Abstract. Monte-Carlo Tree Search (MCTS) is a successful algorithm used in many state of the art game engines. We propose to improve a MCTS solver when a game has more than two outcomes. It is for example the case in games that can end in draw positions. In this case it improves significantly a MCTS solver to take into account bounds on the possible scores of a node in order to select the nodes to explore. We apply our algorithm to solving Seki in the game of Go and to Connect Four.

1 Introduction

Monte-Carlo Tree Search algorithms have been very successfully applied to the game of Go [7][11]. They have also been used in state of the art programs for General Game Playing [9], for games with incomplete information such as Phantom Go [3], or for puzzles [4][17][5].

MCTS has also been used with an evaluation function instead of random playouts, in games such as Amazons [15] and Lines of Action [18].

In Lines of Action, MCTS has been successfully combined with exact results in a MCTS solver [19]. We propose to further extend this combination to games that have more than two outcomes. Example of such a game is playing a Seki in the game of Go: the game can be either lost, won or draw (i.e. Seki). Improving MCTS for Seki and Semeai is important for Monte-Carlo Go since this is one of the main weaknesses of current Monte-Carlo Go programs. We also address the application of our algorithm to Connect Four that can also end in a draw.

The second section deals with the state of the art in MCTS solver, the third section details our algorithm that takes bounds into account in a MCTS solver, the fourth section explains why Seki and Semeai are difficult for Monte-Carlo Go programs, the fifth section gives experimental results.

2 Monte-Carlo Tree Search Solver

As the name suggests, MCTS builds a game tree in which each node is associated to a player, either Max or Min, and accordingly to values $Q_{\text{max}}$ and $Q_{\text{min}}$. As the tree grows and more information is available, $Q_{\text{max}}$ and $Q_{\text{min}}$ are updated. The node value
function is usually based on a combination of the mean of Monte Carlo playouts that went through the node \([7,13]\), and various heuristics such as All moves as first \([10]\), or move urgencies \([8,6]\). It can also involve an evaluation function as in \([15,18]\).

Monte-Carlo Tree Search is composed of four steps. First it descends a tree choosing at each node \(n\) the child of \(n\) maximizing the value for the player in \(n\). When it reaches a nodes with that has unexplored children, it adds a new leaf to the tree. Then the corresponding position is scored through the result of an evaluation function or a random playout. The score is backpropagated to the nodes that have been traversed during the descent of the tree.

MCTS is able to converge to the optimal play given infinite time, however it is not able to prove the value of a position if it is not associated to a solver. MCTS is not good at finding narrow lines of tactical play. The association to a solver enables MCTS to alleviate this weakness and to find some of them.

Combining exact values with MCTS has been addressed by Winands et al. in their MCTS solver \([19]\). Two special values can be assigned to nodes: \(+\infty\) and \(-\infty\). When a node is associated to a solved position (for example a terminal position) it is associated to \(+\infty\) for a won position and to \(-\infty\) for a lost position. When a max node has a won child, the node is solved and the node value is set to \(+\infty\). When a max node has all its children equal to \(-\infty\) it is lost and set to \(-\infty\). The descent of the tree is stopped as soon as a solved node is reached, in this case no simulation takes place and 1.0 is backpropagated for won positions, whereas -1.0 is backpropagated for lost ones.

Combining such a solver to MCTS improved a Lines Of Action (LOA) program, winning 65% of the time against the MCTS version without a solver. Winands et al. did not try to prove draws since draws are exceptional in LOA.

3 Integration of Score Bounds in MCTS

We assume the outcomes of the game belong to an interval \([\text{minscore}, \text{maxscore}]\) of \(\mathbb{R}\), the player Max is trying to maximize the outcome while the player Min is trying to minimize the outcome.

In the following we are supposing that the tree is a minimax tree. It can be a partial tree of a sequential perfect information deterministic zero-sum game in which each node is either a max-node when the player Max is to play in the associated position or a min-node otherwise. Note that we do not require the child of a max-node to be a min-node, so a step-based approach to MCTS (for instance in Arimaa \([14]\)) is possible. It can also be a partial tree of a perfect information deterministic one player puzzle. In this latter case, each node is a max-node and Max is the only player considered.

We assume that there are legal moves in a game position if and only if the game position is non terminal. Nodes corresponding to terminal game positions are called terminal nodes. Other nodes are called internal nodes.

Our algorithm adds score bounds to nodes in the MCTS tree. It needs slight modifications of the backpropagation and descent steps. We first define the bounds that we consider and express a few desired properties. Then we show how bounds can be initially set and then incrementally adapted as the available information grows. We then show how such knowledge can be used to safely prune nodes and subtrees and how the bounds can be used to heuristically bias the descent of the tree.