Strategies for the Automatic Construction of Opening Books

Thomas R. Lincke

Institut für Theoretische Informatik, ETH Zürich,
CH-8092 Zürich, Switzerland.
thomas.lincke@inf.ethz.ch

Abstract. An opening book is an important feature of any game-playing computer program. These books used to be constructed manually by an expert, by storing good moves suggested by theory, or simply by listing all games ever played by strong players [2,5,8]. Interest has recently shifted to automatic opening book construction where positions are selected by a best-first strategy, evaluated using a brute force search and then added to the opening book [3].
This paper presents the new "drop-out expansion" strategy for automatic opening book construction. It generalizes the previously used best-first strategy and reduces the opportunities for the opponent to force the player out of the book. The algorithm was used to calculate opening books for several games, including Awari and Othello, and helped to win the Awari tournament of the Computer Olympiad [10].

Keywords: opening book construction, expansion strategy, best-first, Awari, Othello

1 Introduction

Games are usually divided into three phases: opening, middlegame and endgame. In any of the three phases, the default action of a computer program is to start a brute force search for a "best" move. Since such a search has to be performed within a limited time, it can only examine nodes down to a certain depth, which means that the calculated value is only a heuristic approximation of the game-theoretic value.

However, in endgames a different approach is possible if the game has the convergence property [1], i.e. the number of pieces on the board decreases monotonically. In this case we are able to construct an endgame database of precalculated game-theoretic values for positions with a sufficiently small number of stones [9]. A computer program can benefit from such a database in two ways: first, the values of the endgame positions can be retrieved in one operation, and second, the retrieved values are exact game-theoretic values of the positions instead of approximations. The disadvantage is that we need additional space to store the database.

For openings an analogous approach is possible. Since the state space up to a few plies into a game is small, we can precalculate a database (called "opening book", or just "book") of values for positions that are likely to occur at the beginning of a tournament game. If the opening book is stored as a directed graph, with positions as nodes and

T.A. Marsland and I. Frank (Eds.): CG 2000, LNCS 2063, pp. 74–86, 2001.

moves as arcs, then the values can be propagated within the book. This way the computer program does not only save time compared to brute force search, but also obtains better values, assuming the search depth used for precalculation is greater than the one used during play. Again there is a tradeoff: on the one hand we save time (required for brute force search) and improve value accuracy, on the other hand we need additional space to store the opening book.

Although the benefits of using endgame databases and opening books are similar, algorithms for the construction of endgame databases have received more attention up to the present. The reason for this seems to be twofold: values in endgame databases are exact game-theoretic values, whereas values in the opening book are (mostly) heuristic values. This makes it hard to judge the usefulness of an opening book. Another reason is that efficient indexing functions for complete enumeration can be constructed for positions in endgame databases, whereas for positions in the opening book no efficient indexing functions do exist, due to the fact that the relevance of a position for the opening book stems from the position value and not from the configuration of stones on the board.

There are two approaches to automatic opening book construction. In the first approach, which may be called *passive book construction*, games are collected and, after analysis, used to update or extend the opening book. This approach is discussed in [5]. The basic idea is that if a move has been played before, then it is likely to be good and therefore worth to be added to the book. The main advantage of this approach is that it avoids losing twice using the same opening line: whenever a game is lost, that game is added to the book, which is then modified to play a different line the next time.

In the second approach, which may be called *active book construction*, the current opening book is analysed, and an expansion priority is assigned to all opening lines. The line with the highest priority is then expanded and the whole process is repeated [3]. The main advantage of this approach is that it does not depend on the availability of previously played games.

My main motivation for this work was the construction of an opening book for Awari. Because only a few dozen published Awari games exist, only active book construction could be used. However, for best results both approaches should be combined whenever possible.

In this paper I present a new expansion strategy for active book construction, called "drop-out expansion", which generalizes the previously used best-first strategy [3]. Section 2 sets the basic framework. Section 3 introduces the new expansion strategy and discusses its benefits compared to best-first strategy. Section 4 is a short description of OPLIB, an opening book construction tool, and section 5 summarizes the results.

2 Book Construction Basics

2.1 Book Representation

An opening book is represented as a directed graph. The nodes of the graph represent positions, and an arc between two nodes represents a legal move. One node, called the start node, represents the start position, and all other nodes must be reachable from it. If a node has an edge for each of its moves, then it is called an *interior node*, otherwise it is called a *leaf node*.

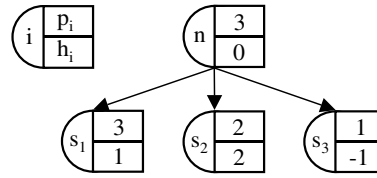

Fig. 1. Node representation

Every node i has two attributes: the heuristic value h_i and the propagated value p_i. The value h_i is computed by the search engine, using a brute force search. For interior nodes, p_i is the negamax value of p_{s_j} of all successor nodes s_j. For leaf nodes, p_i is equal to h_i.

$$p_i = \begin{cases} \max_{s_j}(-p_{s_j}) & \text{for interior nodes} \\ h_i & \text{for leaf nodes} \end{cases}$$

For book construction it is not actually necessary to keep h_i once i has become an interior node. However, for testing the expansion strategies, it is useful to be able to compare h_i and p_i during the calculation. Unless explicitely stated otherwise, let *value* mean *propagated value*.

Figure 1 shows an expample of a node with three successors. To improve readability, all figures in this paper use max-propagation only.

2.2 Book Expansion

When an opening book is created, it contains only the start node. Expansion is done in three steps:

1. Choose a leaf node and add all successors to the book
2. Calculate the values of the new successors
3. Propagate the values

This paper deals with the first step: how to choose the next node for expansion. Step two, the calculation of a (heuristic) value, is a topic of ongoing research in the field of computer games. I simply assume that a state-of-the-art search engine is available. Once the value is calculated, it will be negamax propagated through the graph (step three).

2.3 Goals

When is an opening book good? Of course, the ultimate goal in every tournament game is to win. Since it will rarely be possible to win a game straight from the book, its main benefit is that during the opening phase of the game search-time is saved, which may be used later in the game to outsearch the opponent. Therefore, the primary goal is to maximize the expected number of moves one can play within the book.

Which nodes should we expand to achieve this goal? A naive approach to guarantee a minimal number of moves from the book is to enumerate all positions at depth 1 from

the start position, calculate their values and store them. Next, the same is done for all positions at depth 2, and so on. This will, however, waste a lot of time and space on positions which are very unlikely to occur in a tournament game, because to reach them some player would have to make an obviously bad move. Therefore the secondary goal must be to achieve the primary goal with as few expansions as possible.

3 Expansion Strategies

The goals in Section 2.3 suggest that an expansion strategy should choose nodes for expansion according to the likelihood of their occurrence in a game. Since we can assume that good moves are more likely to occur than bad moves, the most straightforward strategy is best-first. This strategy was implemented in [3], it is used here as a reference against which I want to compare the new strategy.

For the following discussion of expansion strategies I will assume that the game engine using the book will only play optimal book moves. This is a reasonable assumption because there is no point in constructing an opening book and then to ignore it. However there are also good reasons to choose suboptimal moves now and then, for example to reduce predictability of the game engine.

3.1 Best-First Expansion

The rule for best-first expansion is: Expand the leaf node that is reached by following a path of best moves from the start node. If an interior node has more than one best move, choose one of them at random. See Figure 2.

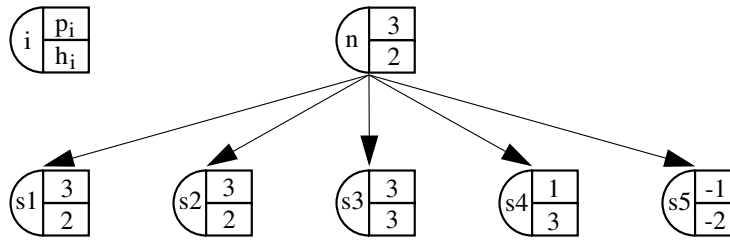

Fig. 2. Example for best-first expansion. The best value among all successors of n is 3, thus the candidate set for expansion is $\{s_1, s_2, s_3\}$

This strategy is simple and ignores bad moves, but it has a major flaw: suppose we just expanded the start position of chess, and the engine returned 0.1 (measured in pawns) for e2–e4, and 0 for all other moves. Expansion will now continue along e2–e4, and it is easily possible that the value for that move will always be > 0, so all other moves from the start node will be ignored forever. This violates the goal of maximizing the expected number of book moves, because the common move d2–d4 by the opponent leads to a position that is not in the book, i.e. we "drop out" of the book.

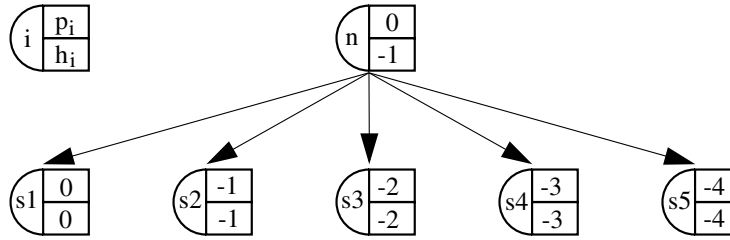

Fig. 3. Successor s_1 is the only best move

Of course the assumption that the value of the best move will only increase is weak. But even if we assume that the propagated value of a node will oscillate in an interval around its original heuristic value, the problem remains.

Take, for example, the situation in Figure 3, where successor s_1 currently is the only best move. Assume that during further expansion the propagated values remain in the range $[h_i - 2, h_i + 2]$. Then s_2 and s_3 may eventually have values larger than that of s_1, the favorite candidate, and thus become eligible for expansion. However if their values ever become less than -2 they will never be selected again, because the value of s_1 will never go below -2. At some point, a situation similar to Figure 4 will be reached, where the values of the alternative moves get stuck just below the lowest value which the optimal move has ever reached. The resulting successor values are misleading, because they are biased to look worse than what they are. This again leads to early drop-out, because the depth of the book after move s_2 will remain shallow, but it is still likely to be played by an opponent.

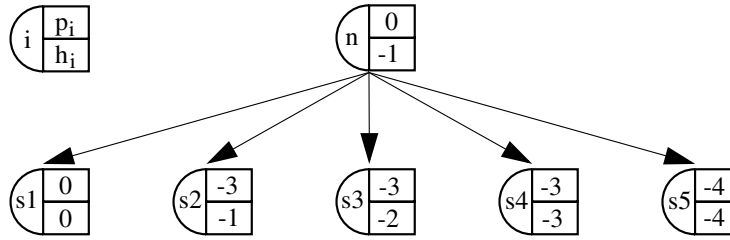

Fig. 4. The successors s_2 and s_3 are expanded until their values fall below -2

The heart of the problem with early drop-out is the fact that we use the same engine to search for values for both the first and the second player. We implicitely assume that the opponent uses the same evaluation function as we do, which is not necessarily true. It would be more reasonable to assume that the evaluation function of the opponent is only similar, and to expand some suboptimal move alternatives in a controlled way.

There is another minor flaw in best-first expansion. Because we make a random choice if more than one best move is available, the probability of choosing a specific leaf

node on a best path depends on the number of best moves at any node along the path. This means that the book grows faster along some paths. We could solve this if we first made a list of all leaf nodes reachable with best moves, and then chose one at random, but that would be both time and space consuming. Instead we prefer to have a strategy that solves this by making a series of local decisions.

3.2 Drop-Out Diagrams

A drop-out diagram shows the depth and value of all leaf nodes that can be reached, under the assumption that the book player only makes optimal moves and that the opponent is allowed to make any move. Figure 5 shows the possible drop-out nodes of a small Othello opening book calculated with best-first expansion. The root value of the book is 0.9, a leaf node with this value is only reached if the opponent only plays optimal moves too. If the opponent plays any suboptimal move, then leaf nodes with higher values may be reached.

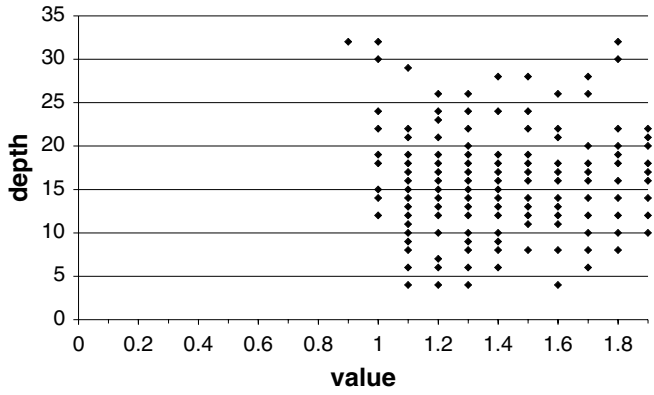

Fig. 5. Drop-out diagram of a small Othello book calculated with best-first expansion

Each dot in the diagram represents one or several candidate leaf nodes for expansion: Which one should we choose? Or, to reformulate the question, if you were the opponent of the book player: Which leaf node would you try to reach? Obviously, if the opponent only plays optimal moves, the book player will be able to play at least 32 plies from the book. However, as the plot shows, there is a line in the book which ends at ply 4, with a value that is only 0.2 points worse than the optimal value. If any opponent of this book player ever finds out about this line, he will start playing this line, and therefore this is probably the best candidate for expansion.

In a drop-out diagram like Figure 5, best-first strategy expands nodes from left to right. On the other hand, breadth-first strategy expands nodes from bottom to top. But what we are looking for is a strategy that expands nodes from bottom-left to top-right.

3.3 Drop-Out Expansion

The problem with best-first expansion was, that in situations as in Figure 4 only the best move is considered for expansion, whereas it is not unlikely that an opponent will play the second best move. To solve this we now consider all moves for expansion, and give each move a priority depending on the depth of the book following the move and the difference between the best value and the value of the move. A successor has high expansion priority if it is a good move and/or it has a shallow subbook, and a successor has low expansion priority if it is a bad move and/or it has a deep subbook. To calculate the expansion priorities we add two new attributes, epb_i and epo_i, to the nodes.

epb_i is the expansion priority for when it is the book player's move (1). It is initialized to zero in leaf nodes, and depends only on the expansion priority of the optimal successors. The $+1$ is the depth penalty, it guarantees that shallow nodes have higher priorities.

epo_i is the expansion priority for when it is the opponent's move (2). It is initialized to zero in leaf nodes, and depends on the expansion priority of all successors. Besides the depth penalty ($+1$), suboptimal moves get an additional penalty which depends on the value difference to the optimal move.

$$epb_i = \begin{cases} 1 + \min_{optimal\ s_j}(epo_{s_j}) & \text{for interior nodes} \\ 0 & \text{for leaf nodes} \end{cases} \quad (1)$$

$$epo_i = \begin{cases} 1 + \min_{s_j}(epb_{s_j} + \omega(p_i - p_{s_j})) & \text{for interior nodes} \\ 0 & \text{for leaf nodes} \end{cases} \quad (2)$$

ω is the weight for the difference $p_i - p_{s_j}$ between the optimal value and the value of the suboptimal successor s_j. It should be ≥ 0, the right choice of ω is game specific and depends on the heuristic value resolution, i.e. $+1$ may mean "one piece ahead" or "0.01 pieces ahead". A low value for ω means higher priority for suboptimal moves, if $\omega = 0$ then all successors will be expanded to the same depth, regardless of their values. On the other hand, if $\omega \to \infty$ then drop-out expansion degenerates into best-first expansion because only optimal moves will be expanded.

What do we gain if we use drop-out expansion? Obviously a move that is only slightly worse than the best move will not be ignored forever, even if the best value never decreases. With increasing depth of the best move, the priority for the expansion of suboptimal moves will increase too and this will eventually lead to their expansion. For the same reason it will not happen that a suboptimal move gets stuck with a bad value, as was shown in Figure 4 for best-first strategy. Thus all the problems observed with best-first expansion have been solved.

An additional benefit from drop-out expansion is that the parameter ω can be used to control the shape of the opening book. The user can choose any shape between full expansion of every line and best-first expansion.

The benefit from drop-out expansion may also be understood as a kind of insurance: if the opponent wants to force a drop-out, he has to pay with a move that is so bad that it has not been considered for expansion yet. If he keeps playing good moves, we will not drop out of the book early. So, at the end of the opening, we have either a good position, or we have saved lots of time, or a combination of both.

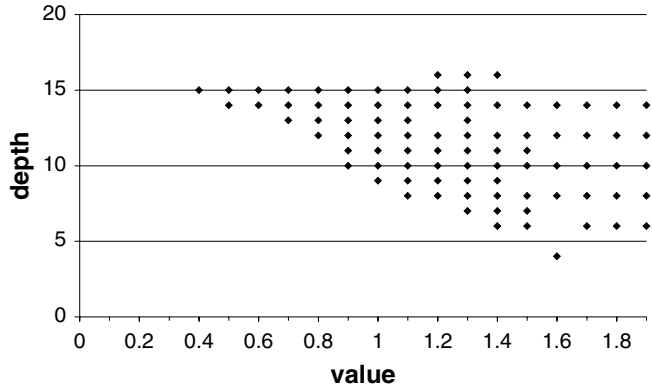

Fig. 6. Drop-out diagram of a small Othello book calculated with $\omega = 1.0$

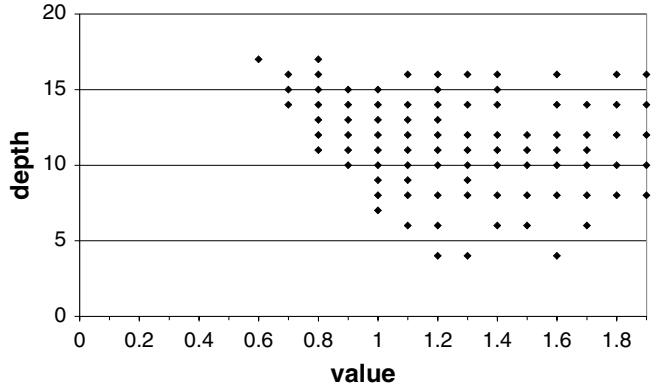

Fig. 7. Drop-out diagram of a small Othello book calculated with $\omega = 2.0$

3.4 Further Enhancements

So far our reasoning has led us to consider not only the values of successor nodes, but also the depths to which they have already been expanded. Thereby the subset of leaf nodes considered for expansion was changed to avoid the problems found with best-first expansion. However, the candidate set of leaf nodes changes most radically when the value of the start node changes. It would be of advantage if we could settle that value first.

This is exactly what conspiracy number search [6] [7] can do: Expand a leaf node that is reached by following a path of best moves from the start node and which is most likely to change the value of the start node. Two new attributes need to be added to each node, one to count the number of leaf nodes that have to change to increase the value of the node, and one to count the number of leaf nodes that have to change to decrease the value.

While conspiracy number expansion is useful to try to settle the value of a node with the smallest number of expansions possible, the expanded nodes may be way off in a

deep remote line, hardly useful with respect to the primary goal, the maximization of the number of expected moves in the book. Therefore conspiracy number expansion should be used only in conjunction with drop-out expansion, and only if the value of the start node is unstable.

Another enhancement is the use of fractional depth. The depth of the successor nodes is not incremented by 1 (as in (1) and (2)), but instead a value within the range $0.01 \leq fractd_i \leq 1$ is used, which depends on the difference between the best and the second best value. If the difference is large (or if there is only one move), then the fractional depth is small, otherwise large.

This favors the expansion of lines with unique or almost unique moves. The increased expansion of these moves is justified by the fact that they do not contribute to the exponential growth of the book as the other moves do.

3.5 Other Considerations

The task of opening book construction would be simplified if we had a good model of the opponent, i.e. if we could predict the opponent's moves with high accuracy. For drop-out expansion I proposed a linear function, $\omega(p_i - p_{s_j})$, for the value-dependant penalty. This is a very simple opponent model, where ω is the estimated similarity of the book player's engine and the opponent's engine. I also considered some non-linear functions, but abandoned the idea because of a lack of efficient implementations.

4 Implementation

All the strategies described in this paper, plus a few experimental ones, were implemented in OPLIB, a game independent software tool for two player games on cyclic graphs. OPLIB is based on an implementation of DCGs with attributed nodes and arcs. On top of the internal DCG, it provides a variety of features such as adding and editing of nodes and arcs, retrieval of lists of successors and predecessors, value propagation, transposition and cycle detection, and statistics. A well defined interface exists through which OPLIB can access game specific functions. So far, the game specific functions have been implemented for Awari, checkers, chess and Othello. Large opening books with about 500000 nodes have been constructed for Awari and Othello, and smaller ones for chess and checkers.

Storing an opening book as a DCG induces some overhead with respect to disk space. For instance, to store one Othello position in the book requires approximately 120 bytes on average, compared to a straightforward encoding of an Othello position requiring only 16 bytes. About 50% of the 120 bytes is used to store the graph structure (nodes and arcs), another 25% to store attribute information and the rest for a hash table for position lookup.

However, it does not make sense to optimize disk usage anyway. With 120 bytes per position we could store more than 80 million positions in 10GBytes, which is well within the limits of current disk storage technology.

The real bottleneck in opening book construction is the preprocessing time. In tournament games, the time available for one move is usually limited to about 3 minutes.

To ensure a competitive quality of the values in the leaf nodes, the search engine of the opening book expander must search for a comparable amount of time per position. Calculating 20 positions per hour, or 500 positions per day, it would take about 440 years to fill an opening book with 80 million positions.

To speed up the construction process, OPLIB can also be run in a distributed mode. Search engine clients running on a network of workstations can then fetch a position from the server, calculate a value and send it back to the server, which then updates the opening book and finds a new position for the client. Up to 50 workstations at a time have been used for book construction.

5 Results

The calculation of large opening books is an ongoing project. Currently I have books with about 300000 nodes for Othello and about 800000 nodes for Awari. Both were constructed using a mix of the strategies described in this paper, as well as some experimental ones, with drop-out expansion responsible for more than 80% of all expansions.

Figure 8 shows the depth distribution and the drop-out diagramm of the nodes in the Othello book. The majority of the nodes is concentrated in the depth range from 10 to 20 plies. The deepest lines were expanded to a depth of 41 plies. The fraction of the number of leaf nodes from depth 15 to 41 is almost constant between 90% and 95%, meaning that about one out of ten successors was expanded. With average successor counts between 10 and 15 moves, this suggests that the deep lines were expanded by best-first expansion. Note that state-of-the-art Othello engines usually can solve positions at depths between 35 and 40 plies. To avoid the influence of solved positions on the shape of the book, the engine was configured to solve positions at depth 43. Thus the book does not yet contain any solved positions.

In the Awari book (Fig.9), the nodes are spread out over a much larger range, the deepest lines go down to 91 plies. This node distribution can partially be attributed to the low average successor count of about 5 moves. But the Awari engine also makes use of endgame databases, which help to solve a substantial number of positions from the 5th ply on, and also help to narrow the book by easily refuting bad moves. This seems to prevent exponential growth of the number of nodes with increasing depth. It is conjectured that the shape of the book was mostly influenced by the endgame databases, and not by the choice of expansion strategy.

An open question is how to measure the performance of the expansion strategies against each other. For instance, a self-play test will always favor the best-first strategy, because it works best against similar opponents. On the other hand, if tested against a different engine, then the outcome might be more influenced by the relative strengths of the search engines than by the chosen expansion strategy.

6 Conclusions

I have shown that, for opening book construction, best-first expansion has certain deficiencies. For instance, it may completely ignore alternative moves with values only slightly inferior to the best value, and it has a tendency to stop expansion of suboptimal

Depth	Nodes	Leafs	%Leafs
0	1	0	0%
1	1	0	0%
2	3	0	0%
3	14	0	0%
4	60	23	38%
5	180	58	32%
6	631	282	45%
7	1709	1003	59%
8	3786	2456	65%
9	7240	5148	71%
10	13053	10149	78%
11	18851	15239	81%
12	25699	21867	85%
13	29230	25549	87%
14	30774	27479	89%
15	28681	25763	90%
16	26758	24317	91%
17	22911	20876	91%
18	18489	16927	92%
19	15091	13783	91%
20	12259	11091	90%

Depth	Nodes	Leafs	%Leafs
21	11234	10434	93%
22	7652	7001	91%
23	6341	5841	92%
24	4419	4021	91%
25	3963	3662	92%
26	2623	2374	91%
27	2139	1941	91%
28	1674	1487	89%
29	1556	1464	94%
30	861	798	93%
31	681	639	94%
32	402	368	92%
33	370	337	91%
34	272	245	90%
35	283	259	92%
36	191	175	92%
37	186	168	90%
38	136	128	94%
39	61	59	97%
40	24	23	96%
41	10	10	100%

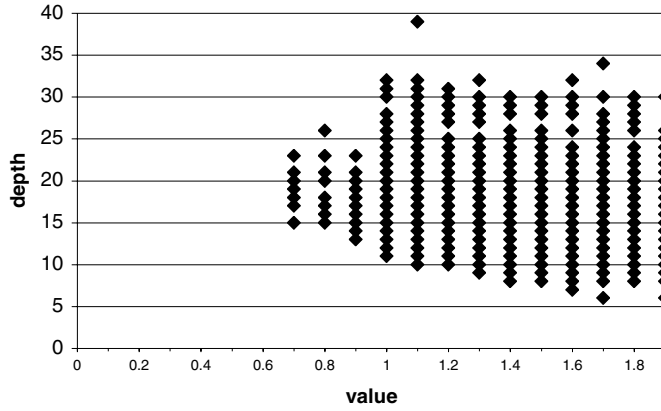

Fig. 8. Depth and drop-out statistics for the Othello book

moves with a misleadingly low value. In both cases, a lucky or an informed opponent can force us to drop out of the book with only a small penalty for him. Both problems are related to the implicit assumption that the opponent uses the same evaluation function.

I propose a new strategy, drop-out expansion, which, in a user controlled way, also considers suboptimal moves for expansion. This not only avoids the problems with best-first strategy, but also gives the user the flexibility to control the growth of the opening book between full-breadth expansion and best-first expansion. For best results, drop-out expansion can be combined into a mixed strategy with conspiracy number expansion.

The flexibility of drop-out expansion can also be used to tune a book to an opponent: if the opponent is known to play similar moves, then parameters can be chosen to construct a best-first like opening book. If the opponent is known to play differing moves often, then parameters can be chosen to expand more alternative moves.

Depth	Nodes	Leafs	%Leafs	Solved	%Solved
0	1	0	0%	0	0%
1	6	0	0%	0	0%
2	36	21	58%	0	0%
3	77	40	52%	0	0%
4	190	128	67%	0	0%
5	319	185	58%	1	0%
6	698	462	66%	1	0%
7	1161	844	73%	17	1%
8	1554	1122	72%	15	1%
9	2103	1560	74%	52	2%
10	2578	1934	75%	102	4%
11	3045	2310	76%	126	4%
12	3452	2643	77%	233	7%
13	3853	3018	78%	233	6%
14	3903	3055	78%	367	9%
15	3874	3021	78%	356	9%
16	3899	3008	77%	474	12%
17	4034	3118	77%	551	14%
18	4062	3183	78%	691	17%
19	3934	3089	79%	650	17%
20	3710	2865	77%	787	21%

Depth	Nodes	Leafs	%Leafs	Solved	%Solved
21	3734	2848	76%	830	22%
22	3948	3126	79%	1029	26%
23	3642	2809	77%	990	27%
24	3681	2918	79%	1134	31%
25	3400	2725	80%	1152	34%
26	3000	2372	79%	1114	37%
27	2857	2240	78%	1032	36%
28	2726	2101	77%	1161	43%
29	2839	2248	79%	1092	38%
30	2493	1867	75%	1271	51%
31	2810	2225	79%	1123	40%
32	2484	1845	74%	1580	64%
33	2850	2243	79%	1052	37%
34	2547	1922	75%	1726	68%
35	2863	2303	80%	1009	35%
36	2416	1787	74%	1769	73%
37	2927	2311	79%	1048	36%
38	2625	1960	75%	2017	77%
39	3129	2522	81%	1206	39%
40	2617	2031	78%	2029	78%
41	2721	2208	81%	1086	40%

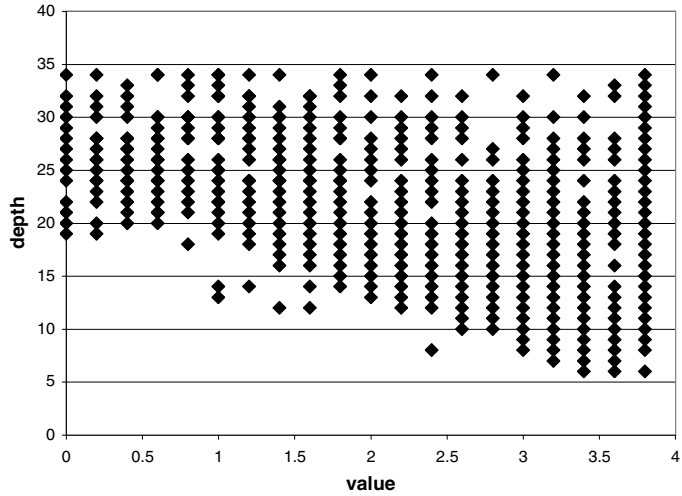

Fig. 9. Depth and drop-out statistics for the Awari book

All expansion strategies mentioned in this paper were implemented on OPLIB, an opening book tool. The results for Othello and Awari show the feasibility of automatic construction of large opening books. A tool like OPLIB may also play a major role in solving games in the future, because it supports the user in managing the solution tree (or solution graph) manually, as was the case with previously solved games [1][4].

Acknowledgments

My thanks go to Alvaro Fussen for letting me use his Othello engine, and to Nora Sleumer for her many helpful comments on earlier versions of this paper.

References

1. L. V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, Maastricht, The Netherlands, 1994.
2. M. G. Brockington. KEYANO Unplugged - The Construction of an Othello Program. Technical Report 97-05, Department of Computing Science, University of Alberta.
3. M. Buro. Toward Opening Book Learning. *ICCA Journal*, 22(2):98–102, 1999.
4. R. U. Gasser. *Harnessing Computational Resources for Efficient Exhaustive Search*. PhD thesis, ETH Zürich, 1995.
5. R. M. Hyatt. Book Learning - A Methodology to Tune an Opening Book Automatically. *ICCA Journal*, 22(1):3–12, 1999.
6. D. A. McAllester. Conspiracy Numbers for Min-Max Search. *Artificial Intelligence*, 35:287–310, 1988.
7. J. Schaeffer. Conspiracy Numbers. *Artificial Intelligence*, 43:67–84, 1989.
8. J. Schaeffer, R. Lake, P. Lu, and M. Bryant. Chinook: The World Man-Machine Checkers Champion. *AI Magazine*, 17(1):21–29, 1996.
9. K. Thompson. Retrograde Analysis of Certain Endgames. *ICCA Journal*, 9(3):131–139, 1986.
10. R. van der Goot and T. R. Lincke. Marvin Wins Awari Tournament. *ICGA Journal*, 23(3):173–174, 2000.