Algorithmic Fun - Abalone[*]

Oswin Aichholzer, Franz Aurenhammer, Tino Werner

Institute for Theoretical Computer Science
Graz University of Technology
Graz, Austria

e-mail: {oaich,auren}@igi.tu-graz.ac.at, tino@sbox.tu-graz.ac.at

Why games?

A main area of research and teaching at our institute is *algorithms design*. Loosely speaking, an algorithm is a method (technique, recipe) which transforms a given input – say a set of character strings – into the desired output – an alphabetically sorted list, for example. "Algorithm" is among the most basic notions in computer science. Beside the important role as a driving force in many fields of computing theory, as well as the countless practical applications, algorithms bear inherent beauty, elegance, and – fun. This may have been the motivation for many theoreticians to constantly contribute to this area. The present note, however, is not intended to present our research publications on algorithms design but rather shows that, in collaboration with students, projects of manageable size can provide all at the same time: teaching of ideas and concepts, fun, and improved results.

For half a century, board games (like chess) have often been considered as benchmark examples for artificial intelligence. Surprisingly enough, the most successful game programs to date are not really based on artificial intelligence. They mostly rely on three pillars: brute force (the power of computers), clever algorithmic techniques (algorithms design), and sophisticated evaluation strategies (developed by human intelligence). Along these lines, we present an algorithm we designed for a sophisticated program playing the board game *Abalone* on an advanced level.

Several famous people in computer science have contributed to the algorithmic theory of games. To name a few whose work is related to our topic, the fundamental principle of minimizing-maximizing strategies for games (described below) was invented by Claude E. Shannon around 1950. In 1975 Donald E. Knuth and Ronald W. Moore investigated the expected performance of alpha-beta pruning. Last not least, the mathematician Emanuel Lasker, World Chess Champion 1894-1921, developed several games and strategies.

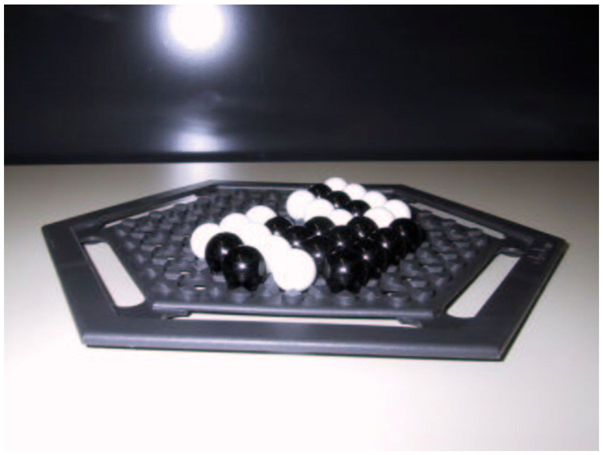


Figure 1: Playing makes fun. Algorithms make fun. Playing with algorithms is fun$^{\text{squared}}$.

Abalone

Though not much older than 10 years, Abalone is considered one of the most popular 'classical' board games. It may be the geometric appeal of the hexagonal board, the compactness of its rules, or simply the nice handling of the elegant marbles which makes Abalone an attractive game. From an algorithmic point of view, Abalone bears challenging questions. In this note, we report on a students project where Abalone[1] was chosen to implement a general search tree strategy, as well as to develop tailor-made heuristics and evaluation methods.

[*]Abalone is a registered trademark of Abalone S.A. - France

[1]For other games (like Four Wins, Scrabble, ...) we developed different algorithmic strategies.

Rules: Abalone is played by two players, black and white, each starting with 14 balls of the respectice color. The rules to move the balls are simple. If it is white's turn, then at most three aligned white balls may be moved (rigidly) by one position, choosing one of the six available directions. Thereby, black balls are pushed along if they lie in the line of movement and are in minority. Balls pushed off the board are out of the game. The goal is to be the first player that pushes out six balls of the opponent.

More details and information about this nice game can be found on the official Abalone web site at `http://www.abalonegames.com`.

Geometric score function

To guide the actions of the players, a means to evaluate a given Abalone constellation (i.e., a particular configuration of the balls on the playing board) is needed. As will become apparent soon, an important objective is to keep this procedure simple and efficient. The following geometric approach turned out to be surprisingly powerful. It relies on the intuitive insight that a strong player will keep the balls in a compact shape and at a board-centered position. This led us to use the static evaluation function, called *score function*, below.

1. Compute the centers of mass of the white and the black balls, respectively.

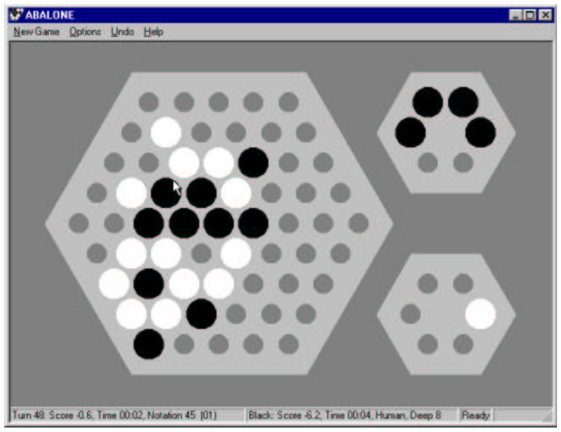
2. Take a weighted average of these centers and the center of the game board (which gets weighted with a factor below one). Call this reference point R.

3. Sum up the distances of all white balls to R (same for the black balls). Distances are measured along the six lines of movement on the board. (This is a hexagonal version of the well-known Manhattan metric.) Balls pushed off the board are granted a suitable constant distance.

4. The discrepancy of the two sums now gives the score of the position.

Minimax strategy

Abalone is a typical two-player, perfect information game. No randomness (like flipping a coin, rolling a dice, or dealing cards) is involved. When taking turns, the two players try to maximize and minimize, respectively, the score function – from the first player's view.



Turn 48: Score -0.6, Time 00:02, Notation 45 (01) | Black: Score -6.2, Time 00:04, Human, Deep 8 | Ready

Figure 3: Screen shot of the program.

Accordingly, the first player is called the *maximizer*, and his opponent the *minimizer*. To figure out good moves in a systematic way, the future development of the game is represented by a *decision tree*: the branches from the root of this tree (the start situation) reflect all possible moves for the maximizer, each leading to a new game position (root of a subtree) where it is the minimizer's turn. Again, all branches from this node display the possible moves, now for the minimizer, and so on.

Suppose now that the minimizer is playing optimally. Then, the best move for the maximizer will be one where the lowest score reachable by the minimizer is as high as possible; confer Figure 2. When looking ahead d steps, a tree of depth d with alternating maximizing/minimizing levels has to be considered. At the 'end' of this tree, namely at its leaves (and only there), the score function is computed for the corresponding game positions. Then, by traversing the tree (using depth-first-search) the maximizer can find the move which is *optimal when looking d steps ahead*. Clearly, the strength of the chosen move depends on how far we can think ahead. The deeper the tree, the more critical situations, impending traps, or winning moves can be taken into account.

Obviously, in a *globally optimal* strategy there is no artificial limit on the depth of the tree. All leaves are situations where the game comes to an end. Unfortunately, for most games this is not realistic; the tree would simply grow too fast (namely exponentially). The most powerful computers available today would not even find a first move unless we bound the depth of the tree. Still with a given bound d we have to be careful. The time complexity of searching the tree is

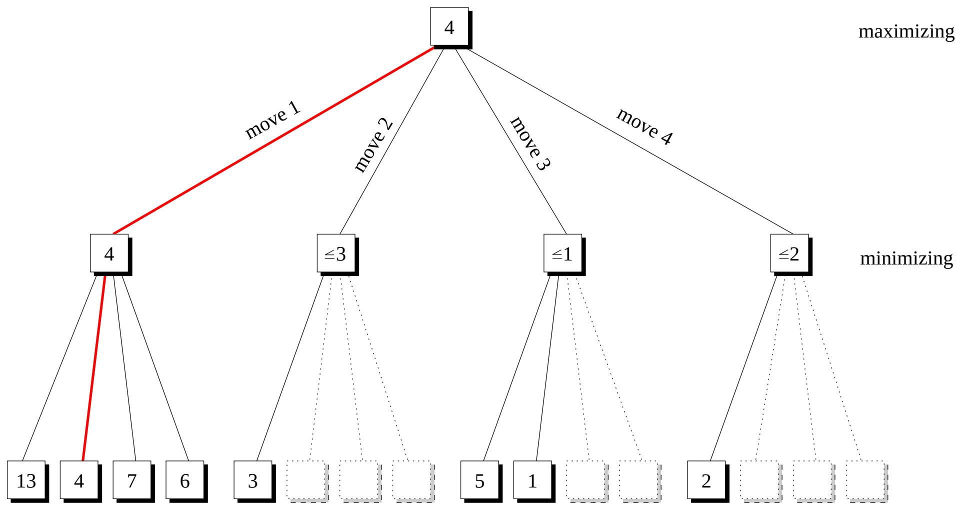


Figure 2: A decision tree with alpha-beta pruning at work: branches and leaves drawn dotted need *not* be considered. This saves time without loosing any information.

roughly B^d, where B is the branching factor (the number of possible moves at a certain position). A quick calculation: Abalone has about 60 possible moves for a typical position, and we want the computer to calculate ahead eight moves. Provided the program can evaluate one million positions per second (which means a lot of computation plus data handling etc.) we estimate $168 \cdot 10^6$ seconds for one move. In other words, you have to wait over 266 years to complete a game of, say, 50 turns. Our assumptions on the computational power are still to optimistic for today's personal computers – we should multiply by a factor of 50. It is now evident that computational power will not be the key to better Abalone playing; algorithmic improvement is asked for.

Alpha-beta pruning

Alpha-beta pruning is a method for reducing the number of nodes explored by the minimax strategy. Essentially, it detects nodes which can be skipped without loss of any information. Consider Figure 2 for an example. After examining the first move of the maximizer (the leftmost subtree) we know he can do a move of score at least 4. When trying his second move we find that the minimizer now has a move leading to score 3 on the first try. We conclude that there is no need to explore the rest of the second subtree: if there are moves exceeding score 3 then the minimizer will not take them, so the best the maximizer can get out of this subtree is score 3, which he will ignore since he already has a better move. In this (lucky) case we know – without looking at the remaining nodes – that no interesting move can be expected from the subtree. In this way, the number of traversed nodes can be significantly reduced.

"I think only one move ahead – but a good one!"
Emanuel Lasker, World Chess Champion 1894-1921, when asked how 'deep' he thinks

Technically, for each explored node alpha-beta pruning computes an α-value and a β-value, expressing the maximum and minimum score, respectively, found so far in related subtrees. With the help of these values it can be decided whether or not to consider further subtrees, because the score of a node will always be at least its α-value and at most its β-value.

Let us point out that this strategy is no heuristic at all: if the tree depth d is fixed, the calculated best move is independent from the use of alpha-beta pruning. The advantage lies in the number of skipped nodes and thus in the decrease of runtime. Observe further that the runtime depends on the order in which subtrees are visited. See Figure 2 again. If the minimizer first considers a move with a score higher than 4 (e.g., the third subtree) then we have to continue until a move with lower score is found. This gets particularly time-consuming if the first move of the maximizer is weak.

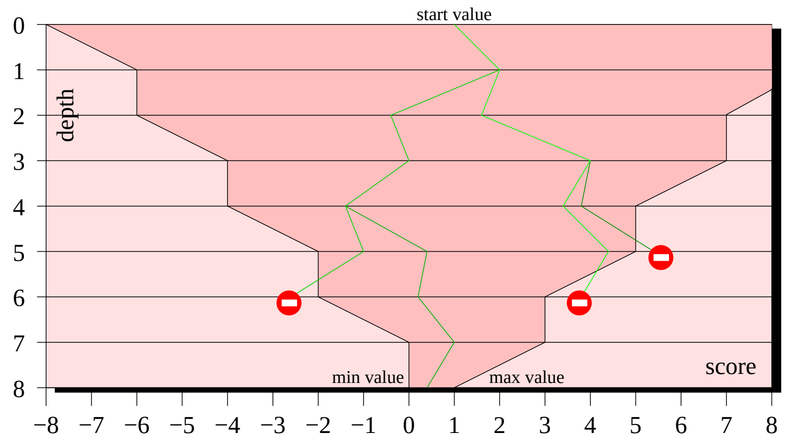


Figure 4: The funnel of our pruning heuristic helps to bypass unrealistically bad (or good) positions.

The best behavior will be obtained when the most promising moves (high score for maximizer, low score for minimizer) come first. It can be proved that such an *optimal alpha-beta pruning* explores about square root of the subtrees per node. (We get a complexity of $2 \cdot B^{d/2} - 1$ for d even, and $B^{(d+1)/2} + B^{(d-1)/2} - 1$ for d odd). This is still exponential in the 'look-ahead' d.

Clearly, optimal alpha-beta pruning cannot be managed in real games. A simple but good approximation is obtained when treating the moves at an actual position as leaves of the tree and presorting them by score.

Heuristic alpha-beta pruning

Heuristics for playing a game try to mimic what human players do: detect all interesting situations and avoid investigation of the majority of 'boring' moves. Usually this drastically reduces the number of work to be done, but bears the risk that tricky moves are overlooked. The ability to intuitively distinguish between strong and weak moves is considered one of the major advantages of human players over computer programs.

There exists a plethora of game heuristics in the literature, and no modern chess program could survive without them. Unlike decision trees and alpha-beta pruning, heuristics strongly depend on the characteristics of the game in question. We found none of the popular heuristics suited well for our purposes; a new method had to be developed.

Unlike in chess, the score of an Abalone position changes rather smoothly. One reason is that all the 'figures' (the balls) are of equal status. (For example, there are no kings which have to be protected under all circumstances.) Thus, for a reasonable move the resulting change in score is bounded, in a certain sense. This fact has an interesting implication. Assume we already have reached a very good score for a position of full depth d. Then we may skip a subtree of the decison tree whenever the score at its root is low (according to the above mentioned bound) and thus destroys the hope of reaching a better situation.

This observation works symmetrically for minimum and maximum, which leads to some kind of 'funnel' that restricts reasonable moves to lie in its interior; see Figure 4. A crucial point is how narrow this funnel can be made. For broad funnels, the heuristic will not discard a significant number of moves. If the funnel is too restrictive, on the other hand, we might skip strong moves and thus miss important positions. Choosing the optimum is, of course, strongly dependent on the specific game. For Abalone we determined the best value through careful experiments. Two observations were made when continuously changing the width of the funnel: while the improvement in runtime changed rather continuously, there was a critical threshold where the strength of play decreased dramatically. We considered the following criterion important in finding an optimal funnel width: playing on level $d + 1$ with heuristic alpha-beta pruning switched on should always be superior to playing on level d with the heuristic switched off.

Performance and strength

Let us first take a look at the efficiency of our program. Table 1 compares the different approaches in two extremal situations: starting phase (new opening) and end game. The table shows the average number

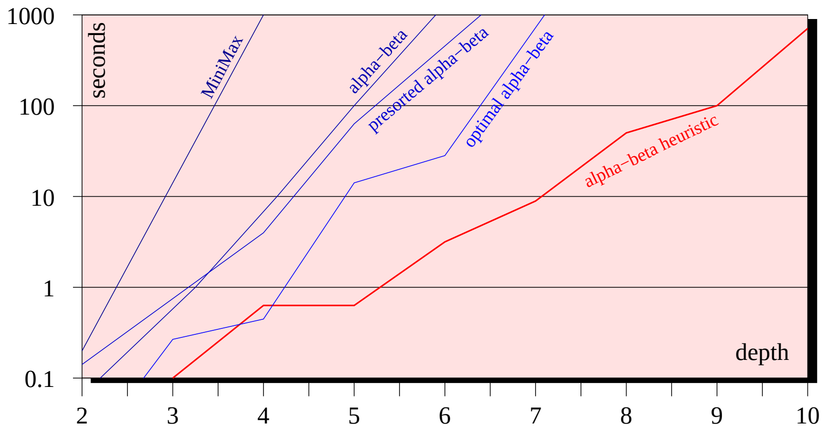


Figure 5: Comparison of runtime for the different approaches

of moves per position which had to be considered, i.e., the average branching factor B in the exponential complexity term before. We see that the employed strategies lead to significant improvements. In particular, our final alpha-beta heuristic cuts down B by a factor around 10 (!) compared to the simple minimax search. For the concrete example we gave earlier – where one move took over 5 years – this means that a move now can be found in about one second! (To be consistent with the PC times in Figure 5, we have to multiply these times by 50.) Figure 5 displays the runtime for different strategies and increasing depth levels, in the opening phase. The results for the ending phase are similar though less distinguishing between different strategies. As a satisfactory result, our heuristic clearly outperforms optimal alpha-beta pruning.

Strategy	Opening	Ending
Minimax strategy	60	65
Alpha-beta pruning	15	28
Presorted alpha-beta	13	13
Optimal alpha-beta	9	9
Heuristic alpha-beta	5	8

Table 1: Average number of moves per position

To evaluate the strength of our program we could win Gerd Schnider, three times World Champion of Abalone, as a human opponent. During the different stages of implementation and development we had several pleasant surprises. For example, the program already 'developed' its individual opening strategy. At first glance, Schnider was disappointed by this unconventional and 'obviously weak' opening – but after a closer analysis he considered it as quite strong. Moreover, when playing against the program at level 7, he got into serious trouble and couldn't beat it, despite retrying several strategies. Finally he joked, "Maybe you should look for another test person." This convinced us that, although a lot of future work remains to be done, our current implementation[2] already plays quite well and offers a challenge for every fan of Abalone.

"I played against the program quite a few times and it is really strong. I was not able to beat it in lightning games (around 5 seconds per move), mostly because it defends weaker positions extremely well. Tactically it can look ahead further than me but there are still some shortcomings in its strategy - to my opinion it should pay more attention to the compactness of the marbles, and it under-estimates the importance of the center a little. Anyway, it is much stronger than all other programs I played against. With its help, Abalone theory will improve much faster than today."
Gert Schnider, three times World Champion of Abalone

[2]A preliminary evaluation version of the program is available at `http://www.igi.TUgraz.at/oaich/abalone.html`