Automatic Discretization of Actions and States in Monte-Carlo Tree Search

Guy Van den Broeck[1] and Kurt Driessens[2]

[1] Katholieke Universiteit Leuven, Department of Computer Science, Leuven, Belgium
guy.vandenbroeck@cs.kuleuven.be
[2] Maastricht university, Department of Knowledge Engineering, Maastricht, The Netherlands
kurt.driessens@maastrichtuniversity.nl

Abstract. While Monte Carlo Tree Search (MCTS) represented a revolution in game related AI research, it is currently unfit for tasks that deal with continuous actions and (often as a consequence) game-states. Recent applications of MCTS to quasi continuous games such as no-limit Poker variants have circumvented this problem by discretizing the action or the state-space. We present Tree Learning Search (TLS) as an alternative to a priori discretization. TLS employs ideas from data stream mining to combine incremental tree induction with MCTS to construct game-state-dependent discretizations that allow MCTS to focus its sampling spread more efficiently on regions of the search space with promising returns. We evaluate TLS on global function optimization problems to illustrate its potential and show results from an early implementation on a full scale no-limit Texas Hold'em Poker bot.

1 Introduction

Artificially intelligent game players usually base their strategy on a search through the so-called game tree. This tree represents all possible future evolutions of the current game state, in principle up to the point where the game ends and the outcome is known. For many games, this game tree quickly becomes too large to fully search and discover the optimal playing strategy. Good strategies can then be found by intelligently searching only a part of the game tree. Monte-Carlo Tree Search (MCTS) is a best-first search technique that is the state of the art in game tree search. It estimates the value of future game states by simulating gameplay, until the game concludes, where the expected value of the sequence of actions is observed. Based on these observations, the algorithm carefully selects actions (or game tree nodes) for the next sample. The goal is to only sample sequences of nodes that are interesting, based on the current beliefs. A sequence of nodes can be interesting because it yields a high expected value, or because its value is still uncertain.

MCTS revolutionized research in computer-Go [1–3] and has since been applied to a.o. MDP planning [4] and Texas Hold'em Poker [5]. Standard MCTS algorithms require discrete actions and states and when applied to continuous action problems, these actions are dealt with by discretization. The problem with off-line discretization is that when the discretization is too coarse, finding a good strategy might be impossible, while when the discretization is too narrow, the branching factor of the game tree might be too high for the MCTS node selection strategy to be successful. For example, in the quasi continuous action domain of no-limit Poker, Van den Broeck et al. [5] used a stochastic universal sampling approach to discretize the betting actions.

This paper introduces Tree Learning Search (TLS) as a stochastic global optimization algorithm and its integration with the MCTS framework to circumvent the continuous action (and state)

problem and automize discretization. TLS learns a model of the function to be optimized from samples generated by MCTS. In return, MCTS uses the model learned by TLS to sample function-values in the most interesting regions of its domain. In the case of game AIs, the function to be optimized is of course the game scoring function. Conceptually, the samples generated by MCTS form a stream of training examples from which a regression tree is learnt. This regression tree is in turn used to generate new samples that are maximally informative under certain assumptions.

The rest of this paper is structured as follows. Section 2 gives a more detailed explanation of MCTS and its sampling strategy. Section 3 discusses data stream mining and more specifically, learning regression trees from streams. In Section 4, we explain how MCTS and data stream mining interact in TLS. Section 5 illustrates the behavior of the current implementation of TLS in a general function optimization setting and as a substitute for MCTS in a Poker bot after which we conclude.

2 Monte-Carlo Tree Search

The original goal of MCTS was to eliminate the need to search MINIMAX game trees exhaustively and sample from the tree instead. MCTS incrementally builds a subtree of the entire game tree in memory. For each stored node P, it also stores an estimate $\hat{V}(P)$ of the expected value $V^*(P)$, the expected value of a game state under perfect play, together with a counter $\mathrm{T}(P)$ that stores the number of sampled games that gave rise to the estimate. The algorithm starts with only the root of the tree and repeats the following 4 steps until it runs out of computation time:

Selection: Starting from the root, the algorithm selects in each stored node the branch it wants to explore further until it reaches a stored leaf. This is not necessarily a leaf of the game tree.

Expansion: One (or more) leafs are added to the stored tree as child(ren) of the leaf reached in the previous step.

Simulation: A sample game starting from the added leaf is played (using a simple and fast game-playing strategy) until conclusion. The value of the reached result (i.e. of the reached game tree leaf) is recorded.

Back-propagation: The estimates of the expected values $V^*(P)$ (and selection counter $\mathrm{T}(P)$) of each recorded node P on the explored path is updated according to the recorded result.

The specific strategies for these four phases are parameters of the MCTS approach. After a number of iterations, an action-selection strategy is responsible for choosing a good action to be executed based on the expected value estimates and the selection counter stored in each of the root's children. MCTS does not require an evaluation heuristic, as each game is simulated to conclusion. Algorithm 1 gives an overview of MCTS.

2.1 UCT Sample Selection

Node selection or sample selection as described above is quite similar to the widely studied Multi-Armed Bandit (MAB) problem. In this problem, the goal is to minimize regret[3] in a selection task with K options, where each selection a_i results in a return $r(a_i)$ according to a fixed probability distribution. The **Upper Confidence Bound** selection strategy (UCB1) is based on the Chernoff-Hoeffding limit that constrains the difference between the sum of random variables and the expected

[3] Regret in a selection task is the difference in cumulative returns compared to the return that could be attained when using the optimal strategy.

Algorithm 1 Monte-Carlo Tree Search

```
function MCTS
  root := leaf node
  repeat
    SAMPLE(root)
  until convergence
  return action leading to best child of root
function SAMPLE(node n)
  if n is a leaf of the tree in memory then
    add the children of n to the tree in memory
    simulate game until conclusion and observe reward r
  else
    select child c of n where sampling is most informative
    simulate action leading to c
    r := SAMPLE(Node c)
  update expected value estimate with r
  return r
```

value. For every option a_i, UCB1 keeps track of the average returned reward $\hat{V}(a_i)$ as well as the number of trials $\mathrm{T}(a_i)$. After sampling all options once, it selects the option that maximizes:

$$\hat{V}(a_i) + C\sqrt{\frac{\ln \sum_j \mathrm{T}(a_j)}{\mathrm{T}(a_i)}} \tag{1}$$

where $\sum_j \mathrm{T}(a_j)$ is the total number of trials made. In this equation, the average reward term is responsible for the exploitation part of the selection strategy, while the second term, which represents an estimate of the upper bound of the confidence interval on $\mathbb{E}[r(a_j)]$, takes cares of exploration. C is a parameter that allows tuning of this exploration-exploitation trade-off. This selection strategy limits the growth rate of the total regret to be logarithmic in the number of trials [6].

UCB Applied to Trees (UCT) [7] extends this selection strategy to Markov decision processes and game trees. It considers each node selection step in MCTS as an individual MAB problem. Often, UCT is only applied after each node was selected for a minimal number of trials. Before this number is reached, a predefined selection probability is used. UCT assumes that all returned results of an option are independently and identically distributed, and thus that all distributions are stationary. For MCTS, this is however not the case, as each sample will alter both $\hat{V}(a)$ and $\mathrm{T}(a)$ somewhere in the tree and thereby also the sampling distribution for following trials. Also, while the goal of a MAB problem is to select the best option as often as possible, the goal of the sample selection strategy in MCTS is to sample the options such that the best option can be selected at the root of the tree in the end. While both goals are similar, they are not identical. Nonetheless, the UCB heuristic performs quite well in practice.

3 Incremental Tree Induction for Data Stream Mining

In Data Stream Mining, the objective is to learn a model or extract patterns from a fast and never ending stream of examples. To introduce the concepts used in Tree Learning Search, we focus on stream mining techniques that learn decision or regression trees.

3.1 Very Fast Decision Tree Induction

Very Fast Decision Tree learner (VFDT) [8] performs incremental anytime decision tree induction from high-speed data streams. VFDT starts off with a single node. As new training examples are read from the stream, VFDT selects the leaf of the decision tree that matches the example and updates a set of sufficient statistics for each possible test that might split that leaf. As soon as enough samples have been gathered to be confident that a certain test is the optimal one, a new split is introduced to replace the leaf. Algorithm 2 shows a generic incremental tree learning algorithm of which VFDT is an instance.

Algorithm 2 Generic Incremental Tree Learning

```
function UpdateTree(node n, example e)
  if n is a leaf then
    for each possible test t in n do
      update sufficient statistics of t with e
    if ∃t that is probably optimal then
      split n using t and create 2 empty leaf nodes
    else
      label n with the majority class of its examples
  else
    let c be the child of n that takes e
    UpdateTree(c, e)
```

To check whether there exists a test that is probably optimal, VFDT uses *Hoeffding bounds* on the class probabilities [9] to compute bounds on the probability that the information gain of a split is higher than the information gain of all other splits. The original VFDT algorithm is restricted to training examples with nominal attributes. This restriction was removed by [10], [11] and [12] who extended VFDT with support for continuous attributes.

3.2 Incremental Regression Tree Induction

When the objective is to learn a regression tree, i.e. a tree that predicts a continuous value, a common heuristic measure to decide which test to split on is the standard deviation reduction SDR [13],

$$SDR = s_{parent} - \sum_i \frac{\mathrm{T}(child_i)}{\mathrm{T}(parent)} s_{child_i}, \tag{2}$$

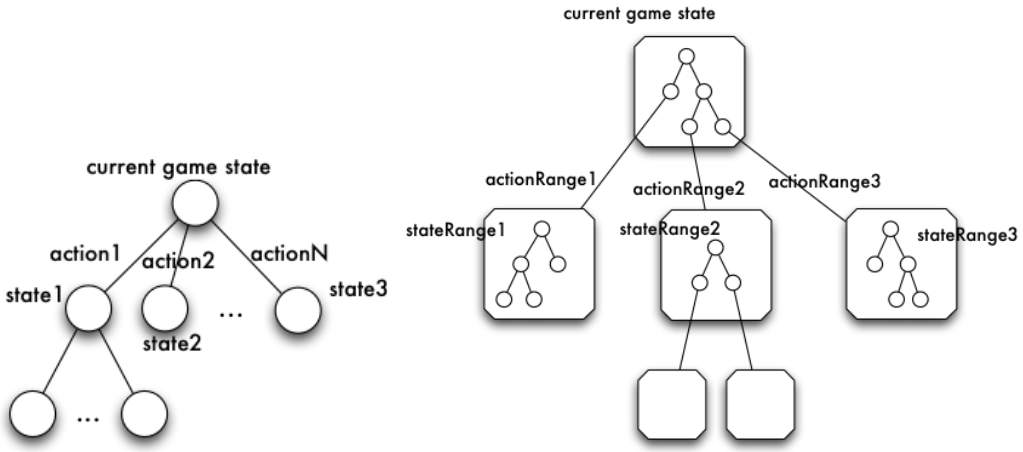
where s_n is the sample standard deviation of all examples that passed through node n.

FIMT [14] modifies VFDT for regression and is an instance of Algorithm 2 that uses SDR.

TG [15] is an incremental first-order regression tree learner that uses SDR as its heuristic measure. TG is not an instance of Algorithm 2 because it does not check whether there exists a test that is probably optimal. Instead it splits as soon as there exists a test that is probably significant, which is a looser criterium. To decide whether a split is probably significant, TG uses a standard F-test [16].

4 Tree Learning Search

By interweaving the ideas from MCTS and incremental regression tree induction, Tree Learning Search (TLS) enables MCTS to select its own discretization of actions and, as a consequence, of the game's state space. In principle, the idea behind TLS is actually quite simple. Each action node in the game-tree searched by MCTS is replaced by an incrementally build regression tree that encodes a data driven discretization of the action space. Figure 1 illustrates this change graphically. Of course, this leads to a number of conceptual changes in the game tree and in the MCTS algorithm that we will discuss in the following sections.



(a) Standard Game Tree (b) Expanded TLS nodes Tree

Fig. 1: TLS incorporation in the game tree

4.1 The Semantics of Nodes and Edges

In a standard game tree, the nodes represent game states, while the edges between them represent the action choices available to one of the players or a stochastic outcome of a game effect (e.g. a roll of a die). In the TLS game tree, the nodes representing the states of the game are the root of a so called "action tree", in which a meaningful discretization of the action is constructed. Each of the leaves of this action tree represents a range of states the agent ends up in when taking an action from the range defined by the constraints in the nodes on the path of the action tree that leads to the leaf.

For example, consider a game such as the no-limit variants of poker, in which a player can place a bet with an amount from a quasi continuous range, i.e., 0 to his full stack of chips. The size of the bet is important as it will almost certainly influence the behavior of opponents. However, there will be ranges of bet-sizes that will lead to the same behavior in opponents. For example, a small bet might tempt the opponent to put in a raise of his own; a medium sized bet might reduce his

strategy to simply calling the bet; while a large bet could force him to fold his hand[4]. Within each of these ranges, the size of the pot (and with it the actual game state) will vary with the size of the bet, but the overall progression of the game stays the same.

4.2 Learning the Action Trees

To deal with continuous action domains without having to specify an a priori discretization, the action domain is recursively subdivided into two domains whose ranges of $\hat{V}$ are significantly different. TLS uses incremental regression tree induction to discover which splits are informative, possibly growing the tree with every new MCTS sample.

Tree Growth Criterium When deciding whether or not to introduce a new split in the decision tree, we are not so much interested in finding the optimal split, but want to introduce a split that is significant. Incremental decision tree learners have mostly focussed on finding the optimal split because this influences the size of the tree at a later stage. Because we use MCTS to focus future samples on high-valued regions, if a split is significant, it will take a finite number of MCTS samples until one branch of the split will never be sampled again within the time given to the algorithm. If the split is optimal, this number of samples is expected to be minimal, but the extra samples (and thus time) needed to guarantee this optimality is expected to counter this effect.

Concept Drift MCTS causes the underlying distribution from which training examples are drawn to change. This is called **virtual concept drift** or **sampling shift** [17]. Work on handling concept drift has so far ignored virtual concept drift. Virtual concept drift does not pose any problems for TLS. After all, the learned tree does not make any wrong predictions, it only becomes too complex in certain branches where MCTS will not sample again for the remainder of its samples. If the learned tree would outgrow the available memory, these branches can safely be pruned.

Splitting internal nodes While standard incremental decision tree learners will only split leaf nodes, in TLS the leafs of the action trees represent internal nodes in the partially constructed game tree. Splitting an internal node raises the question of what to do with the subtree starting in that node. Multiple simple solutions offer themselves. Duplication of the subtree for both new nodes has the advantage of not erasing any information, but could cause problem when branches of that subtree represent illegal or misleading game-situations that are no longer possible or sufficiently probable. Simply deleting the subtree and relearning from scratch removes any such problems but also deletes a lot of information already collected. Tree restructuring procedures could counteract the drawbacks of both these simple solutions, but the additional time lost on restructuring and eventual required bookkeeping, might counteract any benefit TLS can provide.

The fact that nodes on highly promising parts of the game tree are visited most often, and therefore are the most likely ones in which significant splits will be discovered, makes this cheap reuse of experience problem an important one. It is possible (as will be illustrated by the experiments included in this paper) that the success of the TLS algorithm hinges on the resolution of this issue. It should therefor be no surprise that it is high on our future work list. Currently, for the implementation tested for this paper, the subtrees are deleted after a new split is found. That this is suboptimal is shown in the experimental results.

[4] Many online gamblers around the world would love for the Poker game to actually be this simple.

4.3 Changes to MCTS

With the introduction of actions trees, a few (limited) changes need to be made to the MCTS algorithm. The **selection** procedure now has to deal with the action trees when choosing which action to investigate. The action trees are traversed in the same way as the standard search tree, using UCT. Each passed node places a constraint on the action range to be sampled. When a leaf of the action tree is reached, an action is sampled according to these constraints.

When MCTS would **expand** the search tree from a state-node by choosing one or more actions and, for each, add a new child of a current state node to the tree, TLS connects that node to a new, empty action tree. While the **simulation** phase is unchanged, **backpropagation** of the result of the simulation now not only updates the expected values in each of the game-tree's nodes, it is also responsible for the update of the statistics in the traversed leafs of the action nodes. These updates are dictated by the incremental tree algorithm used to learn the action trees. Updating these statistics is what can cause a leaf to split. This of course raises the issues discussed above.

5 Experiments

We evaluate TLS in two steps. In a first setting, we look at the optimization capabilities of TLS in an isolated setting. This allows us to evaluate the search capabilities of TLS when combining UCT sampling with automatic tree construction for a single decision point. In a second step, we test TLS on a full MCTS problem, more specifically on Texas Hold'em Poker. This will introduce the added complexity of multiple regression trees and the resulting information re-use issues discussed above.

5.1 Function Optimization

In a first experimental setup, we use TLS as a global function optimization algorithm. While MCTS has mainly been used for adversarial search, it has also be applied to single-agent games [18], puzzles or planning problems [7, 19]. In this case, MCTS performs global optimization of a class of functions $f(\mathbf{x})$, where f is the expected value and $\mathbf{x}$ is a sequence of actions. When $\mathbf{x}$ does not represent a sequence of actions, but instead represent an atomic variable with a continuous domain, i.e. it represents a stateless game with a single continuous action, the problem maps trivially to a global optimization task.

Evaluating TLS in this degenerate setting will provide answers to a number of questions:

(Q1) Does the MCTS selection strategy succeed in focussing sampling on interesting regions of $\mathbf{x}$?
(Q2) Is TLS able to converge to the global maximum in the presence of many local maxima?

as well as illustrate its alternative ability to serve as a global optimization algorithm.

Standard problems in global function optimization literature exhibit a large number of local optima. We use two representative functions to illustrate TLS's behavior.

Sinus Function The function $f(x) = 0.6 - 0.09x + 0.1x\sin(10x)$ has a number of local maxima. (See Figure 3 for a graphical illustration.)

Six-hump Camel Back Function The six-hump camel back function is a standard benchmark in global optimization. The function is

$$f(x, y) = (4.0 - 2.1x^2 + x^4/3)x^2 + xy + (-4 + 4y^2)y^2$$

with $-3 \leq x \leq 3$ and $-2 \leq y \leq 2$. It has six local minima, two of which are global minima. (See Figure 2 for a graphical illustration.)

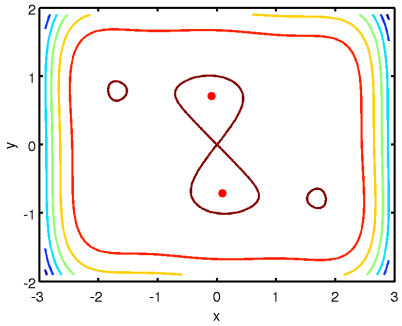
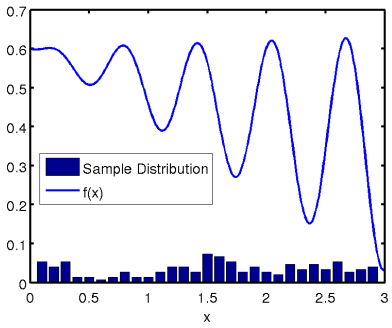
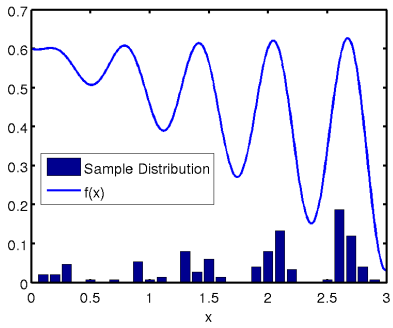


Fig. 2: Six-hump camel back function contours

To answer **Q1** we plot the sampling spread at the start and and near the end of 1000 samples. Figure 3 and Figure 4 show the spread for the sinus and camel function respectively and illustrate the focussing power of TLS. From these figures, it should be obvious that question **(Q1)** can be answered affirmatively.



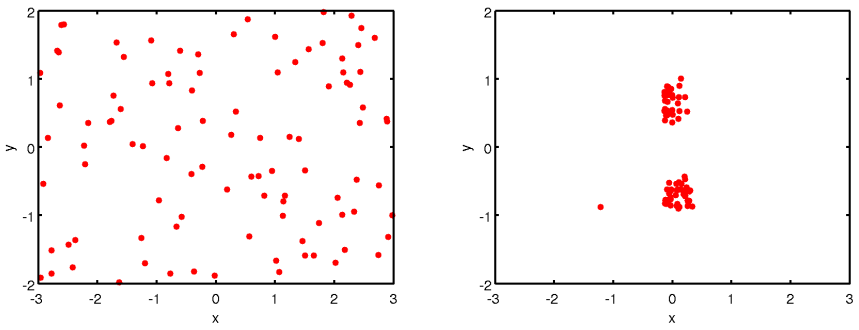
(a) Samples 1 to 150



(b) Samples 851 to 1000

Fig. 3: Sample distributions of sinus function

To answer **Q2** we plot the approximation error for the sinus and camel functions' optima in Figures 5a and 5b respectively. Again, these figures represent an affirmative answer to the question.



(a) Samples 1 to 100 (b) Samples 901 to 1000

Fig. 4: Sample distributions of camel back function

5.2 No-limit Texas Hold'em Poker

Research in computer Poker has been mainly dealing with the *limit* variant of Texas Hold'em. In limit Poker, bet sizes are fixed as well as the total number of times a player can raise the size of the pot. Only recently, attention has started shifting to the no-limit variant and the increased complexity that it brings. Gilpin et al.[20] estimate the game tree in heads-up no-limit Texas Hold'em (with bet-size discretization) to reach a node-count of about 10^{71}.

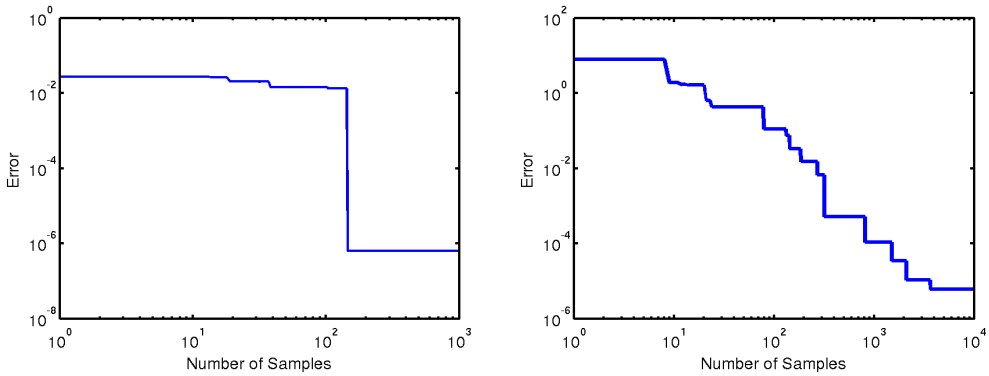
The true tournament version of Texas Hold'em is played with a maximum of 10 players per table, expanding the size of the game tree even more. Traversing the whole game tree is in this case not an option. While most existing bots can either play limit or heads-up poker, we deal with the full complexity of the poker game. The bot used in our experiments is based on the bot introduced by Van den Broeck et al. [5].

In addition to letting TLS deal with the discretization of the action or decision nodes, we also allow it to discretize the opponent nodes of the Poker game tree. The difference between these opponent nodes and the decision nodes is that we do not use the tree learned by TLS to sample opponent actions — in fact, we use the opponent model for this, just as in the original bot — but expect that the game state division as created by the regression tree will lead to more predictive subtrees lower in the game tree.

The question we would like to see answered in these experiments is:

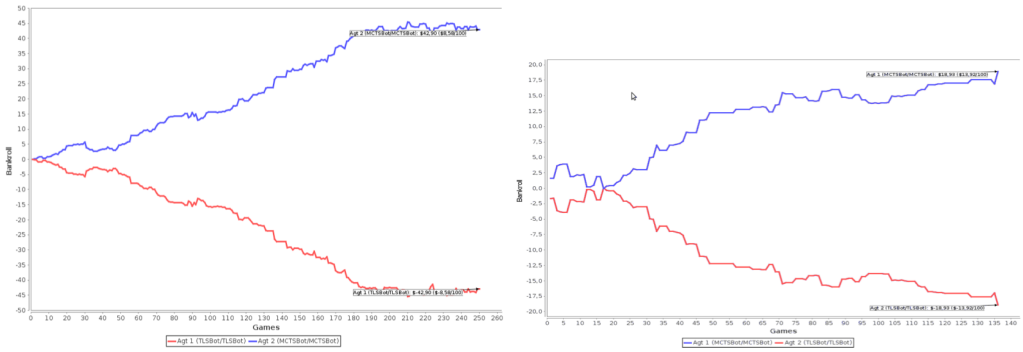
(Q3) Does the added complexity of TLS lead to better performance than standard a priori discretization in MCTS?

The current state of the TLS bot suffers badly from the information loss caused by the deletion of subtrees when a significant splitting criterion is found. Since highly promising branches of the tree will be often selected and sampled, a great deal of learning examples will pass through the nodes of these branches. This also means that many splits will appear on these branches and that a great deal of information gets lost each time a split appears. The result of this becomes obvious when putting the new TLS-based bot against the original MCTS bot as shown in Figure 6. Even when allowing the TLS-bot more time to circumvent added bookkeeping complexity and allowing it the same number of samples as the MCTS-bot, it still looses a substantial amount of credits.



(a) Sinus Function (b) Camel Function

Fig. 5: Error from Global Optimum.



(a) Equal Time (b) Equal Samples

Fig. 6: Performance of the TLS-bot vs. the MCTS bot of [5].

Obviously, for the current implementation of TLS, we have to answer question **Q3** negatively. However, based on the current implementation sub-optimalities, it is hard to make this a strong conclusion and we remain hopeful that, when TLS is able to recycle most of its discovered information after deciding to make an extra split in one of the high expectation branches, it will start to benefit from the more informed action and state discretizations.

6 Related Work

While many incremental decision tree algorithms exist [21, 8, 15, 14], a full discussion of them is out of the scope of this paper. Suffice to say that almost any incremental regression algorithm could be used by TLS, thereby opening up the use of TLS beyond continuous action spaces to other environments with very high branching factors, such as, for example, relational domains.

Most closely related to TLS is the work by Weinstein et al. [22] on planning with continuous action spaces named HOOT. In this work, planning is performed by sampling from the decision process and integrating the HOO algorithm [23] as the action selection strategy into the rollout planning structure. HOO is a continuous action bandit algorithm that develops a piecewise decomposition of the continuous action space. While this is very similar to what TLS does, there are a few important differences. First, HOOT uses a fixed discretization of the state space. This means that, for each state node in the search tree, a fixed set of discretized states is made available a priori. This avoids the internal node splitting difficulties, as each action tree will lead to one of a fixed number of state discretizations. Second, splits in HOOT are made randomly, as actions are sampled randomly from the available range. While random splits may seem strange, they are a lot cheaper to make than informed splits such as those use by TLS and in the setting used by HOOT with a fixed discretization of states, this actually makes sense. However, in the case of TLS the chosen splits also lead to a state space discretization on which further splitting of the search tree is built, in which case it makes more sense to use more informed splits.

7 Conclusions

We presented Tree Learning Search (TLS), an extension of MCTS to continuous action (and state) spaces that employs incremental decision tree algorithms to discover game state specific action (and state) discretizations. TLS adds action trees to the standard game-tree searched by MCTS that divides the continuous action space into meaningful action ranges that should help it discover regions with high expected pay-offs with less samples. Current implementations of TLS show that it works in a general function optimization setting but that information re-use is a critical issue when splitting internal nodes in the full game-tree search setting.

Future work will therefor focus on resolving the tree restructuring issue raised when splitting an internal node of the game-tree. The trade-off between information re-use and required computational and storage efforts will strongly constrain the possible solutions for this problem.

Acknowledgments We are grateful to Thijs Lemmens who was responsible for the implementation of TLS in the Poker bot as part of his Master's thesis. While his implementation was preliminary and contained a number of sub-optimalities at the time of testing, his work on the system provided us with a number of insights that will guide future work on TLS. GVdB is supported by the Research Foundation-Flanders (FWO Vlaanderen).

References

1. Gelly, S., Wang, Y.: Exploration exploitation in go: UCT for Monte-Carlo go. Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006) (2006)
2. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. Lecture Notes in Computer Science **4630** (2007) 72–83
3. Chaslot, G.M.J., Winands, M.H.M., Herik, H., Uiterwijk, J., Bouzy, B.: Progressive strategies for monte-carlo tree search. New Mathematics and Natural Computation **4** (2008) 343
4. Walsh, T., Goschin, S., Littman, M.: Integrating Sample-based Planning and Model-based Reinforcement Learning. In: Proceedings of AAAI. Number 1 (2010)

5. Van den Broeck, G., Driessens, K., Ramon, J.: Monte-carlo tree search in poker using expected reward distributions. In: Proceedings of ACML. Volume 5828 of Lecture Notes in Computer Science. (2009) 367 – 381
6. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2) (2002) 235–256
7. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. Lecture Notes in Computer Science **4212** (2006) 282
8. Domingos, P., Hulten, G.: Mining high-speed data streams. Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00 (2000) 71–80
9. Hulten, G., Domingos, P., Spencer, L.: Mining massive data streams. Journal of Machine Learning Research **1** (2005) 1–48
10. Gama, J.a., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03 (2003) 523
11. Jin, R., Agrawal, G.: Efficient decision tree construction on streaming data. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03 (2003) 571
12. Holmes, G., Richard, K., B: Tie-breaking in Hoeffding trees. In: Proceedings of the Second International Workshop on Knowledge Discovery from Data Streams. (2005)
13. Quinlan, J.: Learning with continuous classes. 5th Australian joint conference on artificial intelligence **92** (1992) 343–348
14. Ikonomovska, E., Gama, J.: Learning model trees from data streams. Discovery Science (2008) 52–63
15. Driessens, K., Ramon, J., Blockeel, H.: Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. Machine Learning: ECML 2001 (2001)
16. Driessens, K.: Relational reinforcement learning. PhD thesis, K.U.Leuven (2004)
17. Ikonomovska, E., Gama, J., Sebastião, R., Gjorgjevik, D.: Regression trees from data streams with drift detection. Discovery Science (2009) 121–135
18. Schadd, M.P.D., Win, M.H.M., Herik, H.J.V.D., b. Chaslot, G.M.J., Uiterwijk, J.W.H.M.: Single-player monte-carlo tree search. In: In Computers and Games, volume 5131 of Lecture Notes in Computer Science, Springer (2008) 1–12
19. Chaslot, G., Jong, S., takeshi Saito, J., Uiterwijk, J.: Monte-carlo tree search in production management problems. In: Proceedings of the 18th Benelux Conference on Artificial Intelligence (BNAIC'06). (2006)
20. Gilpin, A., Sandholm, T., Sørensen, T.: A heads-up no-limit Texas Hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems Richland, SC (2008) 911–918
21. Chapman, D., Kaelbling, L.: Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence. (1991) 726–731
22. Weinstein, A., Mansley, C., Michael, L.: Sample-based planning for continuous action markov decision processes. In: Proceedings of the ICML 2010 Workshop on Reinforcement Learning and Search in Very Large Spaces. (2010)
23. Bubeck, S., Munos, R., Stoltz, G., Szepesvari, C.: Online Optimization in X-Armed Bandits. In: Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, Canada (2008)