Chapter 5

Forward Pruning in Chance Nodes

This chapter is an updated and abridged version of the following publication:

1. Schadd, M.P.D., Winands, M.H.M. and Uiterwijk, J.W.H.M. (2009). ChanceProbCut: Forward Pruning in Chance Nodes. *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, P.L. Lanzi, ed., pp. 178−185, IEEE press, Piscataway, NJ, USA.

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 (forward pruning) (De Groot, 1965). 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 (Knuth and Moore, 1975), can only prune safely if a position is proven to be irrelevant to the principal variation (backward pruning). There are several forward-pruning techniques for the $\alpha\beta$ algorithm, such as the null-move heuristic (Beal, 1989; Goetsch and Campell, 1990), (Multi-)ProbCut (Buro, 1995; Buro, 2000), and Multi-Cut (Björnsson and Marsland, 1999; Björnsson and Marsland, 2000; Björnsson and Marsland, 2001), but they cannot guarantee that the best move is not overlooked. These forward-pruning techniques have been applied to deterministic games with perfect information (e.g., Chess, Checkers, and Go), but so far not for non-deterministic or imperfect-information games.

Non-deterministic (stochastic) games introduce an element of uncertainty (Russell and Norvig, 2003), for instance by a roll of dice (e.g., Backgammon and Ludo, Carter, 2007). Besides non-deterministic games with perfect information, there exist deterministic games with imperfect information. Imperfect-information games hide information from the players (e.g., Cluedo (Van Ditmarsch, 2000; Van Ditmarsch, 2001), Scotland Yard (Sevenster, 2006), Stratego (De Boer, 2007) and Kriegspiel (Ciancarini and Favini, 2007)). Both non-deterministic and imperfect-information games are generally more complex than deterministic games of perfect information

(Reif, 1984; Condon, 1992). A game can also be non-deterministic and have imperfect information (Russell and Norvig, 2003) (e.g., games such as Poker (Billings *et al.*, 1998a), Bridge (Smith, Nau, and Throop, 1998) and Risk (Osborne, 2003)). In imperfect-information games, the actual state of the game is not known and a possible state is referred to as a *world*. When performing a tree search in such a game, all possible worlds need to be considered (Frank, 1990; Frank and Basin, 1998). Alternatively, imperfect-information games can be treated as non-deterministic games as if they contain an element of chance (Russell and Norvig, 2003). For example in Stratego, capturing an unknown piece can be treated as a chance event where the chance model is based on the remaining pieces on the board and the knowledge about the opponent. In order to integrate knowledge about the opponent, opponent modeling techniques may be used (Jansen, 1992; Carmel and Markovitch, 1993; Iida *et al.*, 1994; Billings *et al.*, 1998b; Donkers, 2003). These techniques may improve the playing strength (e.g., a win ratio of 55% in Stratego, Stankiewicz and Schadd, 2009).

For non-deterministic games, *expectimax* (Michie, 1966) is the main algorithm of choice. It extends the minimax concept to non-deterministic games, by adding chance nodes to the game tree. Expectimax may be applied to imperfect-information games as well, although alternatives exist (e.g., Partition Search, Ginsberg, 1996). So far, no specific expectimax forward-pruning technique has been designed for these chance nodes.

This chapter answers the fifth research question by proposing *ChanceProbCut*, a forward-pruning technique based on ProbCut (Buro, 1995). ChanceProbCut is able to stop prematurely the search at a chance node in the expectimax framework. This technique estimates values of chance events based on shallow searches. Based on the correlation between evaluations obtained from searches at different depths, ChanceProbCut prunes chance events in advance if the result of the chance node probably falls outside the search window.

The outline of the chapter is as follows. First the expectimax algorithm is explained in Section 5.1. Next, its variants Star1 and Star2 pruning are described in Section 5.2. Thereafter, Section 5.3 discusses forward pruning. We introduce the new forward-pruning technique ChanceProbCut in Section 5.4. Section 5.5 describes the games Stratego, Dice and ChanceBreakthrough, which are used as test domains. The experiments are presented in Section 5.6. Finally, Section 5.7 gives the conclusions and an outlook on future research.

5.1 Expectimax

Expectimax (Michie, 1966) is a brute-force, depth-first game-tree search algorithm that generalizes the minimax concept to non-deterministic games, by adding *chance nodes* to the game tree (in addition to MIN and MAX nodes). A chance node is added when a die roll occurs (non-deterministic) or information has to be revealed (imperfect information). 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 by the following formula:

$$expectimax(s) = \sum_i P(c_i) \times V(c_i) \tag{5.1}$$

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

We explain expectimax in the following example. Figure 5.1 depicts an expectimax tree. In the figure squares represent chance nodes, regular triangles MAX nodes and inversed triangles MIN nodes. In Figure 5.1, 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, $expectimax(A) = 0.9 \times -250 + 0.1 \times 250 = -200$.

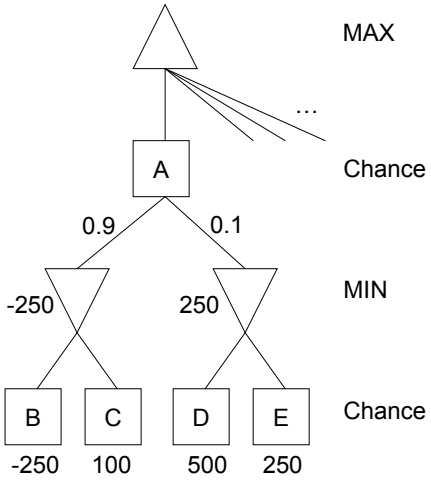

Figure 5.1: An example expectimax tree.

When searching the children of a chance node, the values of the children might be below or above the search window. However, it is possible that the combination of these values falls inside the search window. Therefore, the search window needs to be reset at every child of a chance node. Figure 5.2 shows an example. The value of chance node A is $0.6 \times 2 + 0.4 \times 10 = 5.2$, which is inside the window. If the $\alpha\beta$ search window (4, 6) is passed down to the first MIN node, the value of the first MIN node will be 4, instead of 2. Passing down the search window to the second MIN node has no effect in this case. However, the value of the chance node is incorrectly calculated as $0.6 \times 4 + 0.4 \times 10 = 6.4$ (correct is 5.2). This could even cause an incorrect pruning at the MAX node above A.

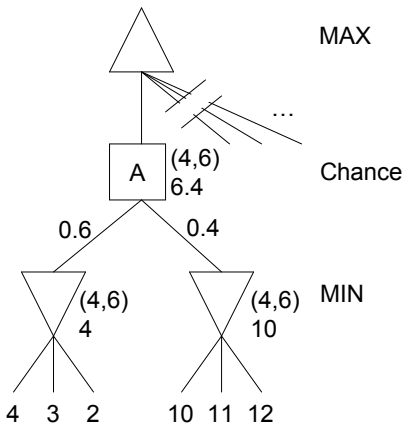

Figure 5.2: Search windows at chance nodes cause incorrect pruning.

5.2 Pruning in Chance Nodes

As stated by Hauk, Buro, and Schaeffer (2006a), the basic idea of expectimax (Ballard, 1983) is sound but slow. Star1 and Star2 exploit a bounded heuristic evaluation function to generalize the $\alpha\beta$ pruning technique to chance nodes (Ballard, 1983; Hauk *et al.*, 2006a). $\alpha\beta$ pruning imposes a search window (α, β) at each MIN or MAX node in the game tree. Remaining successor nodes can be pruned 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 with $\alpha\beta$ is that the search cannot stop at a chance node as soon as one successor falls outside the search window. To end the search, the weighted sum of all successors has to fall outside the search window. These techniques can be applied to non-deterministic games (Ballard, 1983; Hauk, 2004) and imperfect-information games (Stengård, 2006).

5.2.1 Star1

Star1 is able to create cutoffs if the lower and upper bound of the evaluation function are known (called L and U) (Ballard, 1983; Hauk, 2004). These bounds are the game-theoretic values 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 pruning takes place if

$$\frac{(V_1 + V_2 + \ldots + V_{i-1}) + V_i + L \times (N - i)}{N} \geq \beta \tag{5.2}$$

or

$$\frac{(V_1 + V_2 + \ldots + V_{i-1}) + V_i + U \times (N - i)}{N} \leq \alpha \tag{5.3}$$

where V_i is the value of node i and N the number of children, assuming that each child occurs with a probability $\frac{1}{N}$.

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. Safe pruning can be performed if the range defined by these bounds falls outside the search window. Instead of computing these bounds from scratch, it is possible to use an update rule (Ballard, 1983; Hauk, 2004).

Figure 5.3 demonstrates this pruning technique. The evaluation function is bound to the interval [-10, 10] in this example. Assume that node A is entered with an $\alpha\beta$ window of (5, 10). When the third child is searched, the theoretic upper bound for the chance node is $0.25 \times (2 + 4 - 8) + 0.25 \times 10 = 2$. Thus, even if the last child would return the maximum value, the value of node A will always be lower than α (i.e., 5). Thus, there is no need to search the last child.

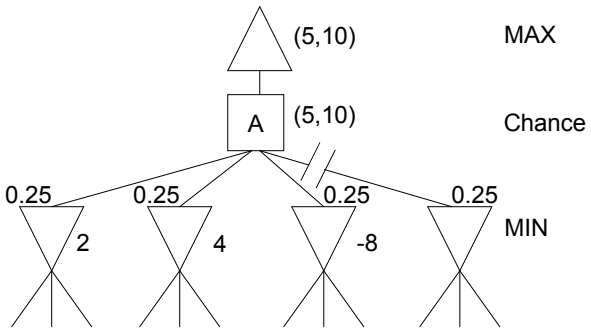

Figure 5.3: Successful Star1 pruning.

Star1 also enables the possibility to compute a window for the children of a chance node, which was previously not possible. The window is defined by the value which the next child should have, so that Formulas 5.2 and 5.3 fall outside the search window. Usually, only the last children to be visited benefit from this enhancement.

5.2.2 Star2

While Star1 returns the same value as expectimax, and uses fewer node expansions to obtain the same value, the amount of search reduction is generally not impressive (Hauk, Buro, and Schaeffer, 2006b). This is due to Star1's pessimistic nature. To obtain more accurate bounds for a node, Star2 probes each child (Ballard, 1983; Hauk, 2004) during a probing phase, which precedes the regular search phase. By only searching one of the available opponent moves, a bound for this node is obtained. This is a lower bound for a MAX node, and an upper bound for a MIN node. This bound is then backpropagated for calculating a more accurate bound for the chance node. A safe pruning is performed at a chance node followed by MAX nodes if

$$\frac{(V_1 + V_2 + \ldots + V_{i-1}) + V_i + (W_{i+1} + \ldots + W_N)}{N} \geq \beta \tag{5.4}$$

where W_i is the probed value for child i. At a chance node followed by MIN nodes, a pruning takes place if

$$\frac{(V_1 + V_2 + \ldots + V_{i-1}) + V_i + (W_{i+1} + \ldots + W_N)}{N} \leq \alpha \tag{5.5}$$

Formulas 5.4 and 5.5 assume that each child occurs with probability $\frac{1}{N}$, where N is the number of children.

We explain Star2 in the following example. Figure 5.4 depicts an expectimax tree with Star2 pruning. Node A is reached with an $\alpha\beta$ window of $(-150, 150)$. At this point, the theoretic 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 theoretic upper bound for A is now updated according to 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 theoretic range of A is $[-1000, -175]$ which is outside the $\alpha\beta$ window. Now nodes C and E can be pruned. Additional search effort would be caused if no pruning would occur. This effect can be nullified by using a transposition table. In case probing does not generate a cutoff, the probed values can be used for pruning or tightening the window during the regular search.

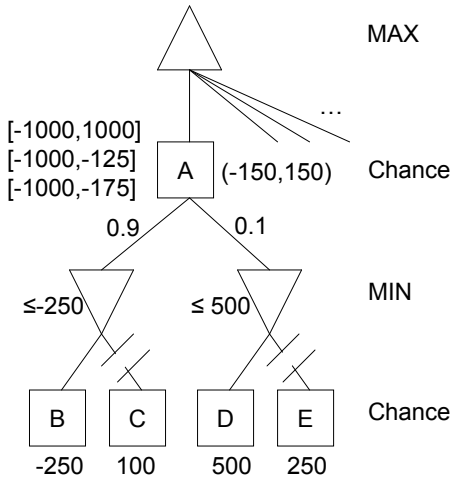

Figure 5.4: Successful Star2 pruning in the probing phase.

5.3 Forward Pruning

Human players are able to find good moves without searching the complete game tree. Using experience they are able to prune unpromising variations in advance (De Groot, 1965). Human players also select promising variants and search them deeper. In $\alpha\beta$ search this concept is known as *variable-depth search* (Marsland and Björnsson, 2001). The technique to abandon some branches prematurely is called *forward pruning*. The technique to search certain branches beyond the nominal depth is called *search extensions*. As such, the search can return a different value than a fixed-depth search.

In the case of forward pruning, the complete minimal game tree may not expanded (Björnsson, 2002), and good moves may be overlooked. However, the rationality is that although the search occasionally goes wrong, the time saved by pruning non-promising playing lines 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, forward pruning should have low risk, limited overhead, be applicable often, and be domain independent. Usually, improving one of these factors will worsen the others (Björnsson, 2002).

The null-move heuristic (Beal, 1989; Goetsch and Campell, 1990) is a well-known approach of forward pruning. Forfeiting the right to move is called a *null move*. This might be a legal move to play (e.g., Go), but in many games it is an illegal move (e.g., Chess). Instead of searching a position to depth d, the null-move is searched to depth $d-R$ (typically $R \in \{2,3\}$). If the result of this search is greater than β, a cutoff is made. The intuition is that generally making a move is better than passing (except in *Zugzwang* positions). Thus, if passing creates a cutoff, it is likely that a regular move creates a cutoff as well. The idea of using null moves in the search tree has been known for a long time (Adelson-Velskiy, Arlazarov, and Donskoy, 1975), and is nowadays used by most chess programs.

Multi-Cut pruning is another forward-pruning technique (Björnsson and Marsland, 1999; Björnsson and Marsland, 2001). Before examining an expected CUT node (cf. Section 2.3) to full depth, the first M child nodes are searched at a reduced depth $d-R$. If at least C child nodes fall outside the search window, a cutoff is produced. In general the behavior of Multi-Cut is as follows. The higher M and R are and the lower C is, the higher the number of prunings is. This technique was generalized later to ALL nodes (Winands *et al.*, 2005).

The *ProbCut* heuristic (Buro, 1995) 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* heuristic (Buro, 2000). Multi-ProbCut performs multiple shallow searches of different depths to decide whether a subtree is pruned.

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 (1993) set up a theoretic model of forward pruning in binary trees with chance nodes. No forward-pruning techniques for chance nodes in non-binary trees have been proposed so far.

5.4 ChanceProbCut

So far, it has not been investigated whether forward pruning can be beneficial at 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 (Buro, 1995), and uses this idea to generate cutoffs at chance nodes. 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 regular search is performed.

ChanceProbCut adapts the ProbCut idea to alter the lower and upper bounds for each possible event at a chance node. 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 (Buro, 1995). The bounds can be estimated by:

$$ELowerBound(v_d) = v_{d-R} \times a + b - t \times \sigma \tag{5.6}$$

and

$$EUpperBound(v_d) = v_{d-R} \times a + b + t \times \sigma \tag{5.7}$$

where a, b and σ are computed by linear regression, and t determines the size of the bounds. Using these bounds, it is possible to create a cutoff if the range of the chance node, computed by multiplying the bounds of each child with the corresponding probabilities, falls outside the search window.

At 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 regular search a cut is produced using a combination of transposition table probing (Veness and Blair, 2007), ChanceProbCut, Star2 and exact values, because the bounds are updated when new information becomes available.

When the regular search with depth d is started, the bounds obtained by ChanceProbCut can be used as a search window. It is unlikely that v_d falls outside this interval. We note that it is possible to perform a re-search after the regular 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 5.5. 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 child, changing

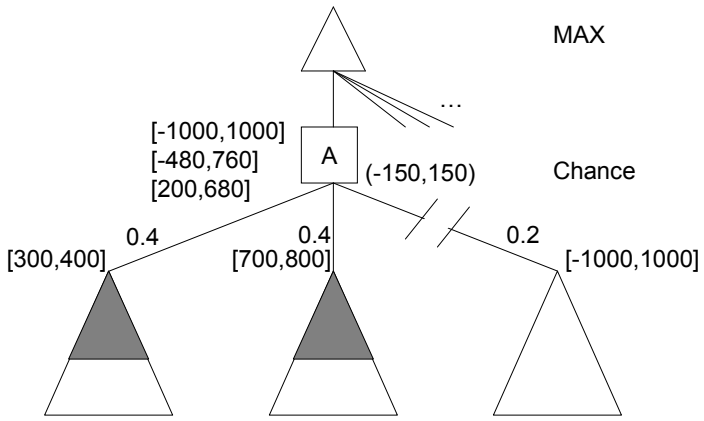

Figure 5.5: ChanceProbCut pruning.

the lower bound of the chance node to $300 \times 0.4 - 1{,}000 \times 0.6 = -480$ and the upper bound to $400 \times 0.4 + 1{,}000 \times 0.6 = 760$. The second ChanceProbCut search returns the window [700, 800]. Now, the lower bound of the chance node is computed as $300 \times 0.4 + 700 \times 0.4 - 1{,}000 \times 0.2 = 200$ and the upper bound is computed as $400 \times 0.4 + 800 \times 0.4 + 1{,}000 \times 0.2 = 680$. The range of the chance node, [200, 680], falls outside the search window and the next node is pruned.

Figure 5.6 depicts a second example, in which ChanceProbCut fails to prune any nodes. The search finds the same values for the first child as in the previous example. The next search reveals [0, 100] as bounds for the second child. This time, it is not possible to prune (with bounds [-80, 400] for the chance node). Even after the next search produces the bounds [50, 150] for the third child, the range of the chance node, [130, 230], does not fall outside the search window. However, after the regular search returns 400 for the first child, the search is terminated based on previously estimated ChanceProbCut values.

Finally, the pseudo code is shown in the next two algorithms. They are described in the negamax framework. Algorithm 5.1 describes the $\alpha\beta$ search part. Here it is assumed that after each move follows a chance node (Line 9). Algorithm 5.2 shows the pseudo code for a chance node. After initialization of the arrays for the bounds (Line 3−6), ChanceProbCut is used (Line 8−20). The confidence interval around the obtained value is computed in Line 13−14, based on linear regression (Buro, 1995). In the case that ChanceProbCut does not produce a pruning, the Star2 search is started (Line 22−32), which might improve the estimates obtained by ChanceProbCut. The *Probe()* procedure (Line 24) is used for obtaining an upper bound of the chance node (Hauk *et al.*, 2006a; Ballard, 1983). If Star2 also fails to obtain a pruning, the algorithm continues with the regular Star1 search (Line 34−44), which benefits from the bounds calculated earlier.

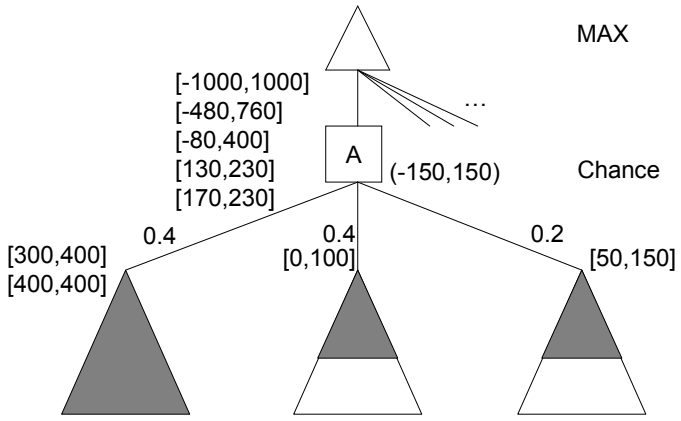

Figure 5.6: The regular search prunes with help of ChanceProbCut.

5.5 Test Domain

To test whether ChanceProbCut performs well, we use three games, Stratego, Dice, and ChanceBreakthrough.

5.5.1 Stratego

Stratego is a deterministic imperfect-information game. It was invented at least as early as 1942 by Mogendorff. The game was sold by the Dutch publisher *Smeets*

```
1: Search(alpha, beta, depth)
2:
3: if depth==0 then
4:   return eval()
5: end if
6:
7: for all Moves i do
8:   doMove(i);
9:   value = −ChanceNode(−beta, −alpha, depth−1);
10:  undoMove(i);
11:  if value ≥ beta then
12:    return beta;
13:  end if
14:  alpha = max(alpha, value);
15: end for
16:
17: return alpha;
```

Algorithm 5.1: $\alpha\beta$ search part of expectimax.

```
1: ChanceNode(alpha, beta, depth)
2:
3: for all ChanceEvents i do
4:    lowerBound[i] = ELowerBound[i] = L;
5:    upperBound[i] = EUpperBound[i] = U;
6: end for
7:
8: if depth > R then //ChanceProbCut
9: for all ChanceEvents i do
10:   doMove(i);
11:   v = Search(L, U, depth−1−R)
12:   undoMove(i);
13:   ELowerBound[i] = max(L, A × v + B − STDEV × t);
14:   EUpperBound[i] = min(U, A × v + B + STDEV × t);
15:   if ∑_j P_j× ELowerbound[j] ≥ beta then return beta;
16:   end if
17:   if ∑_j P_j× EUpperbound[j] ≤ alpha then return alpha;
18:   end if
19: end for
20: end if
21:
22: for all ChanceEvents i do //Star2
23:    doMove(i);
24:    v = Probe(lowerBound[i], upperBound[i], depth−1);
25:    undoMove(i);
26:    upperBound[i] = max(upperBound[i], v);
27:    EUpperBound[i] = min(upperBound[i], EUpperBound[i]);
28:    if upperBound[i] < ELowerBound[i] then ELowerBound[i] = L;
29:    end if
30:    if ∑_j P_j× upperbound[j] ≤ alpha then return alpha;
31:    end if
32: end for
33:
34: for all ChanceEvents i do //Regular Search with Star1
35:    doMove(i);
36:    v = Search(min(Star1LowerBound(), ELowerBound[i]),
37:         max(Star1UpperBound(), EUpperBound[i]), depth−1);
38:    undoMove(i);
39:    lowerBound[i] = upperBound[i] = v;
40:    if ∑_j P_j× lowerbound[j] ≥ beta then return beta;
41:    end if
42:    if ∑_j P_j× upperbound[j] ≤ alpha then return alpha;
43:    end if
44: end for
45:
46: return ∑_i P_i× lowerbound[i]
```

Algorithm 5.2: ChanceProbCut for chance nodes.

and Schippers between 1946 and 1951 (District Court of Oregon, 2005). In this subsection, we first describe briefly the rules of the game, then give the game-tree and state-space complexity and thereafter present related work.

Rules

The following rules are an edited version of the Stratego rules published by the Milton Bradley Company in 1986 (Milton Bradley Co., 1986). 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 each 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 two types of unmovable pieces, the Flag and the Bomb. An example initial position is depicted in Figure 5.7. 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 and 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
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

Figure 5.7: 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 must be moved like a rook in chess. The *Two-Squares Rule* and the *More-Squares Rule* prohibit moves which result in repetition.[1] 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.

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

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 is 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, is removed from the board when attempting to capture a 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 are no legal moves. The game is drawn if both players cannot move.

Game-Tree and State-Space Complexity

Based on a database of 4,500 human games, the average game length is estimated to be 318 plies with an average branching factor of 22. During a game, there are on average 30 captures of unknown pieces with on average 7 possible chance events (Arts, 2010). Thus, the game-tree complexity is $22^{318} \times 7^{30} \approx 10^{452}$. The state-space complexity of Stratego is computed by the following formula.

$$
\begin{aligned}
40 \times 40 \times \left(\sum_{\substack{red\\bombs}}^{y} \sum_{\substack{blue\\bombs}}^{z} \frac{39!}{(39-y)! \times y!} \times \frac{39!}{(39-z)! \times z!} \right) \times \\
\sum_{\substack{red\\spies}}^{a} \sum_{\substack{red\\scouts}}^{b} \cdots \sum_{\substack{blue\\generals}}^{s} \sum_{\substack{blue\\marshals}}^{t} \left(\frac{(90-y-z)!}{(90-y-z-a)! \times a!} \times \right. \\
\left. \frac{(90-y-z-a)!}{(90-y-z-a-b)! \times b!} \times \cdots \times \frac{(90-y-z-a-b-\cdots-s)!}{(90-y-z-a-b-\cdots-s-t)! \times t!} \right)
\end{aligned}
\tag{5.8}
$$

The first line takes care of the locations of the Flag and the Bombs, while the other two lines compute all possible positions of all other pieces. The number of free squares depends on the total number of pieces on the board. There are 92 (where 2 of them are always occupied by both the Flags) available squares minus the number of pieces on board. The upper bound of the state-space of Stratego is 10^{115}.

Previous Work

Stratego has received some scientific attention in the past. Treijtel and Rothkrantz (2001) created a player based on multi-agent negotiations. Stengård (2006) investigated different search techniques for this game. De Boer, Rothkrantz, and Wiggers (2008) described the development of an evaluation function using an extensive amount of domain knowledge in a 1-ply search. Schadd and Winands (2009) tested Evaluation-Based Quiescence Search in Stratego. At this moment, computers play Stratego at an amateur level (Satz, 2008). An annual Stratego computer tourna-

ment[2] is held on Metaforge with an average of six entrants (Satz, 2008; Schadd and Satz, 2008; Jug and Schadd, 2009).[3] Finally, we remark that some research has been done in Siguo, a four-player variant of Stratego (Xia *et al.*, 2005; Xia *et al.*, 2007; Lu and Xia, 2008).

5.5.2 Dice

The game of Dice is a two-player non-deterministic game with perfect information, recently invented by Hauk (2004), in which players take turns placing checkers on an $m \times m$ grid. One player plays columns, the other plays rows. Before each move, an m-sided die 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).

Based on 1,000 computer-played games for the 11×11 variant, the average game length is 78 plies with an average branching factor of 7. Each move is preceded by a roll of the dice, which can have 11 chance events. Thus, the game-tree complexity is $7^{78} \times 11^{78} \approx 10^{147}$. The state-space complexity is $3^{121} \approx 10^{58}$, because there a point in 11×11 grid can be empty, or be occupied by a white or black checker.

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 result of a Dice game is partially dependent on luck. A deep search is still beneficial. Hauk (2004) showed that a 9-ply player wins 65% against a 1-ply player.

Hauk (2004) used Dice to demonstrate the pruning effectiveness of Star-minimax algorithms in a non-deterministic game. Moreover, Veness and Blair (2007) used it to test StarETC, a variant of Enhanced Transposition Cutoffs (Schaeffer and Plaat, 1996).

5.5.3 ChanceBreakthrough

The game of ChanceBreakthrough is the non-deterministic variant of the deterministic Breakthrough game. The standard Breakthrough game was invented by Dan Troyka in 2001 and is played on an 8×8 checkers board (Handscomb, 2001). The pieces are set up in the two back ranks, Black at top and White at the bottom, as can be seen in Figure 5.8. White goes first and then players alternately move. A piece may move one space straight or diagonally forward if the target square is empty. However, it can only capture diagonally like a pawn in chess. The goal of the game is to "breakthrough" and reach the other side of the board before the opponent can achieve it. The game has been used to test Gibbs Sampling (Björnsson and Finnsson, 2009) and to learn search extensions (Skowronski, Björnsson, and Winands, 2010).

In ChanceBreakthrough each player has an extra type of move available at every turn: rolling the dice. Two eight-sided dice are rolled, one representing the rows, the other representing the columns. The resulting square on the board will be emptied. This move is preferable in deadlock situations or when the opponent player has

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

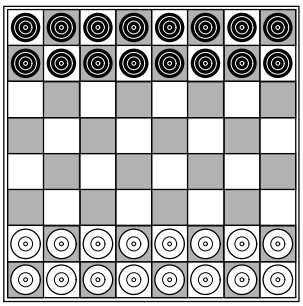

Figure 5.8: Initial position of ChanceBreakthrough.

more pieces than one self (i.e., the probability is higher to remove an opponent piece). Based on 1,000 selfplay games with 5 seconds per move, the game length is on average 53 plies and the average branching factor is 24. There are on average 14 possible chance events. The game-tree complexity is $(24 + 14)^{53} \approx 10^{84}$. The last row on each player's side is "private", because the game ends if an opponent would occupy a square on it. Taking this into account, we computed the state-space complexity of ChanceBreakthrough to be approximately 10^{25}, in a similar manner as Formula 5.8.

5.5.4 Game Engines

We implemented an expectimax engine for Stratego, Dice, and ChanceBreakthrough, enhanced with the Star1 and Star2 pruning algorithms (Ballard, 1983; Hauk, 2004). Furthermore, the history heuristic (Schaeffer, 1983), the killer heuristic (Akl and Newborn, 1977), transposition tables (Greenblatt *et al.*, 1967; Slate and Atkin, 1977) and StarETC (Veness and Blair, 2007) are used.

Stratego Engine

A well-performing evaluation function is an important factor for the playing strength of a Stratego program. For our program, we have chosen a material-based approach, partially based on research by De Boer (2007). The evaluation function is bound to [-1,000, 1,000] which corresponds to losing or winning the game. The evaluation function of a node is subtracted by the evaluated value of the root node. Because the evaluation function is bounded to [-1,000, 1,000] and the sum of the values of the pieces can exceed this interval, a bias is included to find a good move in positions where the evaluations of all leaf nodes would fall outside [-1,000, 1,000]. The piece values are shown in Table 5.1.

The Spy has a value of 100, but when the opponent's Marshal is captured, the value of the Spy is reduced to 10. Furthermore, the value of a Miner is dependent on the knowledge regarding the bombs, based on De Boer (2007). Depending on the game situation regarding the Flag of the opponent and the number of Miners left, a Miner may be valued as high as 200 points or as low as 10 points. Also, each square on the board is assigned a value of importance (i.e., increase the distance of

Table 5.1: Stratego piece values.

Spy	Scout	Miner	Sergeant	Lieutenant	Captain
100(10)	10	100*	20	50	100
Major	Colonel	General	Marshal	Bomb	Flag
140	175	300	400	75	1,000

the opponent to the own Flag, while decreasing the own distance to the opponent's Flag). The importance value of a square can be up to 50 points. A bonus is given if information of a piece is hidden. This bonus is set to 30% of the value of the piece. Furthermore, a variable counts the number of subsequent moves without capturing. For each two non-capture moves, the evaluation score is reduced by 1 point. Using this factor, more risk is taken when nothing happens in the game. The evaluation function is equipped with Evaluation Based Quiescence Search (EBQS) (Schadd and Winands, 2009). EBQS is able to estimate the quiescent value of a position without performing an actual quiescence search, which can result in a search explosion for Stratego. Removing large fluctuations in the evaluation function leads to a better prediction of deeper searches, and thus to better forward pruning. A small random factor (half a Scout) is included to prevent that games are (partially) repeated in the selfplay experiments.

When capturing occurs, a chance node is added to the tree. The probabilities of each event are based on the events in the past (i.e., piece movement, captured pieces). For instance, the opponent has eight unknown pieces left: 1 Flag, 1 Bomb, 1 General, 2 Colonels and 3 Captains. When attempting to capture a piece, the probabilities of encountering a General, Colonel and Captain are $\frac{1}{8}$, $\frac{2}{8}$, and $\frac{3}{8}$, respectively. If the piece has moved in the past, it cannot be a Flag or Bomb. In this case, the probabilities of encountering a General, Colonel and Captain are $\frac{1}{6}$, $\frac{2}{6}$, and $\frac{3}{6}$, respectively. The current approach does not lead to mixed strategies which could be necessary to conceal information. However, a possibility is to alternate the assignment of probabilities for the chance events to become less predictable.

Dice Engine

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 $[-10, 10]$. A small random factor ($\frac{1}{10}$ of a checker) is added, as well.

ChanceBreakthrough Engine

In Breakthrough, the most important factor is the number of pieces on the board. With more pieces it is easier to break through the lines of the opponent. In ChanceBreakthrough this is the most important factor as well, valued 200 points per piece. Furthermore, a progression factor was added to the evaluation function, rewarding 50 points for each row of the most advanced piece. The evaluation function is bound to the interval $[-1500, 1500]$. A random factor of 5 points was included.

5.6 Experiments and 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. Third, ChanceProbCut is evaluated in the game of ChanceBreakthrough. All experiments were performed on an AMD64 2.4 GHz computer.

5.6.1 Stratego

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

Determining Parameters

The first parameters to choose are the depth reduction R and the depths d at 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 do not count chance nodes as a ply for Stratego. While in theory this technique can be applied at 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. To find the parameters σ, a, and b for the linear regression model, 500 value pairs (v_{d-R}, v_d) have been determined. These value pairs are obtained from 200 begin, 200 middle, and 100 endgame positions, created using selfplay.[4] In Figure 5.9 the model is shown for depths 2 and 4. Figure 5.10 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 reasonable standard deviation.

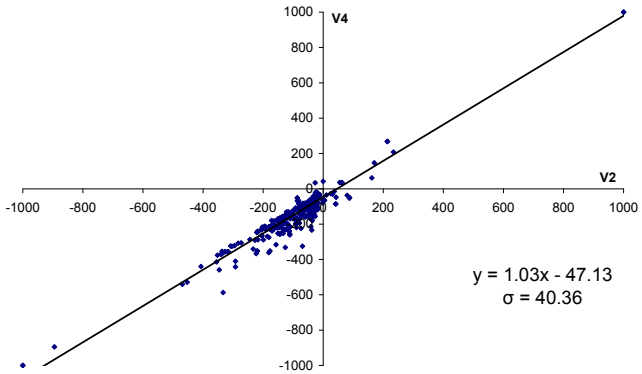

Figure 5.9: Evaluation pairs at depths 2 and 4 in Stratego.

[4] A position was stored after 50, 150 and 300 moves for begin, middle and endgame, respectively. All test positions for Stratego, Dice and ChanceBreakthrough can be downloaded from http://www.personeel.unimaas.nl/Maarten-Schadd/TestSets.html.

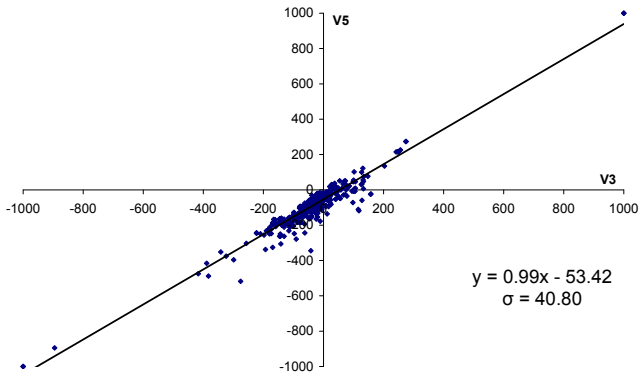

Figure 5.10: Evaluation pairs at depths 3 and 5 in Stratego.

Table 5.2: Performance of ChanceProbCut at depth 7 for 500 Stratego positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	55,207,171	-	-	-
12.8	45,293,207	17.6%	0.14	1.68
6.4	38,748,056	29.8%	0.26	2.88
3.2	34,483,433	37.5%	0.43	4.94
1.6	32,907,389	40.4%	0.43	4.94
0.8	30,661,362	44.5%	0.44	4.94
0.4	30,220,785	45.3%	0.67	7.32
0.2	30,149,981	45.4%	0.73	7.45
0.1	29,950,276	45.7%	0.73	7.45
0.05	29,922,575	45.8%	0.73	7.45

Tuning Selectiveness

Next, we have to find the optimal value for t. If a too large value for t is chosen, the regular search will always be performed. The reduced-depth searches will just cause an overhead. If a too small value for t is chosen, the search 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 models from Figure 5.9 and 5.10 are used at depths 4 and 5, respectively. 500 positions consisting of 200 begin, 200 middle, and 100 endgame situations were tested. Tables 5.2, 5.3 and 5.4 give the results of tuning the t parameter for depths 7, 9 and 11, respectively. For these experiments, the random factor was turned off.

In these tables, *ACPCME* is the average ChanceProbCut move error. This measure compares the value of the best move found by ChanceProbCut with the actual value of this move in a full search. *DCPCME* is the standard deviation of the ChanceProbCut move error.

In Tables 5.2, 5.3, and 5.4 we observe that it is possible to reduce the size of the

Table 5.3: Performance of ChanceProbCut at depth 9 for 500 Stratego positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	1,458,546,577	-	-	-
12.8	586,014,017	59.8%	0.43	3.84
6.4	357,070,424	75.5%	0.24	2.23
3.2	241,811,100	83.4%	0.56	2.83
1.6	215,023,725	85.3%	0.82	3.36
0.8	193,707,382	83.7%	1.16	4.27
0.4	189,763,738	87.0%	1.39	4.87
0.2	188,827,461	87.1%	1.49	5.08
0.1	187,542,414	87.1%	1.51	5.09
0.05	182,938,163	87.5%	1.51	5.11

Table 5.4: Performance of ChanceProbCut at depth 11 for 500 Stratego positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	33,251,941,846	-	-	-
12.8	12,140,806,418	63.5%	5.60	32.11
6.4	8,221,121,960	75.3%	12.56	56.46
3.2	6,305,673,987	81.0%	14.76	57.70
1.6	5,151,596,173	84.5%	16.26	58.21
0.8	4,796,794,009	85.6%	17.44	58.53
0.4	4,546,881,315	86.3%	17.71	58.55
0.2	4,477,761,845	86.5%	17.98	58.58
0.1	4,789,667,704	85.6%	18.04	58.60
0.05	4,763,428,798	85.7%	18.11	58.60

tree significantly without a loss of quality. In Table 5.2 we see that the tree can be reduced by 44.5% before the ChanceProbCut move is wrong by more than 0.5 point on average (i.e., $\frac{1}{20}$ of a Scout). At depth 9, the tree can be reduced with 75.5% of its size without almost no error at all. At depth 11, we observe that a more carful setting is needed. With t set to 12.8, the tree can be reduced by 63.5% with an average evaluation error of 5 points (half a Scout).

When performing the experiments, we noticed that when the t parameter is decreased, the error grows, resulting in a larger error of the best move, and a larger deviation. We also observed that at a larger depth, a more careful t is required. At larger depths ChanceProbCut is applied more often and the error rate increases.

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 whether ChanceProbCut improves the playing strength.

We decided to test ChanceProbCut with one second per move, typically reaching a search depth of 7 to 9 ply. This setting is close to tournament conditions, since in a tournament the thinking time is limited to five seconds per move. 100 starting positions were used to prevent that games were correlated. Each position was played with both colors, to remove the advantage of the initiative. 6,000 games were played to reach statistical significance. Due to the random factor in the evaluation function, different games were played even with the same initial position. The results are shown in Table 5.5. A 95% confidence bound is applied to the win ratio.

Table 5.5: Stratego selfplay experiment, 1 second per move.

t	ChanceProbCut	Regular	Win Ratio
12.8	2,981	3,019	49.7% ± 1.3%
6.4	3,009	2,991	50.1% ± 1.3%
3.2	3,102	2,898	51.7% ± 1.3%
2.4	3,043	2,957	50.7% ± 1.3%
2.0	3,002	2,998	50.0% ± 1.3%
1.8	3,108	2,892	51.8% ± 1.3%
1.6	3,072	2,928	51.2% ± 1.3%
1.4	3,059	2,941	50.9% ± 1.3%
1.2	3,066	2,934	51.1% ± 1.3%
1.0	3,107	2,893	51.8% ± 1.3%
0.8	3,113	2,887	51.9% ± 1.3%
0.6	3,035	2,965	50.6% ± 1.3%
0.5	3,073	2,927	51.2% ± 1.3%
0.4	3,084	2,916	51.4% ± 1.3%
0.3	3,077	2.923	51.3% ± 1.3%
0.2	3,114	2,886	51.9% ± 1.3%
0.1	3,063	2,937	51.1% ± 1.3%
0.05	3,018	2,982	50.3% ± 1.3%

For t values 0.2 and 0.8, a win ratio of 51.9% is achieved. For most other values of t, the program still performs well, taking into account the variance of the results. For t value 0.05, the search has become too selective and makes too many mistakes. For t values 6.4 and 12.8, not enough prunings are achieved to justify the overhead.

5.6.2 Dice

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

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 (2004) has used this variant to test node reductions of the Star1 and Star2 techniques. In Figure 5.11 the regression model is shown for depths 3 and 7. Figure 5.12 shows the linear regression model for depths 5 and 9. These figures show that the linear regression model is suitable for estimating v_d.

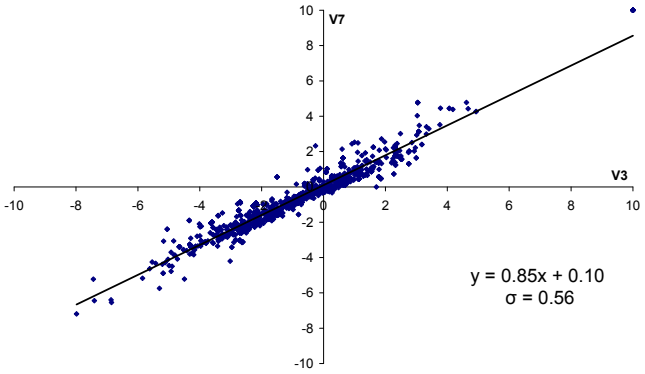

Figure 5.11: Evaluation pairs at depths 3 and 7 in Dice.

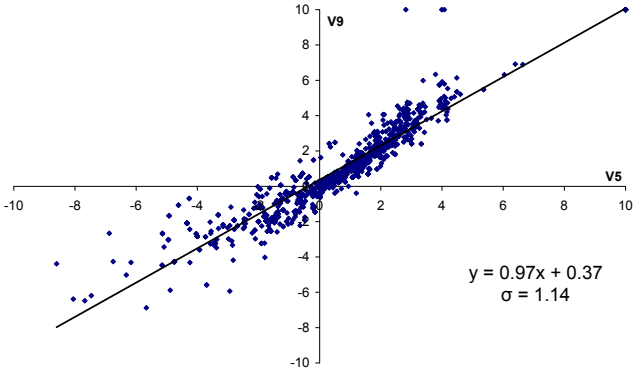

Figure 5.12: Evaluation pairs at depths 5 and 9 in Dice.

Tuning Selectiveness

Again, we have to find the optimal value for t. This tuning will be done in a similar fashion as described for Stratego. For this experiment, at depths 7 and 9 the regression model from Figure 5.11 and 5.12 are used and the t is varied. 1,000 positions were tested with 5 up to 12 checkers on the board. Tables 5.6, 5.7, and 5.8 give the results of tuning the t parameter for depths 9, 11 and 13, respectively.

In the three tables we observe that the average difference of the returned evaluation values increases when the t is decreased. Also the standard deviation of the

Table 5.6: Performance of ChanceProbCut at depth 9 for 1,000 Dice positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	190,822,592	-	-	-
3.2	163,926,279	14.1%	0.00	0.02
1.6	138,491,950	27.4%	0.01	0.05
0.8	112,607,261	41.0%	0.05	0.09
0.4	90,214,153	52.7%	0.10	0.15
0.2	72,684,352	61.9%	0.14	0.14
0.1	63,677,531	66.6%	0.16	0.15
0.05	59,189,953	69.0%	0.18	0.16

Table 5.7: Performance of ChanceProbCut at depth 11 for 1,000 Dice positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	2,322,871,121	-	-	-
3.2	1,945,429,167	16.2%	0.01	0.04
1.6	1,614,635,629	30.5%	0.02	0.05
0.8	1,292,924,069	44.3%	0.05	0.09
0.4	1,019,886,503	56.1%	0.09	0.11
0.2	830,281,706	64.3%	0.12	0.19
0.1	729,552,129	68.6%	0.14	0.15
0.05	679,098,869	70.8%	0.15	0.15

results grows when t is decreased. Table 5.6 shows that when using t=0.05 for depth 9 a reduction of 69.0% is achieved without a great loss in quality. Furthermore, we see that the majority of the game tree can be pruned without a large deterioration of quality. Finally, Table 5.8 shows that even a large reduction is obtained at search depth 13. Applying ChanceProbCut at a larger search depth did not lead to a reduction in quality.

Table 5.8: Performance of ChanceProbCut at depth 13 for 1,000 Dice positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	27,455,156,557	-	-	-
3.2	22,855,978,846	16.6%	0.01	0.02
1.6	18,819,031,785	31.5%	0.03	0.08
0.8	14,906,932,474	45.7%	0.06	0.11
0.4	11,707,480,207	57.4%	0.10	0.13
0.2	9,357,032,870	65.9%	0.13	0.14
0.1	8,313,736,712	69.7%	0.14	0.15
0.05	7,839,082,021	71.4%	0.15	0.15

A general observation for these tables is that the deeper the search, the larger the game-tree reduction is for the same t. In general, the search tree can be reduced by 50% before the ChanceProbCut value is mistaken by more than 0.10 point.

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 (Hauk, 2004; Hauk *et al.*, 2006a). (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 5.9 gives the results of the selfplay experiments of 20,000 games on the 11×11 board using 100 ms per move. With these time settings, the search engine reaches 9 plies in the opening phase, and 13 plies in the endgame. All games started from an empty board. Due to the dice element of the game and the random factor in the evaluation function, there exist no correlation between the games.

Table 5.9: 11×11 Dice selfplay experiment, 100 ms per move.

t	ChanceProbCut	Regular	Win Ratio
3.2	10,103	9,897	50.5% ± 0.7%
1.6	20,226	19,774	50.6% ± 0.5%
0.8	9,964	10,036	49.8% ± 0.7%
0.4	10,060	9,940	50.3% ± 0.7%
0.2	9,981	10,019	49.9% ± 0.7%
0.1	9,982	10,018	49.9% ± 0.7%
0.05	9,882	10,118	49.4% ± 0.7%

We see that ChanceProbCut does have a rather small, but genuine improvement in playing strength. With t set to 1.6, a win ratio of 50.6% ± 0.5% is achieved on 40,000 games.

5.6.3 ChanceBreakthrough

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

Determining Parameters

In ChanceBreakthrough we do not count chance nodes as a ply. We limit the applicability to $d \in \{3, 5\}$. R is set to 2 to handle the odd-even effect. On a test set of 500 positions, generated with selfplay, value pairs (v_{d-R}, v_d) have been determined and a regression line is calculated. In Figure 5.13 the model is shown for depths 1 and 3. Figure 5.14 shows the linear regression model for depths 2 and 4. Finally,

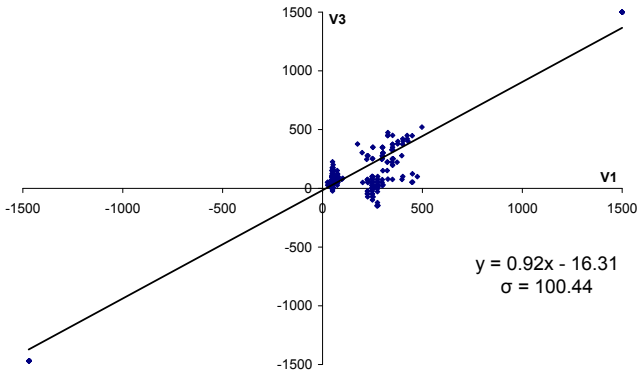

Figure 5.13: Evaluation pairs at depths 1 and 3 in ChanceBreakthrough.

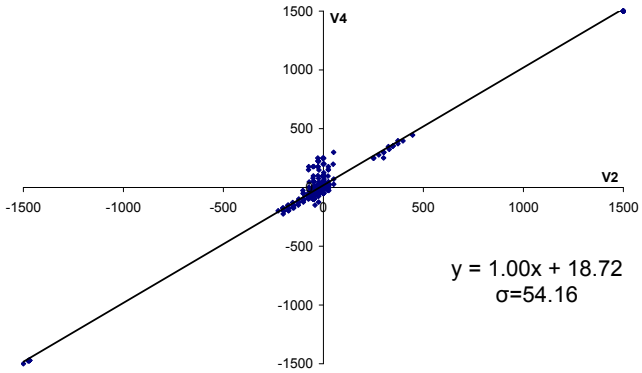

Figure 5.14: Evaluation pairs at depths 2 and 4 in ChanceBreakthrough.

Figure 5.15 depicts the model for depths 3 and 5. These figures show that linear regression is suitable for estimating v_d. It is harder to predict a 3-ply value based on a 1-ply search. However, the model stabilizes for deeper value pairs.

Tuning Selectiveness

We also tune t for the game of ChanceBreakthrough. The results for depth 5 on the test set of 500 positions can be seen in Table 5.10.

We see that a quite large error is being made. In a 5-ply search with $t = 3.2$, an error of 30.26 points is made on average (i.e., one seventh of a piece). Usually, the ChanceProbCut move does not make any error, but in some cases the returned value is far from correct, which is visible in the large standard deviation. These mistakes are always fatal in ChanceBreakthrough. Selfplay experiments should be concluded to find the appropriate t value.

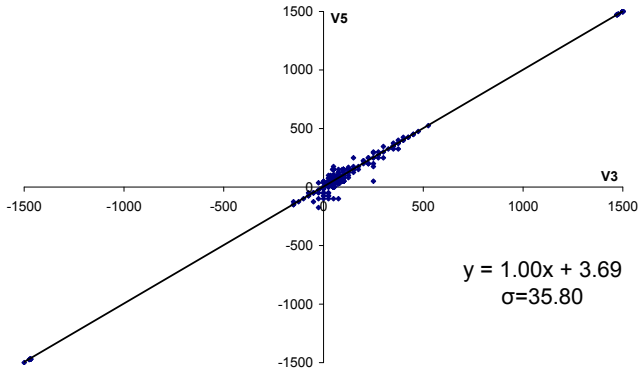

Figure 5.15: Evaluation pairs at depths 3 and 5 in ChanceBreakthrough.

Table 5.10: Performance of ChanceProbCut at depth 5 for 500 ChanceBreakthrough positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	23,496,940,171	-	-	-
3.2	11,377,214,094	51.6%	30.26	172.66
1.6	10,826,472,281	53.9%	32.02	182.73
0.8	9,102,499,638	61.3%	33.17	187.83
0.4	7,463,883,129	68.2%	34.05	190.26
0.2	6,466,057,217	72.5%	35.12	191.43
0.1	5,637,776,939	76.0%	35.58	192.07
0.05	5,359,416,856	77.2%	35.80	192.30

Selfplay

Table 5.11 gives the results of the selfplay experiments of 2,000 games using 5 seconds per move. With these time settings, the search engine reaches around 5 plies. We have chosen to test a different range of t. Based on our experience with the engine, we believe that the optimal value of t was somewhere between 0.8 and 2.0. Due to the chance element of the game and the random factor in the evaluation function, there exists no correlation between the games.

In the table we see that ChanceProbCut is able to improve the playing strength significantly. For the best setting, a win ratio of 54.8% $\pm$ 2.2% was achieved, despite the occasional error, as discussed in the previous section. This is a larger gain than observed for Stratego and Dice. To be sure that this number is accurate, we played an additional 2,000 games for $t = 1.6$. For the total of 4,000 games, this setting was able to achieve a win ratio of 54.4% $\pm$ 1.5%.

Table 5.11: ChanceBreakthrough selfplay experiment, 5 seconds per move.

t	ChanceProbCut	Regular	Win Ratio
2.0	1,057	943	52.9% ± 2.2%
1.8	1,041	959	52.1% ± 2.2%
1.6	1,096	904	54.8% ± 2.2%
1.4	1,064	936	53.2% ± 2.2%
1.2	1,051	949	52.6% ± 2.2%
0.8	1,029	971	51.5% ± 2.2%

5.7 Chapter Conclusions and Future Research

In this chapter we have proposed the forward-pruning technique ChanceProbCut for expectimax. This technique is the first in its kind to forward prune at chance nodes.

ChanceProbCut is able to reduce the size of the game tree significantly without a loss of decision quality in Stratego, Dice, and ChanceBreakthrough. At depth 11 in Stratego, a safe reduction of 85.6% in the number of nodes is observed for t value 0.8. In Dice, a safe reduction of 31.5% of the game tree with 13 plies can be achieved, using t value 1.6. At depth 5 in ChanceBreakthrough, ChanceProbCut prunes 53.9% of the search tree for $t = 1.6$. 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 in Stratego and Dice reveal that there is a small improvement in playing strength, which is still relevant. In Stratego, ChanceProbCut achieves a win ratio of 51.9% ± 1.3% and in Dice 50.6% ± 0.5%. The rather small increase in playing strength is due to the nature of expectimax. We point out two reasons. (1) The result of a game is dependent on luck. Even a weak player may win some games. (2) Deeper search has a small influence on the playing strength of expectimax, compared to minimax. For Dice, Hauk *et al.* (2006b) showed that searching 9 plies instead of 5 increased the win ratio by only 2.5%. A similar phenomenon was observed in Backgammon. If we take this into account, ChanceProbCut performs rather well. In ChanceBreakthrough however, a significant increase in performance was measured. ChanceProbCut was able to win 54.4% ± 1.5% on 4,000 games. The second conclusion we may draw, is that ChanceProbCut improves the playing strength.

We propose four directions for future research. (1) For improving the effectiveness of ChanceProbCut, more value pairs may be used. (2) The regression parameters and cut-threshold t can be bootstrapped according to the game phase. (3) A successor of ProbCut exists, called Multi-ProbCut (Buro, 2000). This technique could also be adapted for chance nodes (Multi-ChanceProbCut). (4) ChanceProbCut should be applied to other non-deterministic or imperfect-information games to test its effectiveness.