Parallelizing Monte-Carlo Tree Search for Dots and Boxes

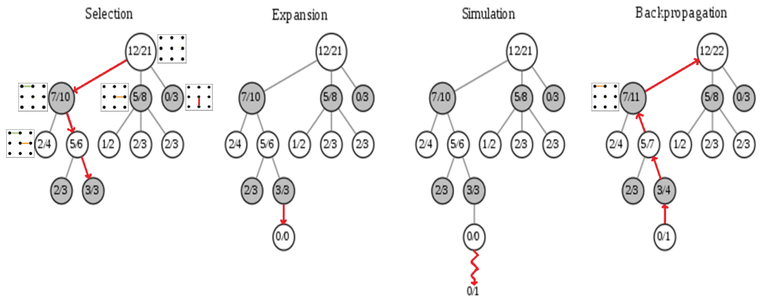
Pranay Agrawal and Uta Ziegler, Western Kentucky University

Abstract

The Monte-Carlo tree search (MCTS) is a method designed to learn how to solve problems. MCTS performs random simulations from the current situation and stores the results in order to distinguish decisions based on their past. After the simulations, the MCTS algorithm selects the best decision and then repeats the process until a stopping state. Parallelizing the MCTS means to divide the learning process among independent learners. After a fixed number of simulations, some learned data is shared and combined. Past research has shown that this approach is faster than non-parallelized approaches. It seems that the time reduction from dividing the learning outweighs the potential costs from redundant learning. This project focuses on the effect of the level of various controlled resources on the learned performance of MCTS. Specifically, we explored how the performance of the game Dots and Boxes learned through a parallelized MCTS approach is effected by (**i**) the number of simulations for every situation, (**ii**) the number of independent learners, (**iii**) the amount of information shared, and (**iv**) the frequency of sharing. A problem with symmetric board positions is presented along with details of the MCTS algorithm. Non-parallelization results are also discussed.

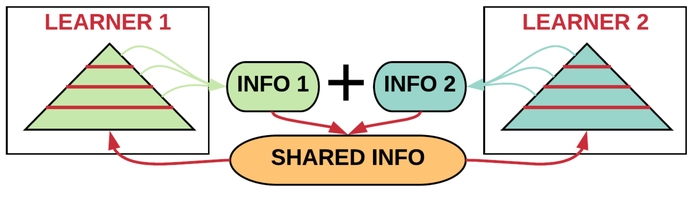
Introduction

- Four steps of MCTS: selection, expansion, simulation, backpropagation [1].



- The rules of the game Dots and Boxes: two players take turns drawing a line on a square grid of dots. If a player draws the 4th line of a square the player collects a point and completes another move again. The player with the most points by the end wins the game.
- Parallelization: sharing learned data involves each learner sending the tree information from one or more levels to a controller that will appropriately combine the information and distribute it back [2] (see figure below).

Parallelization of Two Learners



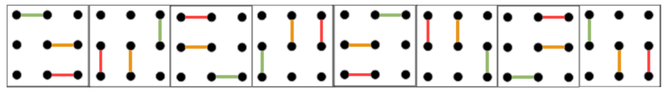
Methods

- Use self-play (two players who learn using MCTS playing against each other) to assess performance
- Measure performance by $\frac{\#\ of\ wins}{total\ matches}$
- Establish a base-line performance of one (non-parallel) player for various size boards against a MCTS player of 5000 simulations per situation
- Develop algorithm based on [3] to incorporate combined data seamlessly into each learner's tree
- Measure performance of parallelized MCTS: vary the number of simulations for every situation but keep constant all other parameters
- ** Update algorithm to account for symmetry of board positions (current)
- Analyze performance relative to varied resource from base-line

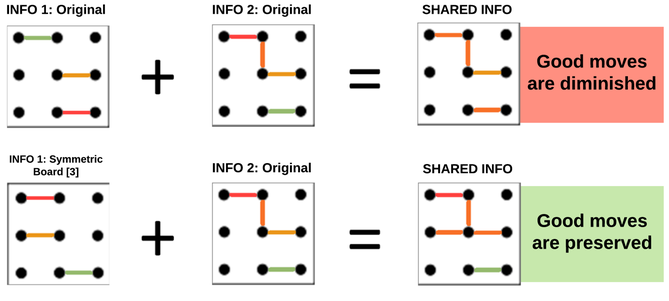
Results

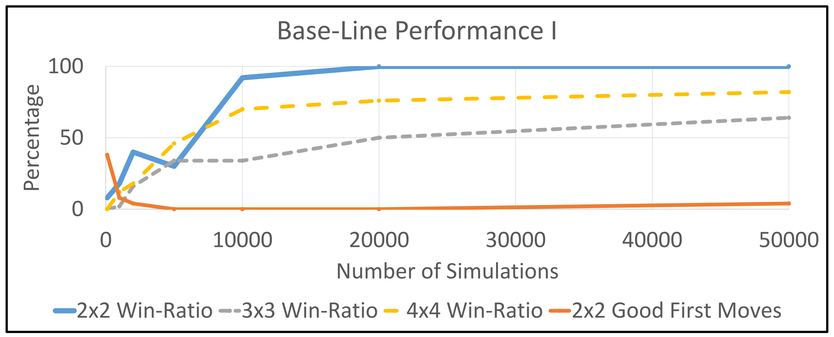
- In the Base-Line Performance graphs, as the number of simulations increases, the win-ratio increases for the single non-parallel player
- The MCTS performance does not decrease evenly with increasing board sizes (3x3 < 4x4 < 2x2 for simulations > 5000)
- The percentage of good first moves for a 2x2 board at first decreases and then increases between 1000 and 1000 simulations
- Parallelization did not behave as expected (potential symmetry issues):
  - Different board positions are symmetric to each other
  - The basic implementation of MCTS learns independently of the others
  - Combining learned information without accounting for symmetric states leads to counter-intuitive results (see figures below)

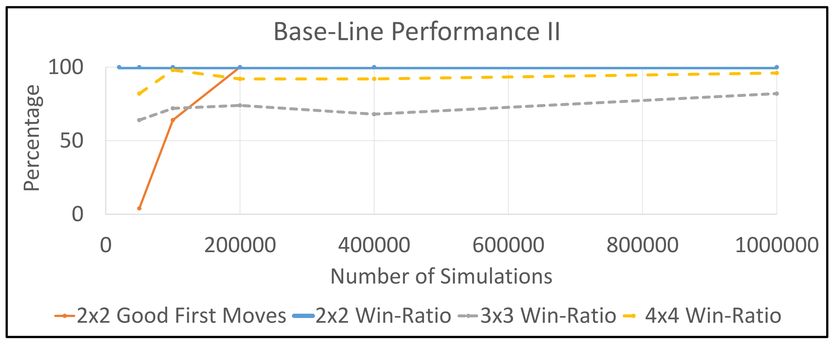
Different but Symmetric Board Positions



Exploiting Symmetry During Combination







Conclusion/Discussion

- Parallelism is predicated on the idea of consistently increasing performance with more simulations
- Different learners learn different aspects of the overall task with some repeated information, and combining it is similar to one learner learning the different aspects using more simulations
- A decrease in performance with increased simulation indicates that the selection of the number of simulations for the current situation for each learner must be made with care
- An analysis of the time requirement for the sharing and combining is needed
- An efficient algorithm to deal with symmetry must be discussed

Acknowledgments

We would like to thank Jared Prince for the development of the base Monte Carlo Tree Search code and Brian Zhu for the root parallelization allowing us to adapt it for multi-level parallelization. We also would like to thank Western Kentucky University for allowing us access to their high performance computing cluster.

References

[1] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, *A Survey of Monte Carlo Tree Search Methods* in IEEE Transactions on Computational Intelligence and AI in Games, Vol. 4, No. 1, March 2012.

[2] A. Bourki, G. Chaslot, M. Coulm, V. Danjean, H. Doghmen, et al.. *Scalability and Parallelization of Monte-Carlo Tree Search*. The International Conference on Computers and Games 2010, Kanazawa, Japan. 2010.

[3] B. E. Childs, J. H. Brodeur, and L. Kocsis, *Transpositions and move groups in Monte Carlo tree search*, *2008 IEEE Symposium On Computational Intelligence and Games*, Perth, WA, 2008, pp. 389-395.