Towards MCTS for Creative Domains

Cameron Browne
Computational Creativity Group
Imperial College London
180 Queens Gate, SW7 2RH, UK
camb@doc.ic.ac.uk

Abstract

Monte Carlo Tree Search (MCTS) has recently demonstrated considerable success for computer Go and other difficult AI problems. We present a general MCTS model that extends its application from searching for optimal actions in games and combinatorial optimisation tasks to the search for optimal sequences and embedded subtrees. The primary application of this extended MCTS model will be for creative domains, as it maps naturally to a range of procedural content generation tasks for which Markovian or evolutionary approaches would typically be used.

Introduction

Ludi is a system for automatically generating and evaluating board games modelled as rule trees (Browne, 2008). New artefacts are created by evolving existing rule trees and measuring the results for quality through self-play. Although this process proved successful by creating a game of notable quality that is now commercially published (Andres, 2009), it also highlighted some problems with the evolutionary approach for game design:

Wastage: Thousands of bad games were generated for every good one.
Focus: Creativity only became evident when introns (flawed rules) were allowed to proliferate and breed.
Bias: The choice of initial population biased the output, and if not themselves well-formed would not likely produce any playable children at all.

Due to the random nature of crossover and mutation, there is no guarantee that the evolutionary process will converge to an optimal result. Might there be a better way?

Monte Carlo Tree Search (MCTS) has revolutionised computer Go and is now a cornerstone of the strongest AI players (Coulomb 2006). It works by running large numbers of random simulations and systematically building a search tree from the results. It has produced world champion AI players for Go, Hex, General Game Playing, and unofficial world champions for a number of other games.

An attractive feature of MCTS is its generality. It can be applied to almost any domain that can be phrased in terms of states and actions that apply to those states, and has been applied to optimisation tasks other than move planning in games, such as workforce scheduling, power grid control, economic modelling, and so on. MCTS is also:

Aheuristic: No heuristic domain knowledge is required.
Asymmetric: The search adapts to fit the search space.
Convergent: The search converges to optimal solutions.

MCTS systematically explores a given search space by preferring high-reward choices while guaranteeing the (eventual) exploration of low-reward options, and only requires a fitness function for completed artefacts to operate. This makes it an attractive proposition for procedural content generation in creative domains; however, such problems tend to be more complex than simple {*state*, *action*} pairs. They are typically modelled as sequences, grammars, rule systems, expression trees, and so on, which are outside the scope of the standard MCTS algorithm.

We propose a generalisation of the MCTS algorithm and its extension from the search for optimal actions to the search for optimal sequences and subtrees. This should have direct applicability to procedural content generation in game design and other creative domains, where it might augment or even provide an alternative to existing methods for creating new high quality artefacts.

MCTS

Figure 1, from Chaslot et al (2006), shows the four basic steps of the MCTS algorithm. Each node represents a state *s* and each edge represents an action *a* that leads to an updated state *s'*. Each node maintains a record of its estimated value, number of visits, and a list of child actions.

The algorithm repeats the following process: starting at the root node *R*, descend through the tree (choosing the optimal action with each step) until a leaf node *L* is reached. Then, expand the tree by adding a new node *N*, complete the game by random simulation, and backpropagate the result up the list of selected nodes.

UCB The key to the algorithm's success lies in the method it uses to select optimal actions from among lists of those available during tree descent. A variation of the Upper Confidence Bounds (UCB) method (Auer et al, 2002) is typically used to select the node that maximises:

$$\overline{X}_i + \sqrt{\frac{2\ log\ n}{n_i}}$$

where X_i is the estimated (mean) value of child i, n_i is the number of times child i has been visited, and n is the number of times the node itself has been visited.

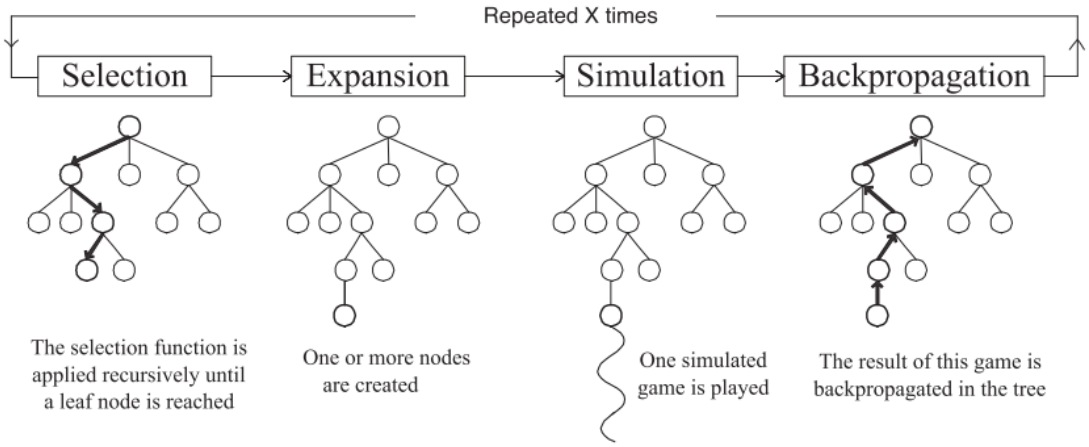

Figure 1. The four basic steps of the MCTS algorithm.

UCB provides a good balance between the *exploitation* of estimated node values and the *exploration* of the search space, so that even low-value nodes are occasionally exercised to increase the reliability of their value estimates. Kocsis and Szepesvari first proposed the use of UCB in an MCTS setting with their UCT ("UCB applied to Trees") method in 2006, and this is the specific embodiment of MCTS used in most current applications.

General MCTS Model

The standard application of MCTS is to find optimal moves in zero-sum, two-player games with alternating moves and fixed turn order, such as Go (Gelly et al, 2006). While it has been applied to more general problems, such as general game playing (Bjornsson and Finnsson, 2009) and some combinatorial optimisation problems, the algorithm is specifically adapted for such different domains. We now consider ways to simplify some underlying assumptions to generalise the algorithm and more cleanly separate it from its given application domain.

Zero-Sum MCTS is often used to model zero-sum games with a win given a discrete value of +1 and a loss -1 (draws are worth 0). We relax this assumption so that the algorithm works with continuous simulation results in the range [-1..1], where the extremes represent ideals that may never actually be realised. This generalises to domains in which individuals are measured by a fitness function rather than discrete win/lose/draw classifications.

Two Players The algorithm often models two *adversarial* opponents, i.e. players competing for opposing rewards, which has implications for the backpropagation stage. In traditional game search terms, the instigator of the search (MAX) will try to maximise their reward while the opponent (MIN) will try to minimise this reward, resulting in a minimax tree. If a simulated playout yields a result of +1 for the current player, then the value backpropagated through the tree will be negated with each search ply: +1, -1, +1, -1, etc.

Multiplayer games (i.e. those with more than two players) can be modelled using a *paranoid* approach in which each player simply assumes that all other players are acting against them. This removes complicating aspects of coalitions and effectively reduces *N*-player games to a 2-player model, but can give good results (Sturtevant, 2002). Playout results for say three players using the paranoid model would be negated in cycles of three during backpropagation: +1, -1, -1, +1, -1, -1, etc.

Single player games, e.g. solitaire puzzles, are *cooperative* as there is only one player who will generally not seek to sabotage their own moves. Such games may return boolean results indicating success (puzzle solved) and failure (dead end), or continuous reward functions that indicate distance from a desired perfect solution. The puzzle environment may respond to player moves; if these responses are *deterministic* then they are simply part of the state update following each action, otherwise if the environment's responses are intelligent and adversarial then the puzzle is actually a 2-player game.

Move Order Not all games have alternating moves; some have variable play order or composite multi-part moves. Consider a hypothetical Go variant in which the player who surrounds an enemy group need not remove all surrounded pieces but may elect which, if any, to remove. If a group of say 20 pieces is surrounded, the mover has $2^{20} = 1{,}048{,}576$ possible ways to remove a subset of 0 to 20 pieces.

It would be ridiculous to list all of these choices for a given node, so instead a more practical option is to treat these optional removals as a *multipart move*, i.e. a variable length sequence of single piece removals. This approach may be strategically dubious as each sub-move is considered in isolation rather than part of the greater move, but it is the only practical solution in many cases and usually proves sufficient (Schmidt, 2010). Such multipart moves are another consideration that must be taken into account during the backpropagation stage, as simulation results must then be negated across variable ply numbers to reward/punish the players correctly.

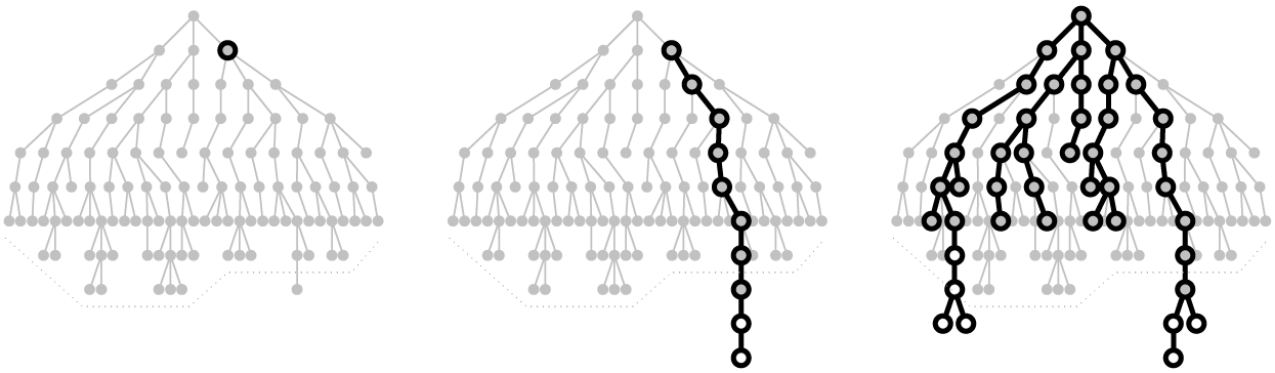

Figure 2. Example solutions for the three search types: action, sequence and subtree.

Generalising MCTS The solution to the limitations described above is *relative node ownership*. Each node in the search tree is assigned an owner – typically the player to move – and is updated during the backpropagation step according to the simulation result relative to its owner. Instead of returning a single value indicating the result of each simulation, the domain produces a vector of values indicating the result relative to each player. This removes underlying assumptions regarding player number, move order, move length and distinctions between adversarial versus cooperative modes of play, allowing a more general MCTS model that paves the way for the following extensions.

Extended MCTS Model

We now extend the general MCTS model from searching for optimal actions to searching for optimal sequences and embedded subtrees, such as those typically used in procedural content generation and computational creativity tasks.

MCTS Sequence Search

Figure 2 (left) shows the result of a standard MCTS search, which is the highest-valued root child action. As the basic operation of the algorithm is to complete a sequence of actions each iteration, it is straightforward to make these sequences the target of the search (Figure 2, middle).

This can be achieved simply by keeping at all times a record of the best sequence so far including any random playouts (a pointer to the sequence's tail is sufficient), and using as the reward value for each sequence an estimate of its fitness. For solitaire puzzles this fitness value may involve distance to solution, or for more creative applications such as music generation, the fitness function may involve aesthetic measurement of passages of notes.

As per standard MCTS, a sequence is run to completion per iteration, and its value backpropagated through the selected nodes. Each sequence may be completed within the search tree or may cross the tree boundary (as shown in Figure 4), hence it is possible that the best sequence could be at least partially randomly completed. As with action search, the root node is not part of the completed sequence but merely defines the list of possible starting points for the search.

MCTS Subtree Search

The second extension of the algorithm – from sequence search to subtree search – is complicated by the *polyadic* (multi-argument) nature of the problem. Rather than each state s having a single action a applied, each state may now have N actions or arguments that are simultaneously applied. For example, the search target may be an expression tree with nodes that contain multiple arguments.

These search target subtrees should not be confused with the MCTS search tree itself; we distinguish between the *search tree* and the *solution subtrees* that are embedded within it. The search tree represents the possible solution space while solution subtrees represent actual realisations of those possibilities. As per the standard MCTS approach, a subtree is completed with each iteration, evaluated, and its value backpropagated through the selected nodes. The subtree may be entirely embedded within the search tree or it may cross the search tree boundary at one or more points, in which case each open branch is randomly simulated to completion (Figure 2, right). A record of the best subtree must be kept at all times; a list of its leaf nodes is sufficient to reconstruct the subtree. It is possible that one or more branches of the best subtree could be randomly completed.

Polyadic Strategies

In order to successfully extend the MCTS method to subtree search, we propose a number of strategies for handling the selection of arguments for polyadic nodes. Dependencies between such arguments – and indeed between nodes and even subtrees – become important in this context.

For example, Figures 3 and 4 show two hypothetical games described as rule trees, such as those that might be generated by the Ludi system (Browne, 2008). In "Kill the Knights" players take turns moving one of their pieces in a knight move and win by capturing three enemy pieces. In "Pin the Knights" pieces instead pin enemy pieces that they land upon and a player wins by forming a stack three high.

While neither game is a masterpiece, "Pin the Knights" may be the more interesting of the two. There is some tension between the benefit of pinning enemy pieces against the danger of providing height 2 stacks that the opponent might exploit to win the game.

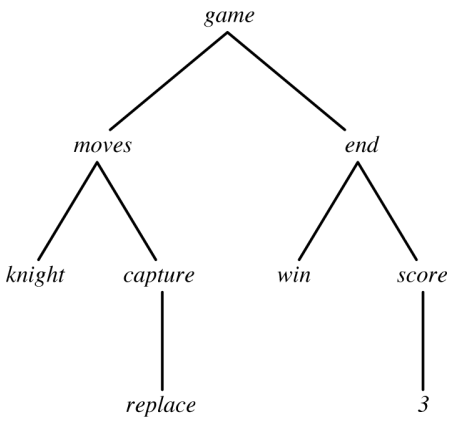

Figure 3. Rule set for "Kill the Knights".

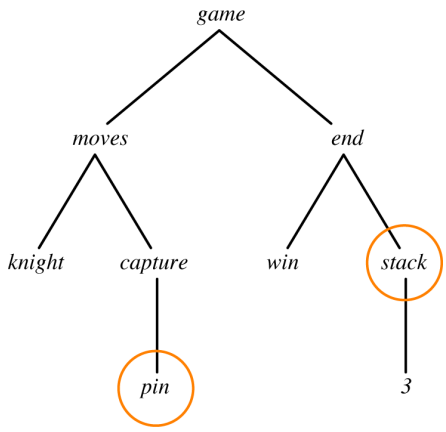

Figure 4. Rule set for "Pin the Knights".

The rule differences between the two games (replace/pin and score/stack) are dependent as changing either one in isolation will break the game by making it unwinnable; both changes must occur simultaneously for the modified game to work. In terms of these two rules, each game is in a local maximum that can only be escaped by modifying both rules simultaneously.

This example demonstrates that superior results can be achieved if related degrees of freedom in the content can be identified and adjusted in tandem. It is not guaranteed that an evolutionary method would ever perform such dependent rule changes simultaneously, whereas MCTS performs a more systematic exploration of the search space due to its well-balanced exploration component and mechanisms may be added for detecting and exploiting such dependencies. We distinguish between *independent* and *dependent* node selection in subtree search, and propose polyadic strategies for each case in the following sections.

Independent Node Selection

The simplest strategies for optimally completing polyadic subtrees during MCTS search ignore node dependencies, so node choices made in one part of the tree will not affect node choices in other parts of the tree during tree descent or backpropagation. We now present two such strategies.

Direct Choice The first case to consider is the most obvious; when presented with a polyadic node, simply choose for each argument the action selected from its available choices by UCB. Each selection is made independently of the other arguments and all other nodes in the tree.

For example, Figure 5 shows a polyadic node p_i with three arguments (branches) to be populated with actions. In the direct method, the action for argument a will be selected from the set $\{a_1, a_2, ..., a_n\}$, the action for argument b will be selected from the set $\{b_1, b_2, ..., b_n\}$, and the action for argument c will be selected from the set $\{c_1, c_2, ..., c_n\}$.

For each iteration of the search, a subtree is completed in this manner, measured for fitness, and the result backpropagated through the selected subtree's nodes. The result of the search will be the completed subtree with the highest reward value.

Embroyonic Development Another approach inspired by genetic programming methods for circuit design (Koza et al, 2001) is to maintain a single "embryonic" individual that is modified over the course of the search.

An embryonic tree is created, then for each iteration a non-branching sequence is followed through it, simulated to completion and the result backpropagated through the sequence, while unvisited nodes outside the active sequence remain frozen for that iteration.

Figure 6 summarises the process for a given polyadic node p_e visited along the active sequence during tree descent. One argument is chosen from those available, in this case b, which will be the best action of the *worst* performing argument. The result of the search will be a copy of the embryonic tree taken on the iteration at which it achieved its highest estimated value.

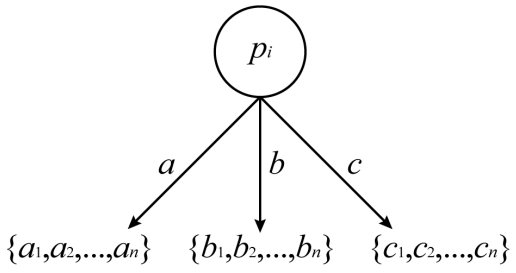

Figure 5. Direct approach to argument completion.

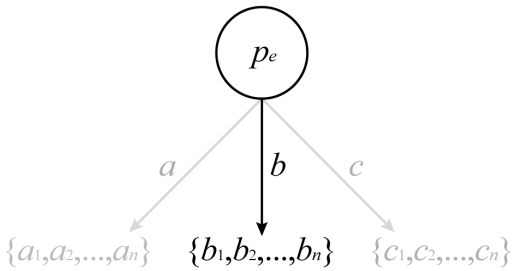

Figure 6. Embryonic approach to argument completion.

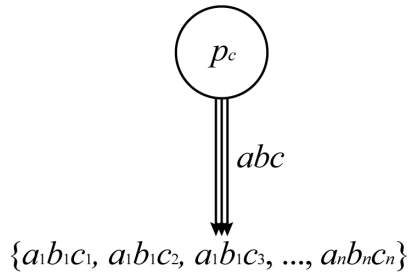

Figure 7. Compound approach to argument completion.

Dependent Node Selection

The following approaches for completing polyadic subtrees preserve node dependencies across the tree. In Johnson-Laird's nomenclature (2002), these node-dependent approaches will be more Lamarckian than Darwinian in nature, as individuals actively improve themselves in a systematic way. This list is not complete, but describes some candidate strategies that we plan to investigate in depth.

Compound Arguments Figure 7 shows a method that correlates the available actions in all arguments by compounding them into a single list of all possible combinations. For example, if the action selected by UCB is $a_3b_1c_4$ then action a_3 is chosen for argument a, action b_1 is chosen for argument b, and action c_4 is chosen for argument c.

This approach provides a minimal degree of node dependence by correlating the actions of sibling arguments.

Boundary Correlation Figure 8 shows a subtree being constructed during a search in progress, with five arguments labelled a to e waiting to be completed along its boundary. Firstly, the most urgent argument is chosen using UCB to select among boundary updates possible from this state, then an action is selected for that argument.

Separate UCT statistics are maintained for each possible combination of {*argument*, *action*} pairs, as indicated by the arrows in Figure 8. For example, if the actions available to argument a are listed $\{a_1, a_2, ..., a_n\}$ and so on, then statistics will be maintained for combinations a_1b_1, a_1c_1, a_1d_1, a_1e_1, a_1b_2, etc. The action with the best average UCB performance over all member combinations, in conjunction with the action's own value, is selected for the chosen argument, the boundary is updated and the process continues.

Once the subtree is completed and evaluated, the back-propagation stage must update not only the values of all visited nodes but also all argument correlations for all boundaries (i.e. rooted subtrees) contained within the solution tree. The result of the search will be the solution tree with the highest reward value.

Subtree Correlation Subtree correlation is similar in principle to boundary correlation, except that instead of storing subtree boundaries with associated argument combinations, the stored subtrees are themselves associated. During subtree construction, each node is then completed by the associated subtree with the highest UCB value.

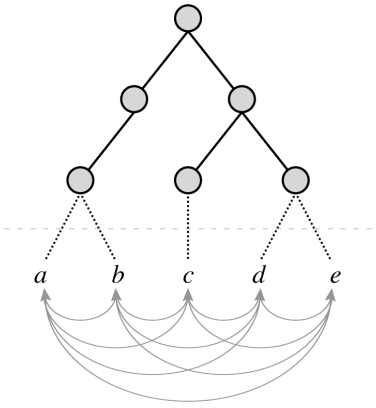

Figure 8. Boundary correlation of arguments.

This approach compares with Contextual MCTS (Rimmel and Teytaud, 2010), which partitions the search space into a number of "tiles". Techniques for transposition tables could benefit this approach (Childs et al, 2008).

The result should be that rule subsets found to work well together, even from disparate parts of the search tree, will occur more often in future solutions. Such correspondence is not guaranteed for crossover in evolutionary strategies.

The statistics thus accumulated could yield useful insights into good rule combinations, as well as indicating individual rules that are more successful. For example, game inventors may be interested to know which subsets of game rules work harmoniously together and warrant further investigation, even if the actual games produced by the system are not of great interest in themselves.

Tree-to-Sequence Conversion Other approaches suggested by combinatorial mathematics include the conversion of trees to sequences, which would then allow MCTS sequence search. This may be achieved using the *Prüfer sequence* of each tree (Prüfer, 1918) or *Euler tours* of their leaf nodes (Bender and Farach-Colton, 2000).

Application Domains

The extended model provides a framework for applying MCTS to PCG in creative domains. The proposed techniques will be tested through their application to creative tasks such as those listed below, which can be divided into two broad categories: sequence-based and tree-based.

Sequence-Based Domains

Artefact creation in sequence-based domains is typically achieved using Markovian approaches, which analyse the recent history of a problem to predict the next step that maximises some reward. Can MCTS-based methods add value to this process by predicting entire sequences?

Word Play *Pseudowords* (non-words that sound plausible within a given language) are useful for a variety of tasks: word games, poetry, psycholinguistic experiments, the creation of unique but memorable usernames, passwords, domain names, CAPTCHAs, and so on.

Pseudowords are typically constructed using Markovian methods based on the distribution of letter or syllable combinations within the target language (Keullers and Brysbaert, 2010). However, the sequential process of word construction maps neatly to the extended MCTS approach.

Music Synthesis The Continuator (Pachet, 2004) is a system that records passages played by musicians and produces continuations in a similar style, operating in real time using a Markovian model combined probabilistically with a fitness function.

Tree-Based Domains

Artefact creation in tree-based domains is typically achieved using evolutionary methods. We will investigate the use of the extended MCTS model for PCG instead.

Game Design We are developing a general game system called Mogal (Modular Game Library) in which users – or the AI – may mix and match rule modules to define new games, which can then be measured for quality through self-play. Mogal will be used to compare the performance of extended MCTS for subtree search for the generation of new rule trees (and the optimisation of existing ones) against standard evolutionary approaches.

Visual Art There are many applications of rule based systems for the automated generation of visual art, including expression trees, L-systems, context free grammars, and so on, most of which use evolutionary approaches (Romero and Machado, 2008). The extended MCTS model will be applied to a number of visual art creation tasks with known fitness functions, such as the approximation of target images in artistic styles and the generation of ornamental (e.g. celtic) designs, and its performance compared against current evolutionary methods.

Conclusion

MCTS offers a number of attractive features for AI search but has to date only been applied to particular domain types. We describe ways in which the standard algorithm may be generalised by decoupling the domain from the search, then extended to other domain types expressed as sequences and trees. This opens up the possibility of using MCTS for a range of computational creativity tasks for which Markovian processes or evolutionary strategies are typically used. The extension of MCTS to sequence search is straightforward, but its extension to embedded subtree search is complicated by its polyadic nature; several strategies for tackling this problem are presented. This work represents the first step in our investigation of MCTS for creative domains.

Acknowledgements

This work is supported by EPSRC standard grant EP/I001964. Thanks to Simon Colton, Stephen Tavener, Phillip Rohlfshagen, Greg Schmidt and the anonymous reviewers for useful comments.

References

Andres, N. 2009. Yavalath. http://www.nestorgames.com.

Auer, P.; Cesa-Bianchi, N. and Fischer, P. 2002. Finite time analysis of the multiarmed bandit problem. *Machine Learning* 47(2-3):235–256.

Bender, M. and Farach-Colton, M. 2000. The LCA Problem Revisited. In *Latin 2000: Theoretical Informatics, LNCS 1776*, 88-94. Springer:Berlin.

Bjornnsson, Y. and Finnsson, H. 2009. CadiaPlayer: A Simulation-Based General Game Player. *IEEE Computational Intelligence and AI in Games* 1(1):14-15.

Browne, C. 2008. *Automatic Generation and Evaluation of Recombination Games*. Ph.D. Thesis, Faculty of Information Technology, Queensland University of Technology.

Chaslot, G.; Bakkes, S.; Szita, I. and Spronck, P. 2008. Monte-Carlo Tree Search: A New Framework for Game AI. In *Proceedings of AIIDEC-08,* 216-217.

Childs, B.; Brodeur, J. and Kocsis, L. 2008. Transpositions and Move Groups in Monte Carlo Tree Search. In *Proceedings of CIG'08*, 389-395.

Coloum, R. 2006. Efficient Backup and Selectivity Operators in Monte-Carlo Tree Search. In *Computers and Games 2006.* 72-83.

Gelly, S.; Wang, Y.; Munos, R. and Teytaud, O. 2006. *Modification of UCT with Patterns in Monte-Carlo Go*. Technical Report 6062, INRIA, Orsay Cedex:France.

Johnson-Laird, P. 2002. How Jazz Musicians Improvise. *Music Perception* 19(3):415-442.

Keullers, E. and Brysbaert, M. 2010. Wuggy: A multilingual pseudoword generator. *Behavior Research Methods* 42(3):627-633.

Kocsis, L. and Szepesvari, C. 2006. Bandit based Monte-Carlo Planning. In *Machine Learning: ECML 2006, LNCS 4212*, Berlin:Springer. 282–293.

Koza, J.; Bennett, F.; Andre, D.; and Keane, M. 2001. Genetic Programming. In *AISB'99 Symposium on AI and Scientific Creativity*, 29-38.

Pachet, F. 2004. Beyond the Cybernetic Jam Fantasy: The Continuator. *IEEE Computer Graphics and Applications* 4(1):31-35.

Prüfer, H. 1918. Neuer Beweis eines Satzes uber Permutations". *Arch. Math. Physics.* 27:742-744.

Rimmel, A. and Teytaud, F. 2010. Multiple Overlapping Tiles for Contextual Monte Carlo Tree Search. *Applications of Evolutionary Computing* 8623:201-210.

Romero, J. and Machado, P. 2008. *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Berlin:Springer.

Schmidt, G. Omega PC Axiom version: The making of. http://boardgamegeek.com/thread/563815/.

Sturtevant, N. 2002. A Comparison of Algorithms for Multi-Player Games. In *Computers and Games 2002*. 108-122.