The paper "Game Tree Searching by Min/Max Approximation" describes a new method of searching game trees. In contrast to the existing depth first search method of searching a game tree using the minimax algorithm with alpha-beta pruning, this alternative method iteratively searches a game tree by approximating a given node's value along the way, and then dynamically decides which node in the game tree to expand.

Game trees are tree structures used to represent the different states that a game can take, based on the decisions that each player makes originating from the current move. Each node in the tree represents a different state of the game after a particular move has been made. The minimax algorithm is a method to traverse the game tree in order to find the best move to take in the game.

The minimax algorithm works by traversing the game tree in a depth first manner until either a terminal game state or a maximum tree depth has been found. At this point, the current node is given a value based on either the game state if it's a terminal node, or the result of a heuristic function for a non-terminal node. Parent nodes get their value by deciding the best move from their child nodes: the current player will try to maximize the value of their moves, while the opponent will try to minimize the value of the current player's moves. In this way, the value of each move is propagated up the tree until the current move has a value, and the best path through the tree has been mapped. Furthermore, the minimax algorithm can be optimized with alpha-beta pruning, by ignoring certain branches and nodes that will never be chosen by a player.

The new method described in the paper differs from the minimax algorithm in one key detail: whereas minimax traverses the entire tree in a depth first search manner, this new method iteratively expands the game tree from a chosen node. First, a heuristic function is used on each node with a path from the root to give that node a penalty value. For paths that contain multiple nodes, the penalty for the entire path is made from the sum of penalties of all nodes in that path. Next, the tip node of the path with the smallest penalty is then expanded, and the heuristic function is again run for each move to find the next node to expand. The tree is expanded in this manner until either the tree cannot be expanded anymore, or a termination condition (such as a time limit) has been met. In other words, at each step the iterative search method analyzes each path from the root node and expands the tip node that contains the smallest penalty.

The results from the paper found that the superior approach depended on the constraint that was being used. In cases where each search algorithm was constrained by CPU time, the minimax algorithm performed better. However, the iterative approach was found to produce superior results in cases where the algorithms were constrained by the number of times that they could call the heuristic evaluator function.