minimaxing. *Artificial Intelligence*, 21(1-2):221–244, 1983.
11. A. Reinefeld. An Improvement of the Scout Tree Search Algorithm. *ICCA Journal*, 6(4):4–14, 1983.
12. J. Schaeffer. The History Heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(1):1203–1212, November 1989.
13. C.E. Shannon. Programming a Computer for Playing Chess. *Philosophical Magazine*, 41:256–275, 1950.
14. N.R. Sturtevant. A Comparison of Algorithms for Multi-Player Games. *Proceedings of the 3rd Computers and Games conference (CG2002)*, (eds. J. Schaeffer, M. Müller and Y. Björnsson), LNCS 2883, Springer-Verlag, Berlin, pages 108–122, 2003.
15. N.R. Sturtevant. *Multi-player games: Algorithms and Approaches.* PhD thesis, University of California, Los Angeles, 2003.
16. N.R. Sturtevant and R.E. Korf. On Pruning Techniques for Multi-Player Games. *Proceedings AAAI-00, Austin, Texas*, pages 201–207, 2000.