Win, Lose, or Get Out the Way - Eliminating Unnecessary Evaluation in Game Search

Hsiu-Chin Lin and **Colleen van Lent**
California State University, Long Beach
Computer Engineering and Computer Science
1250 Bellflower Blvd, Long Beach, CA, 90840
{cvanlent,hlin6}@csulb.edu

Abstract

In this paper we present our approach of improving the traditional alpha-beta search process for strategic board games by modifying the method in two ways: 1) forgoing the evaluation of leaf nodes that are not terminal states and 2) employing a utility table that stores the utility for subsets of board configurations. In this paper we concentrate our efforts on the game of Connect Four. Our results have shown significant speed-up, as well as a framework that relaxes common agent assumptions in game search. In addition, it allows game designers to easily modify the agent's strategy by changing the goal from dominance to interaction.

Introduction

The most common approach to dealing with the time constraints of playing a human-opponent game is to use the mini-max algorithm with a cut-off limit and the alpha-beta pruning (Schaeffer & Plaat 1996), or as was successfully demonstrated by the Chinook checkers player (Schaeffer *et al.* 2007), a database of endgame positions to reduce the time needed to choose a move in real-time.

In this paper we propose a process of improving the alpha-beta pruning with a depth limit by modifying the traditional methods in two ways: 1) forgoing the evaluation of intermediate states and 2) employing a utility table that stores the utility for classified terminal states. This is done by using a two-tier evaluation approach that combines a "shallow" greedy evaluation to the immediate successor states with a "deep" evaluation at the depth cut-off level.

We concentrate our efforts on the game of Connect Four. Our results have shown significant speed-up while maintaining high-quality decisions, as well as a framework that allows our system to relax common game assumptions. In addition, it allows game designers to easily modify the agent's strategy by changing the goal from dominance to interaction.

Methodology

In our approach a two-tier evaluation is used. First, a "shallow" or greedy evaluation is performed on the immediate successors of the current state. Second, once the search agent encounters a terminal state or a terminal leaf, a "deep" evaluation is retrieved from a pre-specified value from a utility table. The evaluation function can be written as:

$$Evaluation(State) = Greedy(1^{st}move) + Utility(State)$$

As shown in Figure 1. Assume state A is one of the immediate successor states of the current state, and state B is a leaf node in the sub-tree of state A. The greedy evaluation of state A is calculated the first time state A is visited. When state B is visited, the utility of state B is retrieved from the utility table. The final evaluation of state B will be the sum of the greedy evaluation from state A and the utility of state B.

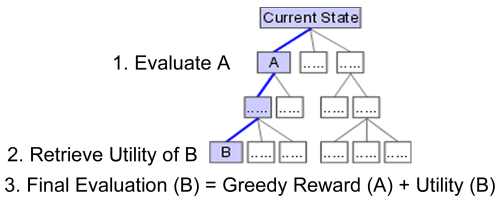

Figure 1: Game Tree Evaluation

Greedy Evaluation

From the current state, there is at most N immediate successor state in an N-column board. Each successor state will be examined, even if the state is not a leaf node. One can image this greedy evaluation is an "immediate reward" for each possible move from the current state.

The greedy evaluation procedure of this system is very similar to that of the human process. The basis is to examine the adjacent cells of the last move. We defined an array that stores the greedy evaluation scores before the search starts. Once an immediate successor state has been visited, a score is retrieved from the array and stored as a part of the current board state. It is important to note that the greedy evaluation scores are chosen in such a way to ensure that their value will not exceed the difference between the utilities of two dissimilar terminal states. Only states that are assigned the same utility in the next step are affected.

There are two reasons that the greedy evaluation is considered in the final evaluation function. First, if two paths lead to the same result, their utility will be the same, and the

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

greedy evaluation can serve as the second criteria. Second, considering the immediate reward is more realistic than relying on estimate the rewards of non-terminal states in the deeper levels.

Utility Table

The deep evaluation function does not attempt to apply a subjective heuristic function to the leaf nodes, but rather tracks the number of winning and losing states that have been encountered from the same path. The formula for calculating the utility is shown below, where ∞ is a large number not in excess of the limit of integer.

$$Utility(Win) = \infty \times \frac{\#wins}{(\#columns + 1)^{(depth+1)/2)}}$$

$$Utility(Loss) = -\infty \times \frac{\#losses}{(\#columns + 1)^{(depth+1)/2)}}$$

$$Utility(Draw) = Utility(Intermediate) = 0$$

This formula is designed in such a way that the utility of each winning state is greater than the utility of all winning states in more moves; and the utility of each losing state is less than the utility of all losing states in more moves.

Prior to the search starts, the program predefines various results in the search space and constructs a utility table to store the utility of each possible terminal state. The utility table is essentially a two-dimensional array, or an M by N matrix where M is the cut-off level and N is the number of columns + 1. Each row in the utility table represents the depth of the game tree. The utilities of terminal states after the agent makes its first move are stored in the first row, and the utilities of terminal states after the agent makes one move and the user makes another move are stored in the second row. The columns indicate the number of different terminal states from the same path. For instance, if there exists two ways to win from the same path in 5 moves, the utility is stored in to the fifth row (depth 5), third column. Below is the general form of the utility table.

Number of Terminal States from the Same Path							
d	0	1	2	3	4	...	N
1	Draw	1 way to win	2 ways to win	3 ways to win	4 ways to win		N way to win
2	Draw	1 way to lose	2 ways to lose	3 ways to lose	4 ways to lose		N way to lose
3	Draw	1 way to win	2 ways to win	3 ways to win	4 ways to win		N way to win
4	Draw	1 way to lose	2 ways to lose	3 ways to lose	4 ways to lose		N way to lose
...	...	...	...	...	...		...
M	Draw	1 way to lose	2 ways to lose	3 ways to lose	4 ways to lose		N way to lose

Figure 2: Utility Table

Experiment

In our experiment, we concentrate our efforts on the game of Connect Four. The standard board has been solved by James Allen and Victor Allis independently in 1988 with perfect play (Allen 1989; Allis 1988). However, the problem remains interesting because finding the optimal moves is too long in real-time.

We tested the search agent with an 8-by-8 board (while the methodology can be applied in various demensions) on a machine with an Intel Celeron CPU2.80 GHz. We performed two sets of trials; each containing 30 independent matches with 6 human players. The first trial had a depth-limit of 8 and the other had a depth-limit of 10.

In the trial with eight-step look ahead, the agent required roughly 20 milliseconds to select a move and won 24 games. In the second trial, the search agent's 10-step look ahead required 200 milliseconds on average. However, the second agent had 28 victories over 30 plays.

Conclusion

Efficiency and speed are essential in game search. We were able to reduce execution time while creating an option to modify the agent's behavior. Although this strategy is used in Connect-Four, similar methodology can be applied to various problems. Recent work by Hom and Marks has focused on balancing various challenges in games (2007). Our future work will entail focusing on novel evaluation criteria. Once the framework has been tested for proof-of-concept, machine learning techniques can be used to improve the gaming experience for individual users, similar to ideas of Yannakakis (2005) and Ponsen *et al.* (2007).

References

Allen, J. D. 1989. A note on the computer solution of connect-four. In *Heuristic Programming in Artificial Intelligence: The First Computer Olympiad*. Chichester, UK: Ellis Horwood. 134–135.

Allis, V. L. 1988. A knowledge-based approach of connect four: The game is over, white to move wins. Master's thesis, Vrije Universiteit, Amsterdam, the Netherlands.

Hom, V., and Marks, J. 2007. Automatic design of balanced board games. In *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference*, 25. Menlo Park, CA: AAAI Press.

Ponsen, M.; Spronch, P.; Munoz-Avila, H.; and Aha, D. W. 2007. Knowledge acquisition for adaptive game ai. *Science of Computer Programming* 67:59–75.

Schaeffer, J., and Plaat, A. 1996. New advances in alpha-beta searching. In *Proceedings of the 24th ACM Computer Science Conference*, 124–130. New York, NY: ACM.

Schaeffer, J.; Burch, N.; Bjrnsson, Y.; Kishimoto, A.; Mller, M.; Lake, R.; Lu, P.; and Sutphen, S. 2007. Checkers is solved. *Science* 317:1518–1522.

Yannakakis, G. N. 2005. *AI in Computer Games: Generating Interesting Interactive Opponents by the use of Evolutionary Computation*. Ph.D. Dissertation, University of Edinburgh, Edinburgh, U.K.