Monte Carlo Tree Search

Monte Carlo tree search allows Go playing programs to evaluate positions without the need for a static board evaluation function. MCTS builds a tree where the root is the current board position and each child node represents a potential move.

Games are randomly simulated from the leaves of the tree. After the simulation is completed the winner is calculated. Using this information the win and run count is updated in the tree. This allows us to decide whether to run more simulations or not. When more simulations are needed, we do so, otherwise the program prints its results.

Monte Carlo Tree Search

• Idle Time:
• Wasted Computation:
• Communication:

Parallelization can often induce many slowdowns into a given program, called parallelization overhead.

The key part of this method is that each processor works on a different part of the tree, but the difficult part is that this method requires multiple processors accessing the tree at the same time.

In the client-server model we have one computer, called the client, which is commanded by a user to solve a specific problem, and n other computers, called servers, which wait to receive information from the client. The client divides the problem into small parts that can be computed independently of each other. For each part, it sends a message to one or more servers giving it the information it needs to compute that subproblem. Each server receives a message and begins to work on the problem. When the problem is finished, the server replies back to the client with the solution and is ready to receive another subproblem. In this way the client is able to combine the results of the subproblems to solve the larger problem.

Some previous work has focused on parallelizing MCTS on a shared-memory, also known as a multi-core, computer. This means that two programs can be running separately on the same computer, as each uses a single core. We call these programs “threads”. Even though a computer has multiple cores, it usually has a single, unified RAM. This means that each thread needs to share the same block of memory. The good news is that this makes communication between threads very easy, as one thread can just change a memory location and the other can see it changed. The bad news is that this severely limits the number of processors one can use.

In our implementation, one thread runs on each server, whereas the servers handle all the steps of the MCTS algorithm. Each server performs MCTS independently, creating its own tree, and sending back results when time is almost up.

With root parallelization the client is rather simple, and the servers are more complex. The client only requests the scores for each of the next possible moves and aggregates this data, whereas the servers handle all the steps of the MCTS algorithm. Each server performs MCTS independently, creating its own tree, and sending back results when time is almost up.

With leaf parallelization, the servers are simple, while the client is more complex. The client must maintain the tree and perform the selection, expansion, and backpropagation steps of the MCTS algorithm. The server only takes the node selected by the client, simulates random games, and reports the winning color. The client then incorporates the information into the tree. In the basic version of leaf parallelization the client waits for all the servers to respond before it continues with the MCTS algorithm and selects a new node on which to run.

Further Information

Acknowledgements

References
