Trial-based Heuristic Tree Search for Finite Horizon MDPs

Thomas Keller
University of Freiburg
Freiburg, Germany
tkeller@informatik.uni-freiburg.de

Malte Helmert
University of Basel
Basel, Switzerland
malte.helmert@unibas.ch

Abstract

Dynamic programming is a well-known approach for solving MDPs. In large state spaces, asynchronous versions like Real-Time Dynamic Programming (RTDP) have been applied successfully. If unfolded into equivalent trees, Monte-Carlo Tree Search algorithms are a valid alternative. UCT, the most popular representative, obtains good anytime behavior by guiding the search towards promising areas of the search tree and supporting non-admissible heuristics. The global Heuristic Search algorithm AO* finds optimal solutions for MDPs that can be represented as acyclic AND/OR graphs.

Despite the differences, these approaches actually have much in common. We present the Trial-based Heuristic Tree Search (THTS) framework that subsumes these approaches and distinguishes them based on only five ingredients: heuristic function, backup function, action selection, outcome selection, and trial length. We describe the ingredients that model RTDP, AO* and UCT within this framework, and use THTS to combine attributes of these algorithms step by step in order to derive novel algorithms with superior theoretical properties. We merge Full Bellman and Monte-Carlo backup functions to Partial Bellman backups, and gain a function that both allows partial updates and a procedure that labels states when they are solved. DP-UCT combines attributes and theoretical properties from RTDP and UCT even though it differs from the latter only in the used Partial Bellman backups. Our main algorithm, UCT* adds a limited trial length to DP-UCT to inherit the global search behavior of AO* , which ensures that parts of the state space that are closer to the root are investigated more thoroughly. The experimental evaluation shows that both DP-UCT and UCT* are not only superior to UCT, but also outperform PROST, the winner of the International Probabilistic Planning Competition (IPPC) 2011 on the benchmarks of IPPC 2011.

Keywords: Planning under Uncertainty, MDPs

Acknowledgements

Thomas Keller is supported by the German Aerospace Center (DLR) as part of the Kontiplan project (50 RA 1221).

1 Introduction

Markov decision processes (MDPs) offer a general and widely used framework for decision making under uncertainty. Among the algorithms that have been applied successfully to MDPs with large state spaces are the asynchronous Dynamic Programming algorithm RTDP (Barto et al., 1995), the Heuristic Search approach AO* and UCT (Kocsis and Szepesvári, 2006), the most a popular representative of Monte-Carlo Tree Search (MCTS) (Browne et al., 2012).

The wide variety of possible choices when it comes to solving MDPs gives rise to the question which differences and commonalities exist among them. We answer this question by introducing Trial-based Heuristic Tree Search (THTS), a framework that subsumes all of these algorithms (Keller and Helmert, 2013). Specific instances of THTS can be described with only five ingredients: heuristic function, backup function, action selection, outcome selection, and trial length. We furthermore use THTS to combine ingredients of UCT, RTDP and AO* to derive algorithms that are stronger than the individual methods and even outperform PROST (Keller and Eyerich, 2012), the winner of the International Probabilistic Planning Competition (IPPC) 2011 on the benchmarks of IPPC 2011.

2 Background

In this paper, we are interested in algorithms for finite horizon MDPs S$\langle S, A, P, R, H, s_0\rangle$ with the usual semantics (Puterman, 1994) that have access to a declarative model of transition probabilities and rewards. Usually, a solution for such an MDP is a policy, i.e. a mapping from states to actions. As a policy is already expensive to describe (let alone compute) in MDPs with large state spaces, we consider algorithms that do not generate the policy offline but interleave planning of a single action and its execution, a process that is repeated H times. In the first step of a run, s_0 is set to a given initial state. In each other step, it is set to the outcome of the last action execution. We estimate the quality of an algorithm by sampling a fixed number of such runs. This approximates the expected reward of a policy π in MDP M, as the exact calculation is also intractable in large MDPs. As it is important for our notion of anytime optimal backup functions, we define the expected reward in terms of the state-value function V^π as $V^\pi(M) := V^\pi(s_0)$ with

$$V^\pi(s) := \begin{cases} 0 & \text{if } s \text{ is terminal} \\ Q^\pi(\pi(s), s) & \text{otherwise,} \end{cases}$$

where the action-value function $Q^\pi(a, s)$ is defined as

$$Q^\pi(a, s) := R(a, s) + \sum_{s' \in S} P(s'|a, s) \cdot V^\pi(s').$$

The optimal policy π^* in M can be derived from the related Bellman optimality equation (Bellman, 1957), which describes the reward for selecting the actions that yield the highest expected reward.

We define states such that the number of remaining steps is part of a state, and denote it with $s[h]$ for a state $s \in S$. A state with $s[h] = 0$ is called terminal. As the number of remaining steps must decrease by one in each state transition, i.e. $P(s'|a, s) = 0$ if $s'[h] \neq s[h]-1$, each finite-horizon MDP induces a directed acyclic graph. Moreover, we require that all policies are proper in M, a constraint that is satisfied in all benchmark domains of IPPC 2011.

3 Trial-based Heuristic Tree Search

In this section, we give a brief overview on the Trial-based Heuristic Tree Search framework. For a detailed description, we refer the reader to the full version of this paper (Keller and Helmert, 2013). The THTS framework bridges the gap between Dynamic Programming, MCTS, and Heuristic Search algorithms and can be used to model most members of these families of algorithms. THTS algorithms maintain an explicit tree of alternating decision and chance nodes. Decision nodes are tuples $n_d = \langle s, V^k\rangle$, where $s \in S$ is a state and $V^k \in \mathbb{R}$ is the state-value estimate based on the k first trials. Chance nodes are tuples $n_c = \langle s, a, Q^k\rangle$, where $s \in S$ is a state, $a \in A$ is an action, and $Q^k \in \mathbb{R}$ is the action-value estimate based on the k first trials. In the following, we denote decision nodes with n_d, chance nodes with n_c, and the set of successor nodes of n in the explicit tree with $\mathcal{S}(n)$. We abbreviate $R(s(n_c), a(n_c))$ with $R(n_c)$ and $P(s(n_d)|a(n_c), s(n_c))$ with $P(n_d|n_c)$.

Initially, the explicit tree contains only the root node n_0, a decision node with $s(n_0) = s_0$. Each trial, which can be separated in three phases, adds nodes to the tree (it explicates them). In the selection phase, the explicit tree is traversed by alternatingly choosing one or more successor nodes according to action and outcome selection. When a previously unvisited decision node is encountered, the expansion phase starts. A child is added to the explicit tree for each applicable action (or for each outcome of each applicable action) and all estimates are initialized with a heuristic. That way, all successor nodes of the currently visited node contain action-value estimates, so THTS algorithms may switch back to the selection phase, a process that continues until the desired trial length is reached (which is given when an empty set of successors is selected). All visited nodes are updated in reverse order in the subsequent backup phase, possible interrupted by additional selection and expansion phases if multiple actions or outcomes have been selected earlier. The trial finishes when the backup function is called on the root node. Another trial is started unless the root node is solved or a time constraint is met..

Algorithm 1: The THTS schema.

```
1  THTS(MDP $M$, timeout $T$):
2      $n_0 \leftarrow$ getRootNode($M$)
3      while not solved($n_0$) and time() $< T$ do
4          visitDecisionNode($n_0$)
5      return greedyAction($n_0$)
6  visitDecisionNode(Node $n_d$):
7      if $n_d$ was never visited then initializeNode($n_d$)
8      $N \leftarrow$ selectAction($n_d$)
9      for $n_c \in N$ do
10         visitChanceNode($n_c$)
11     backupDecisionNode($n_d$)
12 visitChanceNode(Node $n_c$):
13     $N \leftarrow$ selectOutcome($n_c$)
14     for $n_d \in N$ do
15         visitDecisionNode($n_d$)
16     backupChanceNode($n_c$)
```

An THTS algorithm is specified in terms of five ingredients: heuristic function, backup function, action selection, outcome selection, and trial length. We focus on different backup functions in the following due to space constraints, and refer to the full version of this work for a detailed overview on possible choices for the other ingredients. To get an intuition, we introduce popular choices in a nutshell. Action selection is a popular research area which can mostly be divided in two camps. On the one hand are algorithms like AO* or RTDP that always select the successor node with the highest action-value estimate. Combined with an admissible heuristic this can enable considerable pruning effects. On the other hand are action selection strategies that balance exploration and exploitation. One such example is UCT, where the UCB1 formula (Auer et al., 2002) is used to select actions that have been rarely tried or yielded promising results in the past. This allows the usage of non-admissible heuristic functions which are usually cheaper to compute yet similarly informative. Outcome selection is the ingredient with the least variance in well-studied algorithms – almost all algorithms select a chance node according to its probability (possibly biased by solved siblings).

The backup function defines how the information that is gathered in trials is propagated through the tree. In the THTS framework, nodes are updated only based on values of some or all of their successor nodes. In this work, we focus on algorithms that are anytime optimal, i.e. yield reasonable results quickly, improve when more trials are available and eventually converge towards the optimal decision (w.r.t. the expected reward). One way to guarantee optimality in the limit is to demand that the backup function is such that estimates converge towards the optimal value functions, i.e. $Q^k(n_c) \rightarrow Q^*(a(n_c), s(n_c))$ and $V^k(n_d) \rightarrow V^*(s(n_d))$ for all n_d, n_c and $k \rightarrow \infty$. We distinguish between full and partial backup functions. Full backup functions can only be computed for a node if each of its children in the MDP is also represented in the explicit tree, i.e. $\mathcal{S}(n_d) = \{\langle s(n_d), a, Q^k \rangle | a \in A\}$ and $\sum_{n_d \in \mathcal{S}(n_c)} P(n_d|n_c) = 1$. Partial backup functions require only one explicated child, so action-value estimates can be calculated even if only one outcome is in the explicit tree (and the one that was selected in the current trial always is). We focus on the development of such a backup function as we consider MDPs with a potentially huge number of outcomes (up to 2^{50} in our experiments).

Monte-Carlo backup MCTS algorithms like UCT are based on Monte-Carlo backups, a partial backup function which extends the current average with the latest sampled value (Sutton and Barto, 1998). As probabilities of outcomes are not used to calculate Monte-Carlo backups, they are the predominant backup function in learning scenarios where only a generative model of the MDP is available. Let $C^k(n)$ be the number of times node n has been visited among the first k trials. State-value estimate and action-value estimate are calculated with Monte-Carlo backups as

$$V^k(n_d) = \begin{cases} 0 & \text{if } s(n_d) \text{ is terminal} \\ \frac{\sum_{n_c \in \mathcal{S}(n_d)} C^k(n_c) \cdot Q^k(n_c)}{C^k(n_d)} & \text{otherwise,} \end{cases}$$

$$Q^k(n_c) = R(n_c) + \frac{\sum_{n_d \in \mathcal{S}(n_c)} C^k(n_d) \cdot V^k(n_d)}{C^k(n_c)}.$$

For Monte-Carlo backups to converge towards the optimal value function, the outcome selection strategy must be s.t. $\frac{C^k(n_d)}{C^k(n_c)} \rightarrow P(n_d|, n_c)$ for $k \rightarrow \infty$, and the action selection strategy s.t. $\frac{C^k(n_c^*)}{C^k(n_d)} \rightarrow 1$ for $k \rightarrow \infty$, where $n_c^* = \langle s, \pi^*(s), Q^k \rangle$ is the successor of n_d in the optimal policy π^*. It is often not trivial to prove that Monte-Carlo backups converge, but it was shown by Kocsis and Szepesvári (2006) for the UCT algorithm.

Full Bellman backup Another prominent method to propagate information in the tree are Full Bellman backups:

$$V^k(n_d) = \begin{cases} 0 & \text{if } s(n_d) \text{ is terminal} \\ \max_{n_c \in \mathcal{S}(n_d)} Q^k(n_c) & \text{otherwise,} \end{cases}$$

$$Q^k(n_c) = R(n_c) + \sum_{n_d \in \mathcal{S}(n_c)} P(n_d|n_c) \cdot V^k(n_d).$$

As the name implies, they are the full backup function derived from the Bellman optimality function. Unlike Monte-Carlo backups, Full Bellman backups allow a procedure that labels nodes as solved. Obviously, each algorithm that is based on Full Bellman backups and selects actions and outcomes among unsolved nodes is anytime optimal, as all states will eventually be visited with $k \to |S|$.

Max Monte-Carlo backup Both presented backup functions have their advantages and disadvantages. On our way to combine the desirable parts of both, Max-Monte-Carlo backups are a first step. They use the action-value backup function of Monte-Carlo and the state-value backup function of Full Bellman backups. In other words, a decision node is updated based on the value of its best child rather than aggregating over all children.

Like Monte-Carlo backups, Max-Monte-Carlo backups are both partial and do not rely on a generative model of the MDP. But they are also much more resilient for scenarios where different actions yield vastly different expected rewards than their counterpart where decision nodes are updated by aggregating over all successors. Our first algorithm, **MaxUCT**, uses the same ingredients as UCT, but updates nodes with the Max-Monte-Carlo backup function.

Partial Bellman backup In a second step, we aim to derive a backup function that additionally supports a solve labeling procedure. Therefore, it is necessary to exploit the declarative model by considering probabilities in the backups of action-value estimates. Or, from a different point of view, we are looking for a partial version of Full Bellman backups that does not require all successor nodes to be explicated. To calculate an estimate, we weight the outcomes that are part of the tree proportionally to their probability and to the missing outcomes:

$$V^k(n_d) = \begin{cases} 0 & \text{if } s(n_d) \text{ is terminal} \\ \max_{n_c \in \mathcal{S}(n_d)} Q^k(n_c) & \text{otherwise,} \end{cases}$$

$$Q^k(n_c) = R(n_c) + \frac{\sum_{n_d \in \mathcal{S}(n_c)} P(n_d|n_c) \cdot V^k(n_d)}{P^k(n_c)},$$

where $P^k(n_c) = \sum_{n_d \in \mathcal{S}(n_c)} P(n_d|n_c)$ is the sum of the probabilities of all outcomes that are explicated. Intuitively, we expect estimates of outcomes that are not part of the explicit tree to be comparable to the probability weighted outcomes that are explicated. Partial Bellman backups converge towards the Bellman optimality equation under selection strategies that explore the whole tree, as $P^k(n_c) \to 1$ and $Q^k(n_c) \to Q^*(a(n_c), s(n_c))$ for $k \to |S|$.

Our second algorithm, **DP-UCT**, is also identical to UCT except for the backup function. It combines properties of Dynamic Programming and UCT. It resembles MCTS as it incorporates the advantages of partial backup functions, balanced action selection and the usage of non-admissible heuristic functions. The fact that the backup function takes probabilities into account and allows solve labeling and termination when the root node is solved is just like in Dynamic Programming approaches. To our knowledge, this theoretical property was never incorporated into an algorithm that selects actions based on the UCB1 formula.

DP-UCT benefits a lot from the balanced action selection strategy at the beginning of each trial. As the uncertainty grows with the number of simulated steps, states that are far from the root often have only little influence on that decision, even if they are part of the optimal solution or crucial for some future decision. Therefore, investigating the state space close to the root more thoroughly might improve action selection with short time windows. This idea is incorporated in our main algorithm, **UCT***, which is identical to DP-UCT except that is uses a property inherent to Heuristic Search. These global search algorithms finish a trial whenever a tip node is expanded – a natural way to focus the search on states close to the root that maintains optimality in the limit. UCT* still produces the asymmetric search trees that spend more time in promising parts of the tree, but it also makes sure that it takes the time to investigate parts that turn out to be different than what they looked like at first glance.

4 Experimental Evaluation

We evaluate the algorithms MaxUCT, DP-UCT and UCT by performing experiments on 2.66 GHz Intel Quad-Core Xeon computers, with one task per core simultaneously and a memory limit of 2.5 GB. We use the IPPC 2011 benchmarks, a set of eight domains with ten problems each. Each instance is a finite-horizon MDP with a horizon of 40. Each algorithm is evaluated based on 100 runs on each problem rather than the 30 of IPPC 2011 to obtain statistically significant results. The results in Table 1 are achieved with a timeout of 2 seconds per decision. They are obtained by converting the averaged accumulated rewards of the 100 runs to relative scores on a scale from 0 to 1 for each problem, and then calculating the average over these values from all instances for each domain. A relative score of 0 is assigned to an artificial minimum policy taken from IPPC 2011, and the score of 1 is given to the planner that achieved the highest reward. The total result is calculated as the average over all domains. Additional experiments are analyzed in the full version of this work (Keller and Helmert, 2013),

	ELEVATORS	SYSADMIN	RECON	GAME	TRAFFIC	CROSSING	SKILL	NAVIGATION	Total
UCT	0.93	0.66	**0.99**	0.88	0.84	0.85	0.93	0.81	0.86
MaxUCT	**0.97**	0.71	0.88	0.9	0.86	**0.96**	0.95	0.66	0.86
DP-UCT	**0.97**	0.65	0.89	0.89	0.87	**0.96**	**0.98**	**0.98**	0.9
UCT*	**0.97**	**1.0**	0.88	**0.98**	**0.99**	**0.98**	**0.97**	**0.96**	**0.97**
PROST	0.93	0.82	**0.99**	0.93	0.93	0.82	**0.97**	0.55	0.87

Table 1: Score per domain and total scores for the IPPC 2011 benchmarks. Best results (± 0.02) are highlighted in bold.

All algorithms are implemented in the framework of the PROST planner, so they also use the sound reasonable action pruning and reward lock detection methods that are described by Keller and Eyerich (2012). The PROST planner that was used in IPPC 2011 is equivalent to our UCT base implementation, except that it uses a fixed search depth limitation of 15. Both an unlimited UCT version and PROST are included in our evaluation, the latter being the only algorithm in our comparison that is not anytime optimal. The results impressively reflect the theoretical results: DP-UCT is a clear improvement over the algorithms based on Monte-Carlo backups, clearly indicating that it pays off to directly take probabilities into account. Our main algorithm UCT* outperforms all other algorithms, including PROST, on almost all domains. Its advantages become especially apparent in the domains where DP-UCT does not have an edge over the other algorithms. The improvement in the domains with dense transition matrices, SYSADMIN, GAME OF LIFE, and TRAFFIC, shows that it pays off to search the state space close to the root more thoroughly. Moreover, the result in CROSSING TRAFFIC, where the lookahead that is needed for good policies is among the highest of the IPPC 2011 problems, shows that it also performs well if a higher horizon must be considered. This is a clear sign that the asymmetric form of the search tree with the majority of visits in the promising parts of the search space is preserved.

5 Conclusion

We have presented a novel algorithmic framework, Trial-based Heuristic Tree Search, which subsumes MCTS, Dynamic Programming, and Heuristic Search. We have identified five ingredients that distinguish different algorithms within THTS: heuristic function, backup function, action selection, outcome selection, and trial length. By step-wise combining Monte-Carlo and Full Bellman backups to Partial Bellman backups, we were able to derive the MaxUCT and DP-UCT algorithms. The latter inherits the ability to solve states by considering probabilities from Dynamic Programming, and the use of non-admissible heuristics, the good anytime behavior and the guidance to promising parts of the search space from UCT. Adding the trial length from Heuristic Search led to UCT*, an algorithm that distributes its resources even better in the search space. Our empirical evaluation shows that both DP-UCT and UCT* perform significantly better on the benchmarks of IPPC 2011 than any other considered algorithm, including the winner of IPPC 2011, PROST.

References

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47:235–256, May 2002.

Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to Act Using Real-Time Dynamic Programming. *Artificial Intelligence (AIJ)*, 72(1–2):81–138, 1995.

R.E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Cameron Browne, Edward J. 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. *IEEE Transactions Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

Thomas Keller and Patrick Eyerich. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 119–127. AAAI Press, June 2012.

Thomas Keller and Malte Helmert. Trial-based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 135–143. AAAI Press, 2013.

Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, pages 282–293, 2006.

M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, March 1998.