CHANCEPROBCUT: Forward Pruning in Chance Nodes

Maarten P. D. Schadd, Mark H. M. Winands and Jos W. H. M. Uiterwijk

Abstract— **This article describes a new, game-independent forward-pruning technique for EXPECTIMAX, called CHANCEPROBCUT. It is the first technique to forward prune in chance nodes. Based on the strong correlation between evaluations obtained from searches at different depths, the technique prunes chance events if the result of the chance node is likely to fall outside the search window. In this article, CHANCEPROBCUT is tested in two games, i.e., Stratego and Dice. Experiments reveal that the technique is able to reduce the search tree significantly without a loss of move quality. Moreover, in both games there is also an increase of playing performance.**

I. INTRODUCTION

Human players do not consider the complete game tree to find a good move. Using experience, they are able to prune unpromising variants in advance [1]. Their game trees are narrow and deep. By contrast, the original minimax algorithm searches the entire game tree up to a fixed depth. Even its efficient variant, the $\alpha\beta$ algorithm [2], is only allowed to prune if a position is known to be irrelevant to the principal variation (backward pruning). However, there are several forward-pruning techniques for the $\alpha\beta$ algorithm [3], [4], [5], [6]. These techniques have been only applied to deterministic games so far (e.g., Chess, Checkers, and Go).

Non-deterministic games introduce an element of chance by the roll of dice or the shuffle of cards (e.g., Backgammon and Ludo [7]). Also imperfect-information games, where not all information is available to a player, can be treated as non-deterministic games as if they contained an element of chance (e.g., Stratego[1]). For non-deterministic games, EXPECTIMAX [8] is the algorithm of choice. It extends the minimax concept to non-deterministic games, by adding chances nodes to the game tree. So far, no specific $\alpha\beta$ forward-pruning technique has been designed for these chance nodes.

This paper describes CHANCEPROBCUT, a forward-pruning technique inspired by PROBCUT [4] which is able to cut prematurely in chance nodes. This technique estimates values of chance events based on shallow searches. Based on the strong correlation between evaluations obtained from searches at different depths, the technique prunes chance events if the result of the chance node is likely to fall outside the search window.

This paper will first explain the EXPECTIMAX algorithm in Section II and explain possible STAR1 and STAR2 pruning in Section III. Thereafter, Section IV discusses Variable-Depth Search. We introduce the new forward-pruning technique CHANCEPROBCUT in Section V. Section VI describes the games Stratego and Dice. The experiments are presented in Section VII. Finally, Section VIII draws the conclusions of this research.

II. EXPECTIMAX

EXPECTIMAX [8] is a brute-force, depth-first game-tree search algorithm that generalizes the minimax concept to non-deterministic games, by adding *chances nodes* to the game tree (in addition to MIN and MAX nodes). At chance nodes, the heuristic value of the node (or EXPECTIMAX value) is equal to the weighted sum of the heuristic values of its successors. For a state s, its EXPECTIMAX value is calculated with the function:

$$\text{EXPECTIMAX}(s) = \sum_i P(c_i) \times V(c_i)$$

where c_i represents the ith child of s, $P(c)$ is the probability that state c will be reached, and $V(c)$ is the value of state c.

We explain EXPECTIMAX in the following example. Figure 1 depicts an EXPECTIMAX tree. In the figure squares represent chance nodes, regular triangles MAX nodes and inversed triangles MIN nodes.

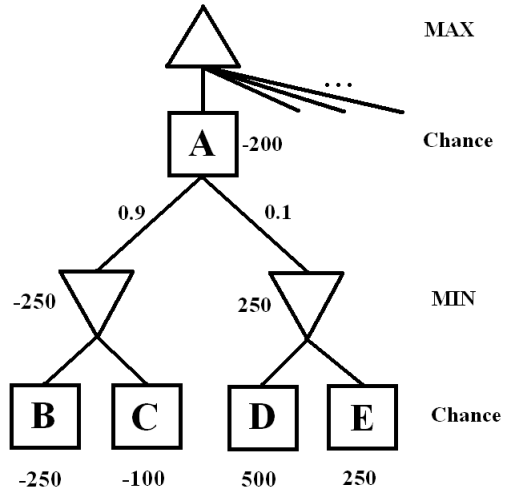

Fig. 1. An example EXPECTIMAX tree

In Figure 1, Node A corresponds to a chance node with two possible events, after which it is the MIN player's turn. The value of Node A is calculated by weighting the outcomes of both chance events. In this example, $Expectimax(A) = 0.9 \times -250 + 0.1 \times 250 = -200$.

Maarten Schadd, Mark Winands and Jos Uiterwijk are members of the Department of Knowledge Engineering, Faculty of Humanities and Sciences, Maastricht University, Maastricht, The Netherlands; {maarten.schadd, m.winands, uiterwijk}@maastrichtuniversity.nl

III. STAR1 AND STAR2

As stated by Hauk et al. [9], the basic idea of EXPECTIMAX is sound but slow [10]. STAR1 and STAR2 exploit a bounded heuristic evaluation function to generalize the $\alpha\beta$ pruning technique to chance nodes [9], [10]. $\alpha\beta$ pruning imposes a *search window* (α,β) at each MIN or MAX node in the game tree. Remaining successor nodes can be skipped as soon as the current node's value is proven to fall outside the search window. STAR1 and STAR2 apply this idea to chance nodes. The difference is that the search cannot return as soon as one successor falls outside the search window. The weighted sum of all successors has to fall outside the search window to be able to return.

STAR1 is able to create cutoffs if the lower and upper bound of the evaluation function are known (called L and U). These bounds are the game-theoretical value of terminal positions ($Loss$ and Win, respectively). If we have reached the ith successor of a chance node, after having searched the first $i-1$ successors and obtained their values, then we can obtain bounds for the value of the chance node. A lower bound is obtained by assuming that all remaining successors return L, an upper bound is obtained by assuming that all remaining successors return U. A safe pruning can be done if either one of these bounds falls outside the search window.

While STAR1 results in an algorithm which returns the same result as EXPECTIMAX, and uses fewer node expansions to obtain the same result, its results are generally not impressive. A reason for this is its pessimistic nature. In order to obtain more accurate bounds for a node, STAR2 probes each child. By only searching one of the available opponent moves, an upper bound for this node is obtained. This upper bound is then back propagated for calculating a more accurate upper bound for the chance node.

We explain STAR2 in the following example. Figure 2 depicts an EXPECTIMAX tree with two STAR2 prunings. Node A is reached with an $\alpha\beta$ window of (-150, 150). At this point, the theoretical lower and upper bounds of node A are [−1000, 1000], which correspond to losing and winning the game. STAR2 continues with probing the first possible chance event (i.e., investigating node B only). The result of this probe produces an upper bound for this chance event (≤ -250). The theoretical upper bound for A is now updated to following the EXPECTIMAX procedure: $-250 \times 0.9 + 1000 \times 0.1 = -125$. A cut is not possible yet and the same procedure is applied to the second chance event. After the second probe, the theoretical window of A is [−1000, −175] which is outside the $\alpha\beta$ window. Now nodes C and E can be pruned. Extra search effort would be caused if no pruning would occur. This effect can be nullified by using a transposition table.

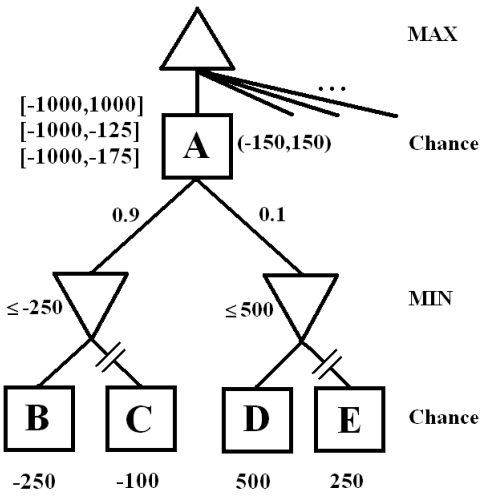

Fig. 2. Successful STAR2 pruning

IV. VARIABLE DEPTH SEARCH

Human players are able to find good moves without searching the complete game tree. Using their experience they are able to prune unpromising variations in advance [1]. Human players also select promising variants and search these deeper. In $\alpha\beta$ search this concept is known as *Variable Depth Search* [11]. The technique to abandon some branches prematurely is called *forward pruning*. The technique to search branches beyond the nominal depth is called *search extensions*. As such, the search can return a different value than that of a fixed-depth search.

In the case of forward pruning, the full critical tree is not expanded, and good moves may be overlooked. However, the rationality is that although the search occasionally goes wrong, the time saved by pruning non-promising lines of play is generally better used to search other lines deeper, i.e., the search effort is concentrated where it is more likely to benefit the quality of the search result.

The real task when doing forward pruning is to identify move sequences that are worth considering more closely, and others that can be pruned with minimal risk of overlooking a good continuation. Ideally, a forward pruning should have low risk, limited overhead, be applicable often, and domain independent. Usually, improving one factor will worsen the others [12].

The *null-move* heuristic [3], [13] is a famous way of forward pruning. Forfeiting the right to move is called a null move. In general, a move can be found with a higher value than the null move. This is not true for *zugzwang* positions, where doing nothing is the best option. The idea of the null-move heuristic is to search the null move and use the resulting value as a lower bound for the node. If this value exceeds β, a pruning takes place.

The *Multi-Cut* heuristic prunes an unpromising line of play if several shallow searches would produce a cutoff [12]. This technique was improved for ALL nodes later [14]. The $ProbCut$ [4] heuristic uses shallow searches to predict the result of deep searches. A branch is pruned if the shallow search produces a cutoff, with a certain confidence bound. This heuristic works well for techniques where the score of a position does not significantly change when searched deeper, as in Othello. The technique was further enhanced as the *Multi-ProbCut* procedure [5].

All these heuristics are applicable at MIN and MAX nodes. However, not much work has been done on forward pruning in trees with chance nodes. Smith and Nau [15] perform an analysis of forward-pruning heuristics on binary trees with chance nodes. No forward-pruning techniques in chance nodes have been proposed so far.

V. ChanceProbCut

So far, it has not been investigated if forward pruning can be beneficial in chance nodes. The null move and Multi-Cut heuristic cannot be adapted to chance nodes because these techniques are based on applying moves. We introduce ChanceProbCut to forward prune unpromising chance nodes. This technique is inspired by ProbCut [4], and uses this idea to generate cutoffs in chance nodes. In this case, a shallow search of depth $d - R$ is an indicator of the true expectimax value v_d for depth d. Now, it is possible to determine whether v_d would produce a cutoff with a prescribed likelihood. If so, the search is terminated and the appropriate window bound is returned. If not, a normal search has to be performed.

ChanceProbCut adapts the ProbCut idea to alter the lower and upper bounds for each possible event in chance nodes. A search with reduced depth is performed for each chance event. The result of this search, v_{d-R}, is used for predicting the value of the chance event for depth d (v_d). The prediction of v_d is calculated by confidence bounds in a linear regression model [4]. The bounds can be estimated by:

$$ELowerBound(v_d) = -\text{Percentile} \times \sigma + v_{d-R} \times a - b$$

$$EUpperBound(v_d) = \text{Percentile} \times \sigma + v_{d-R} \times a - b$$

where a, b and σ are computed by linear regression, and Percentile determines the size of the bounds. Using these bounds, it is possible to create a cutoff if the bounds multiplied with the corresponding chances fall outside the search window.

In a chance node, variables containing the lower and upper bounds are updated during each step. A cutoff may be obtained with values computed from different techniques. It is possible that during the normal search a cut is produced using a combination of Transposition Table Probing, ChanceProbCut, Star2 and exact values, because each time the bound is updated after new information becomes available.

When the normal search with depth d is started, the bounds obtained by ChanceProbCut can be used as search window. It is unlikely that v_d falls outside this interval. We note that it is possible to do a re-search after the normal search returns with a value outside the search window. However, we did not implement this because an error only partially contributes to the value of the chance node.

An example of how ChanceProbCut prunes is given in Figure 3. Here, the regression model $v_d = v_{d-R}$ is assumed, with confidence interval 50. The first ChanceProbCut search returns the bounds [300, 400] for the first chance event, changing the bounds of the chance node to [−480, 760]. After the second ChanceProbCut search returns the window [700, 800], the bounds of the chance node, [200, 680], fall outside the search window and a pruning takes place.

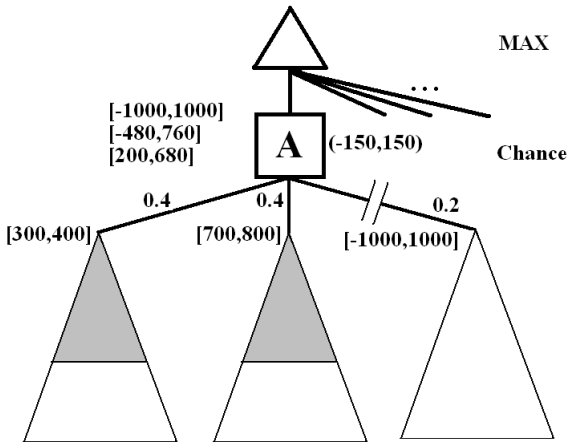

Fig. 3. ChanceProbCut prunes

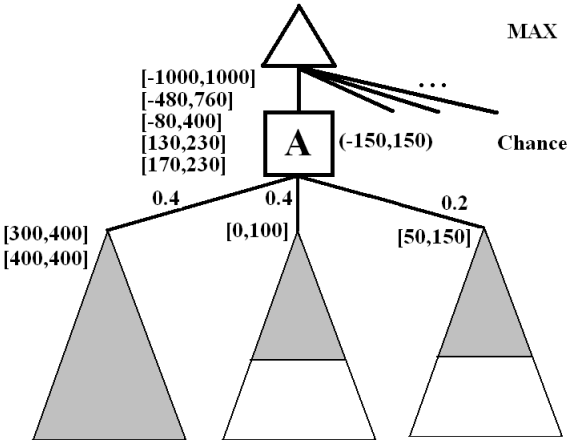

Fig. 4. The normal search prunes with help of ChanceProbCut

Figure 4 depicts a second example, in which ChanceProbCut fails to produce a pruning. At first, the search finds the same values as in the previous example. The next search reveals [0, 100] as bounds for the second event. This time, it is not possible to prune. Even after the next search produces the bounds [50, 150] for the third event, the bounds for the chance node [130, 230] still fall inside the search window. However, after the normal search returns value 400 for the first event, the search is terminated based on previously estimated ChanceProbCut values.

Finally, Algorithm 1 shows pseudo code for ChanceProbCut. The procedure $checkPruning()$ computes whether the lower and upper bounds of the chance nodes exceed the search window and returns if applicable. The $computeLowerBound()$ and $computeUpperBound()$ functions compute the confidence interval around the obtained value, based on a linear regression model [4]. The

Algorithm 1 CHANCEPROBCUT Forward Pruning

```
ChanceNode(alpha, beta, depth)

for all ChanceEvent i do
  lowerBound[i] = ELowerBound[i] = L;
  upperBound[i] = EUpperBound[i] = U;
end for

//ChanceProbCut
if depth>R then
  for all ChanceEvent i do
    doMove(i);
    v = search(lowerBound[i], upperBound[i],
        depth-1-R)
    ELowerBound[i] =
        max(lowerBound[i], computeLowerBound(v));
    EUpperBound[i] =
        min(upperBound[i], computeUpperbound(v));
    undoMove(i);
    if sum_j P_j× ELowerbound[j]>beta then return beta;
    if sum_j P_j× EUpperbound[j]<alpha then return alpha;
  end for
end if

//Star2
for all ChanceEvent i do
  doMove(i);
  v = probe(lowerBound[i], upperBound[i], depth-1);
  upperBound[i] = max(upperBound[i], v);
  EUpperBound[i] =
      min(upperBound[i], EUpperBound[i]);
  if upperBound[i] < ELowerBound[i] then
      ELowerBound[i] = L;
  undoMove(i);
  if sum_j P_j× upperbound[j]<alpha then return alpha;
end for

//Normal Search
for all ChanceEvent i do
  doMove(i);
  v = search(ELowerBound[i], EUpperBound[i],
      depth-1);
  lowerBound[i] = v;
  upperBound[i] = v;
  undoMove(i);
  if sum_j P_j× lowerbound[j]>beta then return beta;
  if sum_j P_j× upperbound[j]<alpha then return alpha;
end for

return sum_i P_i× lowerbound[i]
```

probe method is used for obtaining an upper bound of the chance node [9], [10] and the *search* method investigates the position further using the EXPECTIMAX framework.

VI. TEST DOMAIN

To test whether CHANCEPROBCUT performs well, we use two different games, Stratego and Dice.

A. Stratego

Stratego is an imperfect-information game. It was developed at least as early as 1942 by Mogendorff. The game was sold by the Dutch publisher *Smeets and Schippers* between 1946 and 1951 [16]. In this subsection, we will first briefly describe the rules of the game and thereafter present related work.

1) Rules: The following rules are an edited version of the Stratego rules published by the Milton Bradley Company in 1986 [17].

Stratego is played on a 10×10 board. The players, White and Black, place each of their 40 pieces in such a way that the back of the piece faces the opponent in a 4×10 area. The movable pieces are divided in ranks (from the lowest to the highest): Spy, Scout, Miner, Sergeant, Lieutenant, Captain, Colonel, Major, General, and Marshal. Each player has also two types of unmovable pieces, the Flag and the Bomb. An example initial position is depicted in Figure 5. The indices represent the ranks, where the highest rank has index 1 (the Marshal), and all decreasing ranks have increasing indices (Exceptions are S=Spy, B=Bomb, F=Flag).

4	8	B	F	B	8	6	9	8	3
6	7	2	B	7	9	S	1	B	7
3	B	5	4	6	8	B	8	9	9
5	9	6	9	9	5	9	4	7	5
		X	X			X	X		
		X	X			X	X		
9	6	7	3	9	6	4	5	6	9
5	9	8	9	6	9	5	2	B	9
B	7	S	1	8	B	3	7	4	5
F	B	8	4	B	7	B	8	9	8

Fig. 5. A possible initial position in Stratego

Players move alternately, starting with White. Passing is not allowed. Pieces are moved to orthogonally-adjacent vacant squares. The Scout is an exception to this rule, and may be moved like a rook in chess. The *Two-Squares Rule* and the *More-Squares Rule* prohibit moves which result in

repetition.[2] The lakes in the center of the board contain no squares; therefore a piece can neither move into nor cross the lakes. Only one piece may occupy a square.

A piece, other than a Bomb or a Flag, may attempt to capture an orthogonally adjacent opponent's piece; a Scout may attempt to capture from any distance. When attempting a capture, the ranks are revealed and the weaker piece is removed from the board. The stronger piece will be positioned on the square of the defending piece. If both pieces are of equal rank, both are removed. The Flag is the weakest piece, and can be captured by any moving piece. The following special rules apply to capturing. The Spy defeats the Marshal if it attacks the Marshal. Each piece, except the Miner, will be captured when attempting to capture the bomb.

The game ends when the Flag of one of the players is captured. The player whose Flag is captured loses the game. A player also loses the game if there is no possibility to move. The game is drawn if both players cannot move.

2) Previous Work: Stratego has not received much scientific attention in the past. De Boer [18], [19] describes the development of an evaluation function using an extensive amount of domain knowledge in a 1-ply search. Treijtel [20] created a player based on multi-agent negotiations. Stengård [21] investigates different search techniques for this game. At this moment, computers play Stratego at an amateur level [22]. An annual Stratego Computer Tournament[3] is held on Metaforge with an average of six entrants.[4]

B. Dice

The game of Dice is a two-player non-deterministic game, recently developed by Hauk [23], in which players take turns placing checkers on an $m \times m$ grid. One player plays columns, the other plays rows. Before each move, a dice is rolled to determine the row or column into which the checker must be placed. The winner is the first player to achieve a line of m checkers (orthogonally or diagonally).

The advantages of this game are that (1) it is straightforward to implement and (2) that many chance nodes exist. A disadvantage is that the outcome of the game is partially dependent on luck. A deep search is still beneficial. Hauk showed that a 9-ply player wins 65% against a 1-ply player [23].

Hauk uses this game to demonstrate the pruning effectiveness of STAR-minimax algorithms in a non-deterministic game [23]. Moreover, Veness used it to test StarETC [24].

C. Engine

We implemented an EXPECTIMAX engine for Stratego and Dice, enhanced with the STAR1 and STAR2 pruning algorithms [10], [23]. Furthermore, the History Heuristic [25], Killer Moves [26], Transposition Tables [27], [28] and StarETC [24] are used.

In Stratego, the evaluation function is material based. Furthermore, it awards a bonus for unknown pieces. This creates a game play where the player has to hide his important pieces as long as possible, but tries to reveal the location of the opponent's important pieces as soon as possible. A player receives a bonus for moving pieces to the side of the opponent, to promote the progression of the game. The evaluation function is bound to the interval $[-1000, 1000]$.

In Dice, the evaluation function counts the number of checkers which can be used for forming lines of size m. Checkers, which are fully blocked by the opponent, are not counted. Partially blocked checkers get a lower value. The evaluation function is bound to the interval $[-10, 10]$.

In both evaluation functions, a small random value is included for modeling the mobility [29].

VII. RESULTS

In this section, we first discuss the results of CHANCEPROBCUT in the game of Stratego. Second, we test CHANCEPROBCUT in the game of Dice.

A. Stratego

This subsection presents all the results obtained in the domain of Stratego.

1) Determining Parameters: The first parameters to choose are the depth reduction R and the depths d in which CHANCEPROBCUT is applied. The game tree in Stratego is not regular, meaning that not always a chance node follows a MIN/MAX node. Due to this we will not count chance nodes as a ply for Stratego. While in theory this technique can be applied on each search depth, we limit the applicability to $d \in \{4, 5\}$. R is set to 2, because otherwise an odd-even effect might occur. In order to find the parameters σ, a, and b for the linear regression model, 300 value pairs (v_{d-R}, v_d) have been determined. These value pairs are obtained from 300 begin, middle and endgame positions, created using selfplay.[5] In Figure 6 the model is shown for depths 2 and 4. Figure 7 shows the linear regression model for depths 3 and 5. v_{d-R} is denoted on the x-axis; v_d is denoted on the y-axis. Both figures show that the linear regression model is able to estimate the value of v_d with a small variance.

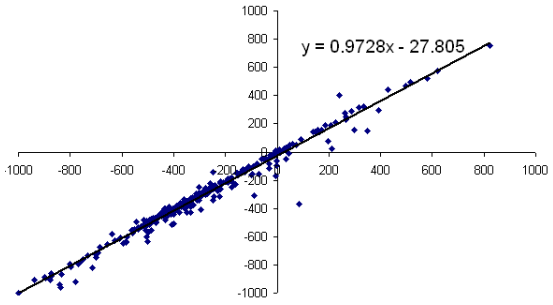

Fig. 6. Evaluation pairs at depths 2 and 4 in Stratego

[2]For details of these rules we refer to the International Stratego Federation (ISF), www.isfstratego.com.

[3]http://www.strategousa.org/wiki/index.php/Main_Page

[4]http://www.metaforge.net/

[5]All test positions for Stratego and Dice can be downloaded at http://www.personeel.unimaas.nl/Maarten-Schadd/TestSets.html

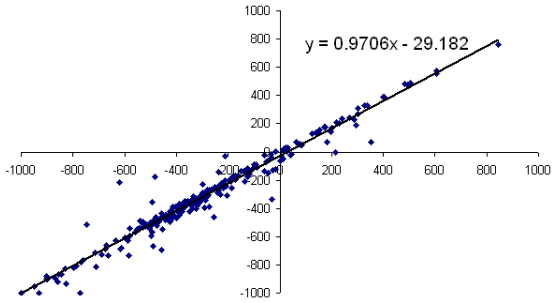

Fig. 7. Evaluation pairs at depths 3 and 5 in Stratego

2) Tuning Selectiveness: Next, we have to find the optimal value for PERCENTILE. If a too large value for PERCENTILE is chosen, the original search will always be performed. The reduced-depth searches will just cause an overhead. If a too small value for PERCENTILE is chosen, a selective search is created which might return incorrect values. For tuning this parameter, we look at the reduction of the game tree and the quality of the returned move. For this experiment, the regression model from Figures 6 and 7 are used at depths 4 and 5, respectively. For tuning, the PERCENTILE is varied. 300 positions were tested, from begin, middle and endgame situations. Tables I and II give the results of tuning the PERCENTILE parameter for depths 9 and 11, respectively.

TABLE I
SEARCH QUALITY FOR DEPTH 9 IN STRATEGO

PERCENTILE	ADE	Nodes	Reduction	Same Move
-	1.9	793,377,181	0%	79.3%
3.2	1.9	782,202,021	1.4%	80.1%
1.6	1.9	588,904,187	25.8%	80.0%
0.8	2.3	480,861,361	39.4%	79.7%
0.4	2.4	389,211,656	50.9%	78.5%
0.2	3.7	492,000,516	38.0%	75.9%
0.1	3.8	480,002,096	39.5%	77.4%
0.05	4.2	424,511,159	46.5%	75.9%

TABLE II
SEARCH QUALITY FOR DEPTH 11 IN STRATEGO

PERCENTILE	ADE	Nodes	Reduction	Same Move
-	1.9	17,797,964,580	0%	80.5%
3.2	2.4	15,781,147,865	11.3%	82.8%
1.6	5.4	9,433,921,421	47.0%	81.6%
0.8	6.5	5,576,812,889	68.7%	81.2%
0.4	8.3	4,216,966,331	76.3%	77.8%
0.2	27.2	2,927,878,965	83.7%	50.9%
0.1	26.5	2,496,375,320	86.0%	53.6%
0.05	25.4	2,353,464,841	86.8%	52.1%

In these tables, *ADE* is the average difference of the returned evaluation value. The percentage that the same move is returned is shown in the last column. The first row of each table is for reference. It indicates that even with the same technique for two different runs, only in approximately 80% of the cases the same move is returned. This is due to the random factor in the evaluation function. There exist moves that are equally good, according to the evaluation function. The random factor is the tie-breaker in that case. Nodes per second are hardly affected and therefore not shown.

In both tables we observe that it is possible to reduce the size of the tree significantly without a loss of quality. At depth 9, the tree is reduced with 39.4% of its size. At depth 11, a reduction of 68.7% is possible before the quality decreases.

In general we may observe that when the PERCENTILE parameter is decreased to a value of 0.2, the error grows, resulting in a larger difference of the evaluation score, and selecting weaker moves.

3) Selfplay: For forward-pruning techniques, a reduction of nodes searched cannot be seen as an indicator of improvement. Selfplay experiments have to be played in order to examine if CHANCEPROBCUT improves the playing strength.

We decided to test CHANCEPROBCUT with one second per move. Ten starting setups of equal strength were used, of which six were designed by De Boer, World Classic Stratego Champion in 2003, 2004 and 2007 [18]. The programs were matched on each possible combination of positions multiple times. Each match was played twice with different colors, to remove the advantage of the initiative. The results are shown in Table III.

TABLE III
SELFPLAY EXPERIMENT ON STRATEGO, 1 SECOND PER MOVE

Percentile	ChanceProbCut	Normal	Win rate
3.2	3,833	3,667	51.1%
1.6	3,831	3,669	51.1%
0.8	3,772	3,728	50.3%
0.4	3,809	3,691	50.8%
0.2	3,285	4,215	43.8%
0.1	2,846	4,654	37.9%
0.05	2,783	4,717	37.1%

For PERCENTILE values 1.6 and 3.2, a win rate of 51.1% is achieved, which is significant. For small values of PERCENTILE, the program clearly plays worse. The search has become too selective and makes many mistakes.

B. Dice

1) Determining Parameters: Because Dice has a regular game tree, chance nodes are counted as plies. We limit the applicability to $d \in \{7, 9\}$. R is set to 4 to handle the odd-even effect. On a test set of 1,000 5×5 positions value pairs (v_{d-R},v_d) have been determined and a regression line is calculated. We have chosen the 5×5 board for reference, because Hauk has used this variant to test node reductions of the STAR1 and STAR2 techniques [23]. In Figure 8 the model is shown for depths 3 and 7. Figure 9 shows the linear regression model for depths 5 and 9. These figures show that the linear regression model is suitable for estimating v_d.

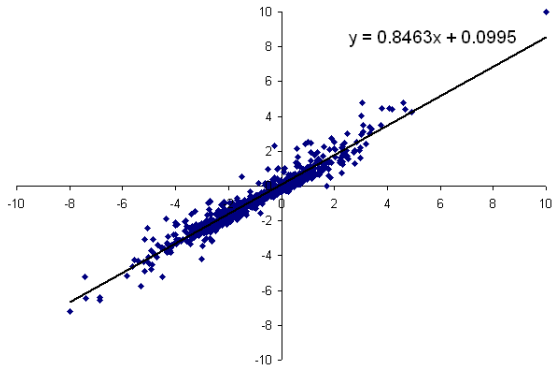

Fig. 8. Evaluation pairs at depths 3 and 7 in Dice

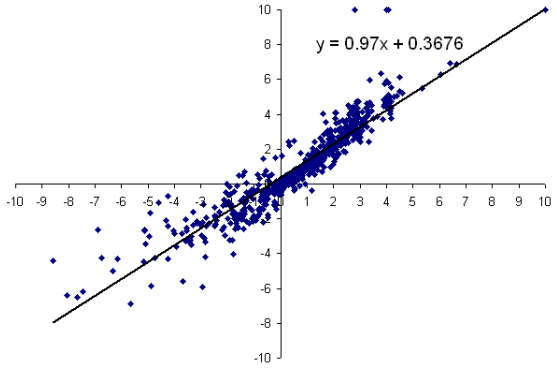

Fig. 9. Evaluation pairs at depths 5 and 9 in Dice

2) Tuning Selectiveness: Again, we have to find the optimal value for PERCENTILE. This tuning will be done in a similar fashion as in the previous subsection. For this experiment, at depths 7 and 9 the regression model from Figures 8 and 9 are used and the PERCENTILE is varied. 1,000 positions were tested with 5 up to 12 checkers on the board. Tables IV, V and VI give the results of tuning the PERCENTILE parameter for depths 9, 11 and 13, respectively.

In the three tables we observe that the average difference of the returned evaluation value increases when the PERCENTILE is decreased. Moreover, the amount of correct moves is slightly decreased. Table IV shows that when using PERCENTILE 0.05 for depth 9 a reduction of 64.6% can be achieved without a great loss in quality. In Table V, it may be observed that with the same values for the PERCENTILE parameter, a much larger reduction can be achieved. This is due to the fact that the regression model for estimating v_9 based on v_5 can be used more often. Furthermore, we see that the majority of the game tree can be pruned without a large deterioration of quality. Finally, Table VI shows that even a larger reduction is obtained at search depth 13.

TABLE IV
SEARCH QUALITY FOR DEPTH 9

PERCENTILE	ADE	Nodes	Reduction	Same Move
-	0.002	283,919,146	0%	77.8%
3.2	0.060	248,949,119	12.3%	80.8%
1.6	0.016	216,908,729	23.6%	80.5%
0.8	0.047	178,774,337	37.0%	80.1%
0.4	0.094	146,325,442	48.5%	76.1%
0.2	0.137	119,721,255	57.8%	77.2%
0.1	0.164	106,329,883	62.5%	75.2%
0.05	0.178	100,484,818	64.6%	76.1%

TABLE V
SEARCH QUALITY FOR DEPTH 11

PERCENTILE	ADE	Nodes	Reduction	Same Move
-	0.001	4,283,166,807	0%	83.5%
3.2	0.020	3,533,643,758	17.5%	86.0%
1.6	0.018	2,548,443,196	40.5%	85.5%
0.8	0.054	1,727,565,360	59.7%	83.6%
0.4	0.097	1,090,612,333	74.5%	80.3%
0.2	0.152	470,174,688	89.0%	80.6%
0.1	0.208	212,490,811	95.0%	79.9%
0.05	0.245	146,313,875	96.6%	78.7%

TABLE VI
SEARCH QUALITY FOR DEPTH 13

PERCENTILE	ADE	Nodes	Reduction	Same Move
-	0.001	62,135,790,037	0%	80.9%
3.2	0.005	50,148,996,143	19.3%	79.9%
1.6	0.024	35,969,087,210	42.1%	78.8%
0.8	0.062	24,316,045,552	60.9%	77.8%
0.4	0.100	13,649,007,595	78.0%	72.3%
0.2	0.122	17,532,456,401	71.8%	68.7%
0.1	0.218	1,353,392,195	97.8%	69.4%
0.05	0.256	798,171,386	98.7%	67.6%

A general observation in these tables is that the deeper the search, the larger the game-tree reduction is for the same PERCENTILEs. It indicates that for increasing depth, a larger PERCENTILE should be chosen for sustaining quality. In general, the search tree can be reduced to less than 40% of its original size without losing quality.

3) Selfplay: We decided to test CHANCEPROBCUT on the 11×11 board. There are two reasons why a large board size has to be chosen. (1) Previous experiments in Dice were conducted on the 11×11 board [9], [23]. (2) With games such as Dice, it is easy to perform a deep search. Our engine is able to evaluate more than 2 million nodes per second. In non-deterministic games, an increase in search depth has limited influence on the playing strength after the first few plies. Due to these reasons, a large board has to be chosen to create an interesting variant.

Table VII gives the results of the selfplay experiments on the 11×11 board using 100 ms. per move. With these time settings, our search engine reaches 9 plies in the begin game, and 13 plies in the endgame.

We see that CHANCEPROBCUT does have a small, but significant improvement in playing strength. With PERCENTILE set to 1.6, a win rate of 50.7% is achieved.

TABLE VII

SELFPLAY EXPERIMENT ON THE 11×11 VARIANT

Percentile	ChanceProbCut	Normal	Win rate
3.2	10,103	9,897	50.5%
1.6	10,133	9,867	50.7%
0.8	9,964	10,036	49.8%
0.4	10,060	9,940	50.3%
0.2	9,981	10,019	50.1%
0.1	9,982	10,018	50.1%
0.05	9,882	10,118	49.4%

VIII. CONCLUSIONS

In this article we have introduced the forward-pruning technique CHANCEPROBCUT for EXPECTIMAX. This technique is the first in its kind to forward prune in chance nodes.

CHANCEPROBCUT is able to reduce the size of the game tree significantly without a loss of decision quality in Stratego and Dice. At depth 11 in Stratego, the game tree can safely be reduced to 31.3%, for PERCENTILE value 0.8. In Dice, a safe reduction to 57.9% of the game tree with 13 plies can be achieved, using PERCENTILE value 1.6. Thus, the first conclusion we may draw, is that CHANCEPROBCUT finds a good move faster in the EXPECTIMAX framework, while not affecting the playing strength. Because CHANCEPROBCUT finds a good move faster, one might consider different approaches of investing the gained time. For instance, this time can be utilized for a more time-consuming evaluation function.

Selfplay experiments reveal that there is a small improvement in playing strength, which is still significant. In Stratego, CHANCEPROBCUT achieves a win rate of 51.1% and in Dice 50.7%. The small increase in playing strength is due to the nature of EXPECTIMAX. We point out two reasons. (1) The outcome of a game is dependent on luck. Even a weak player can win some games. (2) Deeper search has a small influence on the playing strength. Hauk showed that searching nine plies instead of five increased the win rate by only 2.5% [23]. A similar phenomenon was observed in Backgammon [30]. If we take this into account, CHANCEPROBCUT performs rather well. Thus, the second conclusion we may draw, is that CHANCEPROBCUT also improves the playing strength.

ACKNOWLEDGMENTS

This work is funded by the Dutch Organisation for Scientific Research (NWO) in the framework of the project TACTICS, grant number 612.000.525.

REFERENCES

[1] A. D. de Groot, *Thought and Choice in Chess*. The Hague - Paris - New York: Mouton Publishers, 1965.

[2] D. E. Knuth and R. W. Moore, "An Analysis of Alpha Beta Pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.

[3] D. F. Beal, "Experiments with the Null Move," in *Advances in Computer Chess 5*, D. F. Beal, Ed. Elsevier Science Publishers B.V., 1989, pp. 65–89.

[4] M. Buro, "ProbCut: An Effective Selective Extension of the Alpha-Beta Algorithm," *ICCA Journal*, vol. 18, no. 2, pp. 71–76, 1995.

[5] ——, "Experiments with Multi-Probcut and a New High-Quality Evaluation Function for Othello," in *Games in AI Research*, H. J. van den Herik and H. Iida, Eds. Maastricht University, The Netherlands, 2000, pp. 77–96.

[6] Y. Björnsson and T. A. Marsland, "Multi-Cut $\alpha\beta$-Pruning in Game-Tree Search," *Theoretical Computer Science*, vol. 252, no. 1-2, pp. 177–196, 2001.

[7] R. G. Carter, "An Investigation into Tournament Poker Strategy using Evolutionary Algorithms," Ph.D. dissertation, University of Edinburgh, Edinburgh, United Kingdom, 2007.

[8] D. Michie, "Game-Playing and Game-Learning Automata," in *Advances in Programming and Non-Numerical Computation*, L. Fox, Ed., Pergamon, New York, USA, 1966, pp. 183–200.

[9] T. Hauk, M. Buro, and J. Schaeffer, "Rediscovering *-minimax search," in *Computers and Games, CG 2004*, ser. Lecture Notes in Computer Science, H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu, Eds., vol. 3846. Springer-Verlag, 2006, pp. 35–50.

[10] B. W. Ballard, "The *-Minimax Search Procedure for Trees Containing Chance Nodes," *Artificial Intelligence*, vol. 21, no. 3, pp. 327–350, 1983.

[11] Y. Björnsson and T. A. Marsland, "Risk Management in Game-Tree Pruning," *Information Sciences*, vol. 122, no. 1, pp. 23–41, 2000.

[12] Y. Björnsson, "Selective Depth-First Game-Tree Search," Ph.D. dissertation, University of Alberta, Edmonton, Canada, 2002.

[13] G. Goetsch and M. S. Campell, "Experiments with the Null-move Heuristic," in *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer, Eds. Springer-Verlag, 1990, pp. 159–168.

[14] M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and E. C. D. van der Werf, "Enhanced Forward Pruning," *Information Sciences*, vol. 175, no. 4, pp. 315–329, 2005.

[15] S. J. J. Smith and D. S. Nau, "Toward an Analysis of Forward Pruning," College Park, MD, USA, Tech. Rep., 1993.

[16] Case No. 04-1344-KI, "*Estate of Gunter Sigmund Elkan, vs. Hasbro, INC. et al.*" 2005, District Court of Oregon.

[17] *Stratego Instructions*. Milton Bradley Co., 1986, obtained at http://safemanuals.com/.

[18] V. de Boer, "Invincible. A Stratego Bot," Master's thesis, Delft University of Technology, The Netherlands, 2007.

[19] V. de Boer, L. J. M. Rothkranz, and P. Wiggers, "Invincible - A Stratego Bot," *International Journal of Intelligent Games & Simulation*, vol. 5, no. 1, pp. 22–28, 2008.

[20] C. Treijtel and L. J. M. Rothkrantz, "Stratego Expert System Shell," in *GAME-ON 2001*, Q. H. Mehdi and N. E. Gough, Eds., 2001, pp. 17–21.

[21] K. Stengård, "Utveckling av Minimax-Baserad Agent för Strategispelet Stratego," Master's thesis, Lund University, Sweden, 2006, in Swedish.

[22] I. Satz, "The 1st Computer Stratego World Championship," *ICGA Journal*, vol. 31, no. 1, pp. 50–51, 2008.

[23] T. Hauk, "Search in Trees with Chance Nodes," Master's thesis, University of Alberta, Edmonton, Canada, 2004.

[24] J. Veness and A. Blair, "Effective Use of Transposition Tables in Stochastic Game Tree Search," in *Computational Intelligence and Games (CIG 2007)*, A. Blair, S.-B. Cho, and S. M. Lucas, Eds., Honolulu, HI, 2007, pp. 112–116.

[25] J. Schaeffer, "The History Heuristic," *ICCA Journal*, vol. 6, no. 3, pp. 16–19, 1983.

[26] S. G. Akl and M. M. Newborn, "The Principal Continuation and the Killer Heuristic," in *1977 ACM Annual Conference Proceedings*, ACM Press, New York, NY, USA, 1977, pp. 466–473.

[27] R. D. Greenblatt, D. E. Eastlake, and S. D. Crocker, "The Greenblatt Chess Program," in *Proceedings of the AFIPS Fall Joint Computer Conference 31*, 1967, pp. 801–810, Reprinted (1988) in *Computer Chess Compendium* (ed. D. N. L. Levy), pp. 56-66. B. T. Batsford Ltd., London, United Kingdom.

[28] J. D. Slate and L. R. Atkin, "CHESS 4.5: The Northwestern University Chess program," in *Chess Skill in Man and Machine*, P. W. Frey, Ed. New York, USA: Springer-Verlag, 1977, pp. 82–118, Second Edition.

[29] D. F. Beal, "Random Evaluations in Chess," *ICCA Journal*, vol. 18, no. 2, pp. 3–9, 1994.

[30] T. Hauk, M. Buro, and J. Schaeffer, "*-Minimax Performance in Backgammon," in *Computers and Games, CG2004*, ser. Lecture Notes in Computer Science, H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu, Eds., vol. 3846. Springer-Verlag, 2006, pp. 51–66.