Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Distribution of UCT and Its Ramifications

Marc Chee
The University of New South Wales
Sydney, Australia
marcchee@cse.unsw.edu.au

Abstract

My thesis is largely focused on the parallelisation of UCT (and other Best-First Search techniques) and the ramifications of doing so. I have identified issues with chunking in UCT, created by some forms of parallelisation, and developed a solution to this involving buffering of simulations that appear "out of order" and reevaluation of propagation data. I have developed a technique for scalable distribution of both tree data and computation across a large scale compute cluster. The context of most of my work is General Game Playing, but the techniques themselves are largely agnostic to domain.

1 Introduction

The UCT (Upper Confidence Bound for Trees) algorithm is a prevalent instance of Monte Carlo Tree Search. UCT draws inspiration from the work of the mathematical community on Multi-Armed Bandit problems and revolves around finding a balance between exploration and exploitation.

UCT iterates using four phases: Selection, Expansion, Simulation and Backpropagation. Each iteration, the tree is traversed from the root to a leaf which is *selected* in a way that balances favouring promising subtrees (exploitation) against exploring parts of the tree with less information. The tree is then *expanded* by a single node at the selected leaf. A Monte Carlo *simulation* plays to the end of the game so as to get some initial information of the newly created node. This new piece of information is aggregated to the information already present in ancestor nodes as it is *backpropagated* to the root.

The tree grows after each iteration and the information associated to its nodes can be shown to converge to their game theoretic value over time. UCT has also proved to be very successful in practice for numerous game domains [Browne *et al.*, 2012].

UCT converges over time based on the number of completed iterations. It makes sense that parallelisation of UCT should yield convergence in less real time than a serial algorithm. However, the nature of UCT is inherently serial, requiring information from previous iterations to be able to direct future ones. Some efforts have been made to parallelise the algorithm [Cazenave and Jouandeau, 2007], but there is still the potential for the behaviour of parallelised UCT to stray from the mathematical ideal of a standard UCT search. The main focus of my research is to create a framework that will be able to utilise the extra computing power of parallelisation while not sacrificing the benefits of a mathematically correct UCT search.

I first present my *buffering and reevaluation* techniques that allow us to alleviate the issues inherent in creating chunks of simulation data outside of the normal order of UCT. I then present my *distributed framework* that distributes UCT (or any other kind of Best-First Search) across a compute cluster without the need for shared memory and with minimal communications overhead.

2 Chunking and Buffering

I've identified the issue of chunking, which can be caused by modifications on UCT such as leaf parallelisation[Cazenave and Jouandeau, 2007] and the use of transposition tables[Romein *et al.*, 1999]. In these situations, it is possible for a "chunk" of more than one simulation to be created or discovered at a leaf of the tree. Since this chunk is not created with UCT's usual balance of exploitation/exploration, it could mislead its parent node to weigh too heavily in favour of that particular node.

I developed the technique of *buffering* to try to alleviate the negative effects of chunked simulation data without sacrificing the extra information in the chunk. Buffering basically delays the propagation of chunked simulations. When a chunk of data (more than one simulation at a time) is received, we do not immediately update our current node and continue propagation. Instead, we store the data in a buffer at the receiving node. During backpropagation a parent node and its children are considered as a group. The parent loops, using UCT selection to select a child and propagate a single simulation (using the average winrate of that child's buffered simulations) from that child into the parent's buffer. This loop continues until it reaches a child that has an empty buffer, potentially leaving other buffers unused, but importantly propagating from the children in a ratio according to a UCT search.

Continuing work on the buffering system showed that this technique still had possibly negative ramifications on the convergence of a uct search, so I developed the idea of *buffering with reevaluation*. The first step was to realise that the buffering technique would take a significant amount of propagations to balance weighting between two different chunks at

the same node. Reevaluation removes all prior propagations from a node if a new chunk arrives and then recalculates the node's statistics based on the average of all simulations that have been propagated to the node and its current UCT value. It then carries out a buffered propagation.

Reevaluation also had a potential issue in that it could put a node out of balance with the other nodes in its generation. The next step was to develop *multi-node reevaluation* which has brought the buffering technique to the point where it will reevlauate a generation of nodes, using whatever buffers are stored in each of them. The result of the multi-node reevaluation is a generation of nodes and their parent who end up balanced as if all simulations were created and propagated using a normal UCT search and in addition, the winrates in the simulations benefit from all the simulations in the chunks at each node. In an IJCAI submission [Chee *et al.*, 2015] we show experimentally that this technique can significantly alleviate the issues inherent in chunked UCT simulations, accelerating convergence over naive methods. The paper also shows that buffering is a principled technique that can return a tree with chunked data to the correct the mathematical balance of exploration and exploration.

3 A Framework for Distributed Best-First Search

This framework distributes a UCT search across multiple processors without the need for a shared memory space. It seeks to leverage the fact that UCT converges based on the number of iterations, and so the more processing power can be utilised, the less real time is necessary. This means that it is capable of being run on a compute cluster such as the Leonardi UNSW Engineering HPC cluster. The framework consists of a master process that is responsible for organising the cluster as well as reporting results and an arbitrary number of worker processes. Each worker process is uniquely addressed and will be storing and processing one or more subtrees of the UCT tree.

One worker is selected by the master as the "root worker" and is assigned the root of the UCT tree and builds a sub-tree over a set amount of time or number of simulations. From then on, the master will send "requests for work" to the root worker, who will begin a UCT selection descent of their own tree. When they reach a leaf of their tree, they will either assign that leaf to another worker or, if it is already assigned, pass on the request. Any worker receiving a request will perform a selection descent of the corresponding sub-tree in the same way. Once a worker selects an unassigned leaf, it will send a "work order" to the worker that the request was sent on behalf of. This worker will be given the leaf of the tree as the root of a new sub-tree, which it will then build for a set amount of time or simulations. Once the sub-tree is completed, the worker will send a propagation message back to the worker that sent it the work order (its parent in terms of the current sub-tree). The propagation message will contain all the statistics that have been gathered at the root node of the sub-tree and will then be propagated through other workers sub-trees exactly as in a normal UCT until eventually reaching the global root.

In this way, a UCT tree is built that is distributed across the processors. This allows us to distribute both processing (building and descending sub-trees) as well as memory usage (storing sub-trees) across multiple processors and their memory. Communication overheads are reasonably low as the messages being passed between workers never need to contain actual tree information, only identifiers or statistics (for UCT the statistics are numbers of simulations and scores). The framework has been scaled up to 2000 CPUs in testing.

It does, however have one disadvantage and that is the issue of chunking. When a worker propagates all of the statistics of its sub-tree to another worker, that information arrives in a chunk at a single leaf node. The chunk may cause a single leaf node to have unrealistically high weighting in the statistics that are propagated. Utilising the buffering and reevaluation techniques, I hope to alleviate the negative effects of chunking in this framework.

4 Current and Future Work

Both techniques in this abstract have been developed to the point of proof of concept and reliability but have yet to be integrated into one system together. It will be interesting to see if what is theoretically a weakness of the cluster player framework is in practice alleviated by the introduction of buffering.

A prototype of the distributed framework is incorporated into Quorum Player, UNSW's General Game Player and has taken part in the International GGP Competition in 2014.

On top of this, the buffering technique has been shown to be effective in a few particular scenarios. I am currently testing it in General Game Playing conditions under several different domains to see whether the effect is as readily noticeable in "normal" game conditions as it is in situations that are theoretically more susceptible to the chunking issue.

Another potential avenue for future work is to apply my buffering technique to best-first searches using transposition tables.

References

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[Cazenave and Jouandeau, 2007] Tristan Cazenave and Nicolas Jouandeau. On the parallelization of uct. In *Proceedings of the Computer Games Workshop*, pages 93–101, 2007.

[Chee *et al.*, 2015] Marc Chee, Abdallah Saffidine, and Michael Thielscher. A principled solution to the problem of chunking in uct. In *IJCAI*, 2015. Submitted.

[Romein *et al.*, 1999] John W Romein, Aske Plaat, Henri E Bal, and Jonathan Schaeffer. Transposition table driven work scheduling in distributed search. In *AAAI/IAAI*, pages 725–731, 1999.