A Heuristic Search Algorithm for Empire-Based Games

CS 470 Project
Nathan Freeburg
December 11, 2002

Keywords: minimax, heuristic search, strategy games.

Abstract

Computer gaming has become one of the most prolific and challenging arenas for artificial intelligence applications. Further, in the gaming world, the empire-based strategy game presents some of the greatest challenges to AI programmers. The prohibitive size of the search space has curtailed the ability to use the techniques which have proven successful in classic board games such as chess, leaving the majority of games to use static rule-based systems. This paper details the use of aggressive pruning strategies and hierarchical organization to reduce the search space to a reasonable size. This approach results in a computer player which is capable of adaptively selecting moves based on the current state of the game. The resulting system can then react in a more intelligent manner to unusual situations presented to it by a human player. With these strategies, it is possible to reduce the search space to $n\,m^{d+1}$ separate states for a game with n pieces, m moves per piece, and a look ahead depth of d. This is a remarkable improvement over the $(m^n)^d$ states which would need to be examined for a full minimax search.

1.0 Introduction

The purpose of this project is to explore possibilities for improved competitive behavior in artificial intelligence programs designed to play empire-based strategy games. Game playing is often viewed as a proving ground for artificial intelligence techniques. While great strides have been made in games like chess, the techniques employed in more complex empire-based games are largely limited to static rule-based systems. Some of the methods currently used in empire-based games include finite-state machines and decision trees [4], multi-agent systems [3], and goal-directed reasoning [1].

One genre of computer games that presents many challenges for artificial intelligence is that of the empire-based strategy game. These types of games typically revolve around the conquest of the world and resource management. The world is usually represented by a hexagonal or grid-based map on which the various units and resources are placed. Game play can be conducted in a turn-based or real-time fashion. Some well known examples include Civilization, Warcraft, Age of Empires, and even the board game Risk. These are referred to as empire-style games after the Unix game Empire, which was one of the first games of this genre.

The limitations on computer systems playing empire-style games are due to the size of the search space. In chess, a player may only move one piece per turn, resulting in multiplicative behavior in the number of moves possible in a given turn. For instance, if a player has n pieces, each with m

moves available, then there are *mn* moves to be considered for that turn. The minimax tree for this game would then have a branching factor, *b*, of $b = mn$. To look ahead to a depth of *d*, a computer would need to consider $(mn)^d$ separate moves. Utilizing proper pruning techniques, this is a small enough number to allow a look ahead depth of up to 10 or 12 in a reasonable amount of time for a game like chess.

By contrast, an empire-based strategy game allows the player to move any or all pieces in a single turn. This results in exponential behavior in the number of moves possible in a given turn. For our same number of pieces, *n*, and same number of moves, *m*, this style of game requires that m^n moves be considered in a single turn. Thus, for a look ahead depth of *d*, the computer must consider $(m^n)^d$ separate moves.

As a result of these limitations, most commercial strategy games today utilize hard-coded rule-based techniques, such as finites state machines, to determine moves for the computer controlled players. While these techniques can be complex in and of themselves, they have the disadvantage of being static. They rely on patterns, which a human player can learn and exploit. Additionally, these rule-based systems are unable to adapt to the strategies a human player can devise, and must rely on greater access to resources than the human player possesses to remain challenging. Human players often perceive this as cheating, and it can be a source of frustration in many games.

2.0 Methodology

Since rule-based systems are subject to so many limitations, it is desirable to explore other, more adaptable techniques. One such technique, which has shown some promise, is hierarchical heuristic search [2]. At a high level, the design of this technique is similar to an abbreviated military command hierarchy (see figure 1). At the root of the command tree is the General, who determines the overall strategy. Decisions at this level can take the form of "all-out attack" or "defend bases." Below the General are the Commanders, who determine unit-level strategies for the units under their control. At this level, decisions can take the form of "attack enemy unit X with my unit A" or "advance unit A towards city 1." The primary goals at this level will be determined by the decisions made at the level above. At the lowest level, below the Commanders, are the Soldiers. This is the level at which the actions determined above are implemented.

2.1 Reducing the Search Space

From the analysis in section 1.0, it can be seen that full heuristic search for a strategy style game, even for small numbers of pieces and possible moves, results in a prohibitively large search space. In order to keep the time required to decide on a move at a reasonable level, the lookahead depth would have to be extremely small, perhaps no more than 2 or 3. This would result in little or no improvement over rule-based techniques at the cost of greater complexity and much higher processor usage. However, if techniques could be utilized to reduce this search space, a sufficiently deep lookahead might be achieved to justify the higher implementation cost.

2.2 Hierarchical Heuristic Search

The first search space reduction technique to examine is hierarchical organization [2]. This breaks the decision making process up vertically. High level strategy decisions can be made at the topmost level, giving smaller, mid-range objectives to the next level down. This topmost level can remain unconcerned with the movements of individual units, while the lower level can remain unconcerned with long-term goals. Since each move at the topmost level will likely consist of several moves for individual units, a deep lookahead is implicitly performed at this level.

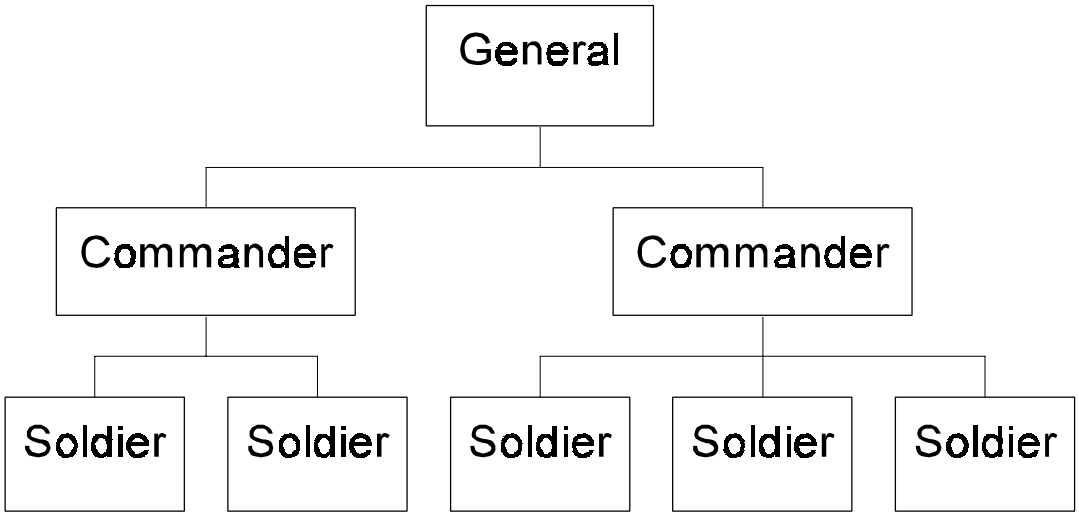

Figure 1: A command chain hierarchy.

2.3 Pruning Strategies and Algorithm Used

Further search space reduction can be attained at some small cost to the overall ability of the algorithm to approach the optimal sequence of moves. One technique, which has proven successful in reducing the search space to a reasonable level while retaining much of the power of heuristic search, is to:

- Choose the first unit to focus on
- Perform a search to a depth of one for all other pieces available
- Perform a minimax search to the full desired depth for the focus piece
- Repeat for all available pieces.

In order to retain some minimal group tactic behavior, we must account for the movements of all other pieces at each level of the minimax search. To accomplish this, a search to a depth of one

should be performed for all other pieces at each level of the minimax search. Essentially, this will select the single best move for that piece assuming that only that piece is moved. Better performance, as well as improved group tactic behavior, can be achieved by remembering the sequence of moves selected for each piece as minimax generates the search tree.

This algorithm, by restricting the minimax search to one piece at a time, reduces the branching factor to m, resulting in a search which looks at m^{d+1} states, rather than the $(m^n)^d$ required by a full minimax search. Since there are a total of n pieces, the given algorithm will need to look at a maximum of $n\,m^{d+1}$ states for a depth of d. While this is still a large number of states, it is much closer to the complexity of the search space for a game like chess or checkers, and is thus more practical for use in real applications.

3.0 Experiment

In order to test this algorithm, a simple strategy game was created. This game was based on *Empire Lite,* as described by Mock [2], which was in turn based on the Unix game *Empire*. This game is played on a 10x10 grid by two players. Valid moves are one square left, right, up, down, or nowhere, for a total of 5 moves. If a unit attacks an opposing unit with equal or higher strength than itself, both units lose one point of strength. If a unit attacks a unit with lower strength than itself, the attacking unit loses one point of strength and the attacked unit loses two points. If a unit's strength drops to zero or below, that unit is removed from the game. The only economic resource available is cities, which are initially unoccupied. Occupying a city raises the strength of all units on that player's side by one, while losing a city decreases the strength of all units on that player's side by one.

The hierarchy used in this algorithm employed only two levels. The strategy at the top level was to count the number of pieces surrounding points of interest, such as cities and the opponent's pieces. Once all such points had been examined, a target was selected based on the difference between number of friendly pieces and number of enemy pieces within a three square radius of each point. After the target was chosen, it was communicated to the second layer, which began the minimax search. The heuristic used for the second stage was a weighted polynomial that favored proximity to the chosen target, occupying cities, having greater total strength than the opponent, having more units than the opponent, being close to cities, being close to the opponent's units, and then reducing the opponent's total strength:

H = 20(Sum(10-Distance_Each_Unit_To_Target)) + 10(Num_A_Cities - Num_B_Cities)
+ 8(Total_A_Strength - Total_B_Strength) + 5(Num_A_Pieces - Num_B_Pieces)
+ (Sum(10 - Distance_Each_Unit_To_Closest_City))
+ (Sum(10 - Distance_Each_Unit_To_Closest_Enemy_Unit))
- (Total_Enemy_Strength)/2

This combination of strategies tended to favor swarming cities and enemy units with friendly units, though individual units were able to act independently to capture targets of opportunity.

3.1 Experimental Results

For a game with four pieces per side, a 400 Mhz processor was able to achieve a search depth of 8 in about five seconds for all four pieces. Though not yet a challenging opponent for a human player, the computer was able to select reasonable moves in a relatively short amount of time. Some moves that benefitted from heuristic search are shown in figure 2.

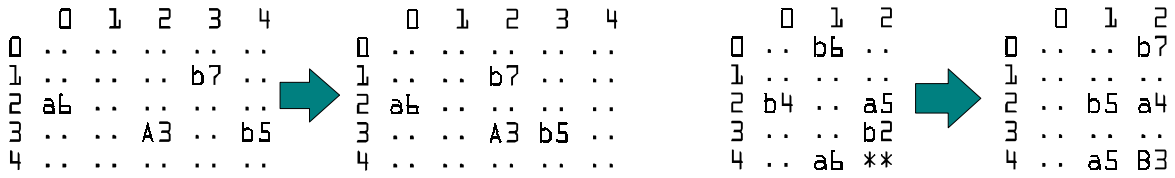

Figure 2: Example Moves

In the sequence at the left of figure 2, the piece represented by A3 is a city belonging to player A. The computer, player B, moves two of its units into position to take A's city and protect it from the piece a6. If acting without the benefit of lookahead, one of the units might very well pursue some alternate goal, leaving the city ripe for reconquest by player A. In the right-hand sequence, the **'s represent an unoccupied city belonging to A. Player B sends its low strength piece to conquer the city, increasing the strength of B's units and lowering that of A's. Simultaneously, B brings its higher strength pieces closer to the disputed area. If acting alone, B's low strength piece would have retreated to avoid being destroyed. Instead, it was moved into the path of harm to bring a greater benefit to the rest of B's pieces.

While the processor cycle usage is still too high to warrant use in mainstream games, the technique bears further consideration as processor speeds increase and multi-processor machines become more affordable. Perhaps in the future, a full production strategy game could use a derivative of this algorithm to create a challenging and adaptive computer player.

4.0 Future Work

There is still a great deal of work to be done to realize a computer opponent that can challenge a human player. To begin with, the units could be divided into separate groups, each of which could be viewed as being under a different commander. Each group could be considered to be a single superunit at the topmost level of the algorithm, and each could be given different orders. These divisions could be done geographically, or more interestingly by a technique such as influence mapping [5]. Additionally, the topmost level could be refined to use a more robust algorithm for high level decision making.

5.0 References

[1] Harmon, Vernon. (2002). "An Economic Approach to Goal-Directed Reasoning in an RTS." *AI Game Programming Wisdom.* Charles River Media, 2002.

[2] Mock, Kenrick. (2002). "Hierarchical Heuristic Search Techniques for Empire-Based Games." 2002 International Conference on Artificial Intelligence, June 24-27, 2002, Las Vegas, NV. Retrieved 11/25/2002 from http://www.math.uaa.alaska.edu/~afkjm/papers/empsearch.pdf

[3] Scott, Bob. (2002). "Architecting an RTS AI." *AI Game Programming Wisdom.* Charles River Media, 2002.

[4] Tozour, Paul. (2002). "The Evolution of Game AI." *AI Game Programming Wisdom.* Charles River Media, 2002.

[5] Woodcock, Steven (Ed.). (July 15, 1995). Influence Mapping thread. Retrieved 11/25/2002 from http://www.gameai.com/influ.thread.html