Alpha–beta pruning

From Wikipedia, the free encyclopedia

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.[1]

Contents

History

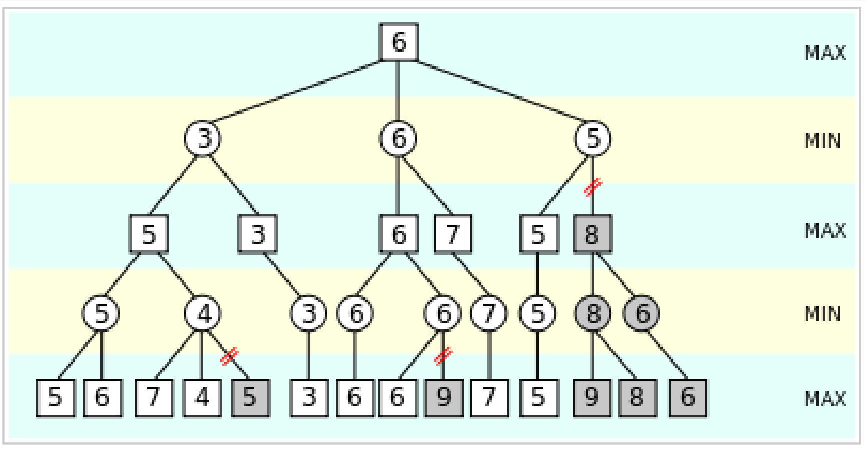
Allen Newell and Herbert A. Simon who used what John McCarthy calls an "approximation"[2] in 1958 wrote that alpha–beta "appears to have been reinvented a number of times".[3] Arthur Samuel had an early version and Richards, Hart, Levine and/or Edwards found alpha–beta independently in the United States.[4] McCarthy proposed similar ideas during the Dartmouth Conference in 1956 and suggested it to a group of his students including Alan Kotok at MIT in 1961.[5] Alexander Brudno independently discovered the alpha–beta algorithm, publishing his results in 1963.[6] Donald Knuth and Ronald W. Moore refined the algorithm in 1975[7][8] and Judea Pearl proved its optimality in 1982.[9]

Improvements over naive minimax

The benefit of alpha–beta pruning lies in the fact that branches of the search tree can be eliminated. This way, the search time can be limited to the 'more promising' subtree, and a deeper search can be performed in the same time. Like its predecessor, it belongs to the branch and bound class of algorithms. The optimization reduces the effective depth to slightly more than half that of simple minimax if the nodes are evaluated in an optimal or near optimal order (best choice for side on move ordered first at each node).

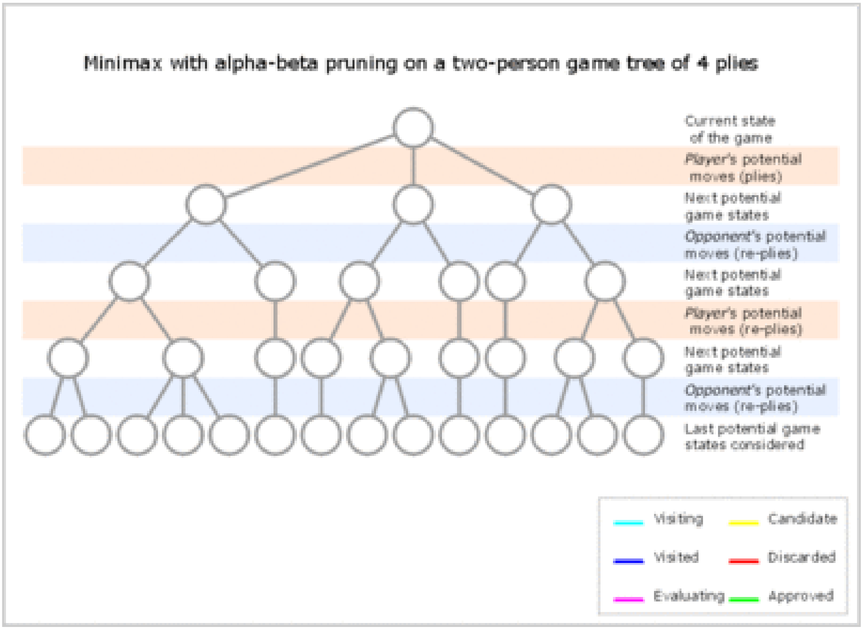
With an (average or constant) branching factor of b, and a search depth of d plies, the maximum number of leaf node positions evaluated (when the move ordering is pessimal) is $O(b*b*...*b) = O(b^d)$ – the same as a simple minimax search. If the move ordering for the search is optimal (meaning the best moves are always searched first), the number of leaf node positions evaluated is about $O(b*1*b*1*...*b)$

for odd depth and $O(b*1*b*1*...*1)$ for even depth, or $O(b^{d/2}) = O(\sqrt{b^d})$. In the latter case, where the ply of a search is even, the effective branching factor is reduced to its square root, or, equivalently, the search can go twice as deep with the same amount of computation.[10] The explanation of $b*1*b*1*...$ is that all the first player's moves must be studied to find the best one, but for each, only the best second player's move is needed to refute all but the first (and best) first player move—alpha–beta ensures no other second player moves need be considered. When nodes are ordered at random, the average number of nodes evaluated is roughly $O(b^{3d/4})$.[2]



An illustration of alpha–beta pruning. The grayed-out subtrees need not be explored (when moves are evaluated from left to right), since we know the group of subtrees as a whole yields the value of an equivalent subtree or worse, and as such cannot influence the final result. The max and min levels represent the turn of the player and the adversary, respectively.

Normally during alpha–beta, the subtrees are temporarily dominated by either a first player advantage (when many first player moves are good, and at each search depth the first move checked by the first player is adequate, but all second player responses are required to try to find a refutation), or vice versa. This advantage can switch sides many times during the search if the move ordering is incorrect, each time leading to inefficiency. As the number of positions searched decreases exponentially each move nearer the current position, it is worth spending considerable effort on sorting early moves. An improved sort at any depth will exponentially reduce the total number of positions searched, but sorting all positions at depths near the root node is relatively cheap as there are so few of them. In practice, the move ordering is often determined by the results of earlier, smaller searches, such as through iterative deepening.


Minimax with alpha-beta pruning on a two-person game tree of 4 plies
Current state of the game
Player's potential moves (plies)
Next potential game states
Opponent's potential moves (re-plies)
Next potential game states
Player's potential moves (re-plies)
Next potential game states
Opponent's potential moves (re-plies)
Last potential game states considered
Visiting
Candidate
Visited
Discarded
Evaluating
Approved
An animated pedagogical example that attempts to be human-friendly by substituting initial infinite (or arbitrarily large) values for emptiness and by avoiding using the negamax coding simplifications.

The algorithm maintains two values, alpha and beta, which represent the maximum score that the maximizing player is assured of and the minimum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity, i.e. both players start with their lowest possible score. It can happen that when choosing a certain branch of a certain node the minimum score that the minimizing player is assured of becomes less than the maximum score that the maximizing player is assured of (beta<=alpha). If this is the case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, the other branches of the node do not have to be explored.

Additionally, this algorithm can be trivially modified to return an entire principal variation in addition to the score. Some more aggressive algorithms such as MTD(f) do not easily permit such a modification.

Pseudocode

The pseudo-code for the fail-soft variation of alpha-beta pruning is as follows:[10]

```
function alphabeta(node, depth, α, β, maximizingPlayer)
     if depth = 0 or node is a terminal node
         return the heuristic value of node
     if maximizingPlayer
         v := -∞
         for each child of node
             v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
             α := max(α, v)
             if β ≤ α
                 break (* β cut-off *)
         return v
     else
         v := ∞
         for each child of node
             v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
             β := min(β, v)
             if β ≤ α
                 break (* α cut-off *)
         return v
```

```
(* Initial call *)
alphabeta(origin, depth, -∞, +∞, TRUE)
```

With fail-soft alpha-beta, the alphabeta function may return values (v) that exceed ($v < \alpha$ or $v > \beta$) the α and β bounds set by its function call arguments. In comparison, fail-hard alpha-beta limits its function return value into the inclusive range of α and β.

Heuristic improvements

Further improvement can be achieved without sacrificing accuracy, by using ordering heuristics to search parts of the tree that are likely to force alpha–beta cutoffs early. For example, in chess, moves that take pieces may be examined before moves that do not, or moves that have scored highly in earlier passes through the game-tree analysis may be evaluated before others. Another common, and very cheap, heuristic is the killer heuristic, where the last move that caused a beta-cutoff at the same level in the tree search is always examined first. This idea can also be generalized into a set of refutation tables.

Alpha–beta search can be made even faster by considering only a narrow search window (generally determined by guesswork based on experience). This is known as *aspiration search*. In the extreme case, the search is performed with alpha and beta equal; a technique known as *zero-window search*, *null-window search*, or *scout search*. This is particularly useful for win/loss searches near the end of a game where the extra depth gained from the narrow window and a simple win/loss evaluation function may lead to a conclusive result. If an aspiration search fails, it is straightforward to detect whether it failed *high* (high edge of window was too low) or *low* (lower edge of window was too high). This gives information about what window values might be useful in a re-search of the position.

Over time, other improvements have been suggested, and indeed the Falphabeta (fail-soft alpha-beta) idea of Fishburn is nearly universal and is already incorporated above in a slightly modified form. Fishburn also suggested a combination of the killer heuristic and zero-window search under the name Lalphabeta ("last move with minimal window alpha-beta search".)

Other algorithms

More advanced algorithms that are even faster while still being able to compute the exact minimax value are known, such as SCOUT,[11] Negascout and MTD-f.

Since the minimax algorithm and its variants are inherently depth-first, a strategy such as iterative deepening is usually used in conjunction with alpha–beta so that a reasonably good move can be returned even if the algorithm is interrupted before it has finished execution. Another advantage of using iterative deepening is that searches at shallower depths give move-ordering hints, as well as shallow alpha and beta estimates, that both can help produce cutoffs for higher depth searches much earlier than would otherwise be possible.

Algorithms like SSS*, on the other hand, use the best-first strategy. This can potentially make them more time-efficient, but typically at a heavy cost in space-efficiency.[12]

See also

- Pruning (algorithm)
- Branch and bound
- Minimax
- Combinatorial optimization
- Negamax
- Transposition table

References

- George T. Heineman, Gary Pollice, and Stanley Selkow (2008). "Chapter 7: Path Finding in AI". *Algorithms in a Nutshell*. Oreilly Media. pp. 217–223. ISBN 978-0-596-51624-6.
- Judea Pearl, *Heuristics*, Addison-Wesley, 1984
- John P. Fishburn (1984). "Appendix A: Some Optimizations of α-β Search". *Analysis of Speedup in Distributed Algorithms (revision of 1981 PhD thesis)*. UMI Research Press. pp. 107–111. ISBN 0-8357-1527-2.

1. Russell, Stuart J.; Norvig, Peter (2010). *Artificial Intelligence: A Modern Approach* (http://aima.cs.berkeley.edu/) (3rd ed.). Upper Saddle River, New Jersey: Pearson Education, Inc. p. 167. ISBN 0-13-604259-7
2. McCarthy, John (LaTeX2HTML 27 November 2006). "Human Level AI Is Harder Than It Seemed in 1955" (http://www-formal.stanford.edu/jmc/slides/wrong/wrong-sli/wrong-sli.html). Retrieved 2006-12-20. Check date values in: `|date=` (help)
3. Newell, Allen and Herbert A. Simon (March 1976). "Computer Science as Empirical Inquiry: Symbols and Search" (http://archive.computerhistory.org/projects/chess/related_materials/text/2-3.Computer_science_as_empirical_inquiry/2-3.Computer_science_as_empirical_inquiry.newell_simon.1975.ACM.062303007.pdf) (PDF). *Communications of the ACM* **19** (3). Retrieved 2006-12-21.
4. Edwards, D.J. and Hart, T.P. (4 December 1961 to 28 October 1963). "The Alpha–beta Heuristic (AIM-030)" (http://hdl.handle.net/1721.1/6098). Massachusetts Institute of Technology. Retrieved 2006-12-21. Check date values in: `|date=` (help)

5. Kotok, Alan (XHTML 3 December 2004). "MIT Artificial Intelligence Memo 41" (http://www.kotok.org/AI_Memo_41.html). Retrieved 2006-07-01. Check date values in: `|date=` (help)
6. Marsland, T.A. (http://www.cs.ualberta.ca/~tony/) (May 1987). "Computer Chess Methods (PDF) from Encyclopedia of Artificial Intelligence. S. Shapiro (editor)" (http://www.cs.ualberta.ca/~tony/OldPapers/encyc.mac.pdf) (PDF). J. Wiley & Sons. pp. 159–171. Retrieved 2006-12-21.
7. * Knuth, D. E., and Moore, R. W. (1975). "An Analysis of Alpha–Beta Pruning" (http://www.fileserve.com/file/ZgR5t3j/An_Analysis_of_Alpha-Beta_Pruning.pdf) (PDF). *Artificial Intelligence* **6** (4): 293–326. doi:10.1016/0004-3702(75)90019-3 (https://dx.doi.org/10.1016%2F0004-3702%2875%2990019-3).
    - Reprinted as Chapter 9 in Knuth, Donald E. (2000). *Selected Papers on Analysis of Algorithms* (http://www-cs-faculty.stanford.edu/~knuth/aa.html). Stanford, California: Center for the Study of Language and Information - CSLI Lecture Notes, no. 102. ISBN 1-57586-212-3. OCLC 222512366 (https://www.worldcat.org/oclc/222512366).
8. Abramson, Bruce (June 1989). "Control Strategies for Two-Player Games" (http://www.theinformationist.com/pdf/constrat.pdf/) (PDF). *ACM Computing Surveys* **21** (2): 137. doi:10.1145/66443.66444 (https://dx.doi.org/10.1145%2F66443.66444). Retrieved 2008-08-20.
9. Pearl, Judea (August 1982). "The Solution for the Branching Factor of the Alpha–beta Pruning Algorithm and its Optimality". *Communications of the ACM* **25** (8): 559–564. doi:10.1145/358589.358616 (https://dx.doi.org/10.1145%2F358589.358616).
10. Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (http://aima.cs.berkeley.edu/) (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2
11. Pearl, J., "SCOUT: A Simple Game-Searching Algorithm With Proven Optimal Properties," *Proceedings of the First Annual National Conference on Artificial Intelligence,* Stanford University, August 18–21, 1980, pp. 143-145.
12. Pearl, Judea; Korf, Richard (1987), "Search techniques", *Annual Review of Computer Science* **2**: 451–467, doi:10.1146/annurev.cs.02.060187.002315 (https://dx.doi.org/10.1146%2Fannurev.cs.02.060187.002315), "Like its A* counterpart for single-player games, SSS* is optimal in terms of the average number of nodes examined; but its superior pruning power is more than offset by the substantial storage space and bookkeeping required."

External links

- http://www.emunix.emich.edu/~evett/AI/AlphaBeta_movie/sld001.htm
- http://sern.ucalgary.ca/courses/CPSC/533/W99/presentations/L1_5B_McCullough_Melnyk/
- http://sern.ucalgary.ca/courses/CPSC/533/W99/presentations/L2_5B_Lima_Neitz/search.html
- http://www.maths.nott.ac.uk/personal/anw/G13GAM/alphabet.html
- http://chess.verhelst.org/search.html
- http://www.frayn.net/beowulf/index.html
- http://hal.inria.fr/docs/00/12/15/16/PDF/RR-6062.pdf
- Minimax (with or without alpha–beta pruning) algorithm visualization - game tree solving (Java Applet), for balance or off-balance trees (http://ksquared.de/gamevisual/launch.php?agent=2)
- Demonstration/animation of minimax game search algorithm with alpha–beta pruning (using html5, canvas, javascript, css) (http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html)
- Java implementation used in a Checkers Game (https://github.com/ykaragol/checkersmaster/blob/master/CheckersMaster/src/checkers/algorithm/AlphaBetaAlgorithm.java)

Search algorithms