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 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. The Heuristic Search algorithm AO* finds optimal solutions for MDPs that can be represented as acyclic AND/OR graphs.

We introduce a common framework, Trial-based Heuristic Tree Search, that subsumes these approaches and distinguishes them based on five ingredients: heuristic function, backup function, action selection, outcome selection, and trial length. Using this framework, we describe three new algorithms which mix these ingredients in novel ways in an attempt to combine their different strengths. Our evaluation shows that two of our algorithms not only provide superior theoretical properties to UCT, but also outperform state-of-the-art approaches experimentally.

Introduction

Markov decision processes (MDPs) offer a general framework for decision making under uncertainty. Early research on the topic has mostly focused on Dynamic Programming algorithms that are optimal but only solve MDPs with very small state spaces. Value Iteration (Bellman 1957) and Policy Iteration (Howard 1960), two representatives of this kind of algorithm, need the whole state space in memory to even get started and hence do not scale well. One step towards solving MDPs with large state spaces are asynchronous versions of Value Iteration that do not search the state space exhaustively. Trial-based Real-Time Dynamic Programming (RTDP) (Barto, Bradtke, and Singh 1995), for example, uses greedy action selection and an admissible heuristic combined with Monte-Carlo samples, and only updates the states that were visited in the trial with Bellman backups.

A finite-horizon MDP induces an acyclic AND/OR graph which can be solved by the AO* algorithm (e.g. Nilsson 1980). This Heuristic Search approach gradually builds an optimal solution graph, beginning from the root node representing the initial state. It expands a single tip node in

the current best partial solution graph, which is the subgraph that can be reached by applying only greedy actions. It assigns admissible heuristic values to the new tip nodes and propagates the collected information through the acyclic graph to the root. These steps are repeated until all tip nodes in the best partial solution are terminal nodes.

The UCT algorithm (Kocsis and Szepesvári 2006) is the most popular representative of Monte Carlo Tree Search (MCTS) (Browne et al. 2012). It differs from Dynamic Programming and Heuristic Search in two important aspects: first, Monte-Carlo backups are used to propagate information in the search tree. While this allows the use of MCTS when only a generative model of state transitions and rewards is available, it is not necessary for this work as we are interested in solving finite horizon MDPs with declarative models. Even worse, it is not possible to determine if a state is solved, and sampling the same trial over and over is necessary for convergence towards an optimal policy. Second, the greedy action selection strategy used in Dynamic Programming and Heuristic Search is replaced by a method that balances exploration and exploitation by applying the UCB1 formula used to solve multi-armed bandit problems (Auer, Cesa-Bianchi, and Fischer 2002). This allows the usage of non-admissible heuristic functions, according to Bonet and Geffner (2012) a possible reason why UCT has an edge over RTDP in anytime optimal planning even though it also means that potential pruning effects inherent to greedy action selection with admissible heuristics are lost. Anytime algorithms avoid the calculation of a policy that is defined over all states that can be reached with non-zero probability by interleaving planning for a single current state with execution of the taken decision.

Despite the differences, these approaches actually have much in common. After defining the theoretical background in the next section, we present a framework for Trial-based Heuristic Tree Search algorithms (THTS). We show how algorithms can be specified in the framework by describing only five ingredients: heuristic function, backup function, action selection, outcome selection, and trial length. This is followed by the main part of this paper, where we use THTS to combine attributes of UCT, RTDP and AO* 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 that 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 (Keller and Eyerich 2012), the winner of the International Probabilistic Planning Competition (IPPC) 2011 on the benchmarks of IPPC 2011.

Background

An MDP (Puterman 1994; Bertsekas and Tsitsiklis 1996) is a 4-tuple $\langle S, A, P, R \rangle$, where S is a finite set of states; A is a finite set of actions; $P : S \times A \times S \rightarrow [0, 1]$ is the transition function, which gives the probability $P(s'|a, s)$ that applying action $a \in A$ in state $s \in S$ leads to state $s' \in S$; and $R : S \times A \rightarrow \mathbb{R}$ is the reward function. Sometimes a cost function replaces the reward function, but as the problems of minimizing costs and maximizing rewards are equivalent we use reward-based MDPs in this paper.

Usually, a solution for 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 in this paper that do not generate the policy offline, but interleave planning for a single current state $s_0 \in S$ and execution of the chosen action. This process is repeated H times, where $H \in \mathbb{N}$ is the finite horizon. 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 runs.

We define states such that the number of remaining steps is part of a state in the finite-horizon MDP $M = \langle S, A, P, R, H, s_0 \rangle$, 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.

We demand that $\sum_{s' \in S} P(s'|a, s) = 1$ for all state action pairs s, a unless $s[h] = 0$, i.e. that all actions are applicable in all non-terminal states, a property owed to the IPPC 2011 benchmark set that is used for our experiments. We transfer this constraint to our theoretical framework as it ascertains elegantly that there are no dead ends in M, that each path $s_0, \ldots, s_H$ with H action applications therefore yields a finite accumulated reward $\hat{R}$ as the sum of H finite immediate rewards $\hat{R} = \sum_{i=1}^{H} R(s_{i-1}, a, s_i)$, and that all policies are proper in M.

All algorithms considered in this paper have access to a declarative model of the MDP, i.e. transition probabilities and reward function are revealed to the algorithm. This does, of course, not mean that each algorithm must take this information into account: an algorithm that is able to cope with generative models can simply use the declarative model to generate state transitions and immediate rewards.

Even though we do not evaluate our algorithms in terms of the expected reward of a policy π in MDP M with initial state s_0, it is important for our notion of anytime optimal backup functions later in this paper. We define it 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; Bertsekas 1995), which describes the reward for selecting the actions that yield the highest expected reward:

$$V^*(s) = \begin{cases} 0 & \text{if } s \text{ is terminal} \\ \max_{a \in A} Q^*(a, s) & \text{otherwise,} \end{cases}$$
$$Q^*(a, s) = R(a, s) + \sum_{s' \in S} P(s'|a, s) \cdot V^*(s').$$

Trial-based Heuristic Tree Search

All algorithms that are considered in this paper can be described in terms of the THTS framework presented in Algorithm 1. Some well-known algorithms reduce to other algorithms in finite-horizon MDPs, and others differ slightly from their original version. Reasons include that they implement a method that deals with dead ends or cycles, both of which do not exist in finite-horizon MDPs; that they implement a sophisticated procedure to label search nodes as solved, which can be replaced in our tree-based framework with a very simple method based on successor and leaf nodes only; or that we adapt them to fit the online scenario by adding a timeout-based termination criterion. Among the algorithms that reduce to others are HDP (Bonet and Geffner 2003a) and LRTDP (Bonet and Geffner 2003b), which differ from Depth First Search and RTDP only in how states are labeled as solved, and LAO* (Hansen and Zilberstein 2001), which is a modification of AO* for cyclic graphs.

The THTS schema bridges the gap between Dynamic Programming, MCTS, and Heuristic Search algorithms for finite-horizon MDPs. Algorithm 1 shows THTS as a tree search algorithm, even though a version that does not unfold the directed acyclic graph into an equivalent tree is more efficient. In the directed acyclic graph search version, duplicate searches are avoided if the same state can be reached along different paths. Nevertheless, both versions solve the same class of problems. Moreover, some precaution must be taken for the adaption of our base algorithm UCT to directed acyclic graphs (Childs, Brodeur, and Kocsis 2008; Saffidine, Cazenave, and Méhat 2012), which would detract from the focus of this paper. For these reasons, we only present the tree search version here.

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

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)

$n_c = \langle s, a, Q^k \rangle$, 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 consists of phases: in the selection phase, the explicit tree is traversed by alternatingly choosing successor nodes according to action and outcome selection. When a previously unvisited decision node (i.e. a tip node in the explicit graph) is encountered, the expansion phase starts. A child is added to the explicit tree for each action, and a heuristic value is assigned to all estimates. This leads to the same situation as in the selection phase, where all successor nodes of the currently visited node have action-value estimates. THTS algorithms therefore switch back to the selection phase and alternate between those phases until an empty set of successors is selected (which determines the trial length). All visited nodes are updated in reverse order in the subsequent backup phase. If a node is reached in this process with children that were selected earlier but not visited yet in the current trial, the selection phase starts again. A trial finishes when the backup function is called on the root node. This process is repeated until time constraints do not allow for another trial.

Our framework bears some obvious similarities with other schemata. The initialization of estimates of nodes with a heuristic function is borrowed from Heuristic Search methods. The MCTS schema can be derived from THTS by replacing the initialization of novel nodes with a simulation phase (Browne et al. 2012). A heuristic function that resembles the simulation phase therefore allows to model any MCTS algorithm within our framework. Moreover, THTS is related to Bonet and Geffner's (2003a) FIND-and-REVISE schema. The selection phase of THTS can be regarded as a FIND step, and the REVISE step corresponds roughly to our backup phase. THTS is tailored to finite-horizon MDPs, though, and exploits the fact that the search space is a tree.

Despite the commonalities, a framework subsuming Dynamic Programming, MCTS and Heuristic Search has not been described before. An THTS algorithm can be specified in terms of five ingredients: heuristic function, backup function, action selection, outcome selection, and trial length. In the following, we will discuss these attributes and show how several algorithms can be modeled in the framework.

Heuristic Function

Algorithms that compute heuristic estimates have received much attention in recent years. In this work, we are not interested in the methods themselves – even though previous work has shown their crucial influence in THTS algorithms (Gelly and Silver 2007; Eyerich, Keller, and Helmert 2010) – but only briefly discuss the properties that influence the choice of ingredients. It is common terminology to call a heuristic function admissible if it never underestimates the reward, and blind if it always maps to the same constant.

We consider two possible answers to the question which successors of a previously unvisited decision node n_d are initialized. In action-value or Q-value initialization, a heuristic $h : S \times A \mapsto \mathbb{R}$ calculates an estimate that is used to initialize the action-value estimates of the successors of n_d. If state-value initialization is used, a heuristic value $h : S \mapsto \mathbb{R}$ is assigned to the state-value estimates of the grandchildren of n_d.

Backup Function

The backup function defines how the knowledge on state-value estimates $V^k(n_d)$ and action-value estimates $Q^k(n_c)$ that is gathered in the trials is propagated through the tree. Depending on the algorithm, any number of additional values might be updated, e.g. the variable that counts the number of visits in UCT, or a solve label that indicates that a node's estimate has converged. In the THTS framework, nodes are updated only based on values of some or all of their successor nodes. As we also consider problems with a huge number of outcomes, we use the decomposed representation for chance nodes of the PROST planner (Keller and Eyerich 2012). It ensures that each chance node has only 2 successors by representing 2^n possible outcomes as a chain of n chance nodes, thereby allowing efficient backups.

Even though it is not mandatory for algorithms in the THTS framework, this work focuses 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. 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 $\sum_{n_d \in \mathcal{S}(n_c)} P(n_d|n_c) = 1$. With the ingredients discussed in this work, full backup functions can only be paired with state-value initialization, as action-value initialization does

not ascertain that all children of all nodes that are visited in a trial are explicated (it does for decision nodes). Partial backup functions require only one child, so action-value estimates can be calculated even if only one outcome is in the explicit tree (and the selected one always is).

A backup function induces a contributing subtree, which is the subtree of the explicit tree that contains all nodes that contribute to the state-value estimate of the root node. The contributing subtree is either identical to the explicit tree or to the best partial solution graph (as induced by AO^*) in the backup functions considered in this paper. The former is given for backup functions that calculate state-value estimates by aggregating over action-value estimates, while the latter is true in functions that maximize instead.

Monte-Carlo backup The MCTS algorithms UCT and BRUE (Feldman and Domshlak 2012) 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 scenarios where only a generative model of the MDP is available (e.g. in Reinforcement Learning). 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

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

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

The contributing subtree is identical to the explicit tree as both functions aggregate over all children. This causes a potential pitfall: if a node n_d in the best solution graph has a child n_c that yields a very low reward compared to an optimal sibling, a single trial over n_c can bias $V^k(n_d)$ disproportionately for many trials.

Given the right mix of ingredients, Monte-Carlo backups nevertheless converge towards the optimal value function. Then the outcome selection strategy must be s.t. $\frac{C^k(n_d)}{C^k(n_c)} \to P(n_d|, n_c)$ for $k \to \infty$, and the action selection strategy s.t. $\frac{C^k(n_c^*)}{C^k(n_d)} \to 1$ for $k \to \infty$, where $n_c^* = \langle s, \pi^*(s), Q^k \rangle$ is the successor of n_d in the optimal policy π^* (we assume for simplicity and w.l.o.g. that there is exactly one optimal policy). 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. The pitfall described in the context of Monte-Carlo backups is not given in algorithms that use Full Bellman backups, as state-value estimates are updated based on the best successor. That means that the contributing subtree is the best partial solution tree. Moreover, such an algorithm can be equipped with 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 $k \to \infty$.

Action Selection

The variation regarding the action selection strategy is surprisingly small in the literature and can mostly be divided in two camps. On the one hand are algorithms like AO^* or RTDP that always act greedily, i.e. they select the successor node with the highest action-value estimate. On the other hand are algorithms like UCT and AOT, which balance exploration and exploitation by applying techniques borrowed from the multi-armed bandit problem (Berry and Fristedt 1985): AOT applies ϵ-greedy action selection, where the greedy successor is selected with a given probability, and the successor with the biggest potential impact otherwise. UCT selects the child that maximizes the UCB1 formula, a strategy that minimizes the regret in multi-armed bandit problems by favoring successor nodes that were visited rarely (exploration) or led to promising results in previous trials (exploitation). An attempt to enhance this technique is BRUE, where action selection follows an alternating schema of applying the UCB1 formula and sampling uniformly.

Successful anytime optimal algorithms that use greedy action selection like RTDP or AO^* pair it with an admissible heuristic. To obtain equivalents of these algorithms with the ingredients discussed in this paper, they must additionally be paired with a full backup function and, subsequently, a state-value initialization. This is because it is necessary for convergence of greedy algorithms that all children of all nodes that are visited in a trial contain admissible values at all times. Otherwise, the optimal choice might never be selected. Note that it is possible to design backup functions for optimal greedy algorithms where partial backup functions and action-value initialization are sufficient, e.g. with a common upper bound for unvisited outcomes. We are not aware of an optimal algorithm that uses such an initialization, though. GLUTTON (Kolobov et al. 2012), the LRTDP-based runner-up at IPPC 2011, uses subsampling to deal with this situation, but it loses optimality in the process.

Greedy and balanced action selection also use slightly different labeling procedures. Greedy action selection can entail considerable pruning effects (if paired with the aforementioned ingredients), as a node can be labeled as solved if it is a leaf or if its greedy successor is solved, i.e. it is possible that the tree is not searched exhaustively. Balanced action selection does not come with this advantage. There, a node can only be labeled as solved if it is a leaf or if all its children are solved. If paired with a backup function that allows for labeling, obviously only unsolved successor nodes are considered for action selection in THTS algorithms.

Note that our framework also allows the selection of more than one action, as we believe this might be an opportunity for future research. As an anytime optimal algorithm must not spend too much time in a single trial, a lot of precaution is necessary when designing such a selection method, though. Due to this, we do not consider algorithms like Value Iteration, Depth-First Search, or Learning Depth-First Search (Bonet and Geffner 2006) in this paper even though they can be specified within this framework.

Outcome Selection

After an action has been chosen, the outcome of applying that action must be determined (again, selecting multiple is possible but not considered here). There is only little research dealing with outcome selection. Almost all algorithms we are aware of perform Monte-Carlo sampling, i.e. they choose an outcome according to its probability or, in algorithms that label states as solved, according to its probability biased by solved siblings. The only algorithm that explicitly uses a method different from Monte-Carlo sampling is AOT, where the outcome is selected that has the biggest potential impact. Bonet and Geffner (2012) state that their method is superior to Monte-Carlo sampling, especially in the absence of informed heuristics.

Trial Length

Obviously, all algorithms finish a trial when a leaf node is reached, and many, including RTDP and its variants, only determine the trial length based on this criterion. There is also a variant of RTDP that finishes trials after each heuristic calculation – it is commonly referred to as AO*. As only tip nodes are important in these global search algorithms, it is rather unusual to describe a method like AO* in a trial-based framework. If the trial length is determined such that new information is propagated to the root node right away they can easily be modeled within the THTS framework, though.

UCT has also been described in both versions. In its original form (Kocsis and Szepesvári 2006), it starts the backup phase when a previously unvisited node is visited like a global search algorithm. A version that finishes a trial only when a leaf is encountered has been described for the Canadian Traveler's Problem (Papadimitriou and Yannakakis 1991) by Eyerich, Keller, and Helmert (2010). The PROST planner is a domain-independent version of this idea, which additionally limits the search depth artificially by considering a decreased horizon. PROST thereby loses optimality, and a good value for the search depth limitation is domain specific and can, following Kolobov, Mausam, and Weld (2012), only be based on a guess.

Remotely related to the trial length are reverse iterative deepening search versions of THTS algorithms, as incorporated for example in GLUTTON. These cannot be modeled within our framework, but a simple driver routine that calls THTS with an increasing number of remaining steps in the initial state is sufficient to support them. As all iterations but the last have to solve the root state, reverse iterative deepening search is independent from the underlying algorithm barring pruning effects. While such algorithms raise interesting questions, they are beyond the scope of this paper.

Algorithms

We investigate now how THTS ingredients can be grouped in novel ways to derive stronger algorithms for MDPs with a comparably small set of actions (at most 50 in our experiments) yet potentially many outcomes (up to 2^{50} in our experiments). In such MDPs, it is desirable to use an algorithm that supports Q-value initialization, as the number of children of a decision node is equal to the number of actions, while the number of its grandchildren can be as large as that number multiplied with the size of the state space – an extreme case that actually occurs in the benchmarks used for evaluation in this paper.

Bonet and Geffner (2012) argue that greedy action selection has poor anytime properties, mostly because it does not allow the use of non-admissible heuristic functions which are usually cheaper to compute yet similarly informative. The success of algorithms that balance exploration and exploitation seems to prove them right, and we also follow this approach in our algorithms. UCT seems like a good base algorithms which additionally supports non-admissible heuristics and Q-value initialization. It bases its action selection on balancing exploration and exploitation with the UCB1 formula for the multi-armed bandit problem. UCT chooses the successor node n_c in node n_d that maximizes

$$B \cdot \sqrt{\frac{\log C^k(n_d)}{C^k(n_c)}} + Q^k(n_c),$$

where $C^k(n)$ is the number of visits of node n during the first k trials, and B is a bias parameter that is set to $V^k(n_d)$ in our experiments as in the PROST planner.

Even though we believe that there might be even better action selection strategies for finite-horizon MDP planning than the regret minimization based strategy of UCT, it would be beyond the scope of this paper to derive one. All algorithms that are described in the following therefore base action selection on the UCB1 formula. The same reasoning applies to the Monte-Carlo sampling that is used for outcome selection. The heuristic function is the third ingredient that is not the focus of this work. We use the non-admissible Q-value initialization of the PROST planner. It is based on an iterative deepening search on the most-likely outcome determinization, which terminates when a timeout or a satisfying degree of informativeness is reached (for details, see Keller and Eyerich 2012).

Of course, UCT also incorporates some properties that can be improved in the setting we are dealing with. Most notably, we are interested in using a backup function that supports a solve labeling procedure. Our experimental evaluation will show that it is a severe drawback if it is not possible to determine when the estimate of a node has converged. Moreover, the ability to compute provably optimal policies is a theoretical property that is desirable to have. As Full Bellman backups cannot be used with non-admissible heuristics, we derive a novel backup function that combines the strengths of both functions in two steps.

MaxUCT　The only ingredient that distinguishes the Max-UCT algorithm from UCT is the used backup function. We

	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.

perform a first step in combining Monte-Carlo and Full Bellman backups by merging the action-value backup function of the former with the state-value backup function of the latter. In other words, we update decision nodes based on the value of its best child rather than aggregating over all children. Formally, we define Max-Monte-Carlo backups as:

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

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

Kocsis and Szepesvári's (2006) proof of optimality for UCT also holds for MaxUCT as the action selection method UCB1 never stops exploring, and as all outcomes will be sampled proportionally to their probability for $k \to \infty$. The pitfall discussed in the context of Monte-Carlo backups does not apply anymore for MaxUCT because the contributing subtree of Max-Monte-Carlo backups is identical to the best partial solution tree. Moreover, Max-Monte-Carlo backups are both partial and do not rely on a generative model of the MDP, so they can be used in the same scenarios as UCT.

DP-UCT While Max-Monte-Carlo backups are a first step towards merging Monte-Carlo and Full Bellman backup functions, a solve labeling procedure is not supported. 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 rely on all successor nodes in the MDP. 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 $k \to \infty$. Full Bellman backups can be seen as the special case of Partial Bellman backups where all outcomes are explicated in the tree.

DP-UCT, the algorithm that uses Partial Bellman backups rather than Monte-Carlo backups, 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.

UCT* The balanced action selection strategy of UCT, which guides the search towards parts of the state space that have been rarely visited or that led to promising results in previous trials, is a key factor of the algorithm's good anytime behavior. It leads to an asymmetric tree that is skewed towards more important regions of the search space. While the direction of search is thereby taken into account, the depth of search is not. After all, we are not interested in complete policies but only in the next decision. 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.

DP-UCT benefits a lot from the balanced action selection strategy at the beginning of each trial, but the further from the root the lesser the benefit. We therefore alter another ingredient, the trial length, and combine DP-UCT with 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. The resulting algorithm, UCT*, still produces the asymmetric search trees that spend more time in promising parts of the tree, but it does build them in a way that resembles Breadth-First Search more than Depth-First Search. It thus makes sure that it takes the time to also investigate parts that turn out to be different than what they looked like at first glance.

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
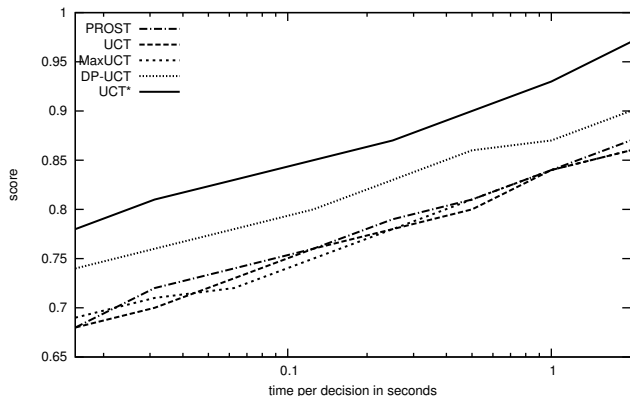
Figure 1: Total scores on the benchmarks of IPPC 2011 with increasing time per decision.

a memory limit of 2.5 GB. We use the finite-horizon MDPs from 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.

There is a notable implementation difference between UCT and PROST on the one hand and the three novel algorithms on the other: we implemented two versions of each of the five algorithms, one where the values from the Q-value initialization are only considered for action selection, and one where the heuristic is additionally used in the backup operation of the parent node (i.e. it is propagated through the tree). We observed that propagating the heuristic drastically improves the behavior of the algorithms that maximize over all children in the state-value backup functions, and the other way around in those that use the sum. It seems that not propagating the heuristic leads to trees where an insufficient amount of information is used in nodes close to the root for MaxUCT, DP-UCT and UCT*. And propagating heuristic estimates from all nodes that have ever been initialized through the tree diminishes the influence of the information that is actually important in UCT and PROST. All results presented in this paper are from the respective version that performed better in total.

The comparison of our algorithms is based on the time per decision, with time windows of 2^i seconds for $i \in \{-6, \ldots, 1\}$ (from ≈ 0.015 to 2 seconds). 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.

We evaluate the algorithms based on several aspects derived from this experiment. The results in Table 1 are achieved with the highest timeout of 2 seconds. 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. We performed the same experiment also with blind versions where all nodes n_c are initialized with a constant heuristic that sets $Q^k(n_c)$ to 0. Note that this is non-admissible in the IPPC 2011 domains due to the presence of positive and negative rewards. We will use some of those results in the following discussion, but omit detailed results for space reasons.

Figure 1 shows total scores for increasing timeouts. The rightmost points on the curve therefore correspond to the total results in Table 1. Regardless of the timeout, all results are normalized with the same minimum policy and the best result among all algorithms and timeouts. Figure 2 shows the average number of remaining steps in the first root state that was labeled as solved in a run for DP-UCT and UCT*, the algorithms that use a labeling procedure. It indicates how early in a run an algorithm started to make optimal decisions (i.e. the closer to the horizon the better). Figure 3 shows the average runtime in percent of the runtime of our base algorithm UCT. Again, only DP-UCT and UCT* are depicted, as differences in runtimes are mostly due to solved states, and as only those algorithms support solve labeling.

MaxUCT The MaxUCT algorithm differs from UCT only in that it uses Max-Monte-Carlo backups. Therefore, it builds its estimates on the best partial solution graph, which is a theoretical improvement over Monte-Carlo backups. Our experimental results back up the expectations at least partially. In comparison with the baseline UCT algorithm, it yields the same total score over all domains, yet with some numbers shifted between the scores for domains.

There are two domains with clearly inferior results compared to UCT. RECON is a special case where all algorithms that propagate heuristic values perform worse than the other algorithms. We believe that the quality of the heuristic function is more important in algorithms that propagate those values, and that the PROST heuristic gives rather poor estimates in RECON. This is backed up by the fact that the blind versions yield results that are similar to the informed ones. The same applies to NAVIGATION, but it becomes only apparent in MaxUCT as the solve labeling procedure of DP-UCT and UCT* more than compensates for it.

The results in all other domains are in favor of MaxUCT. CROSSING TRAFFIC and ELEVATORS are good examples where the altered backup function pays off. Both are domains where long sequences of actions have to be planned in advance, and where the difference between good and bad choices is highest – clearly a setting where maximizing over successors to compute state-value estimates is favorable.

DP-UCT Rather than weighting outcomes by sampling them according to the probability and averaging over all tri-
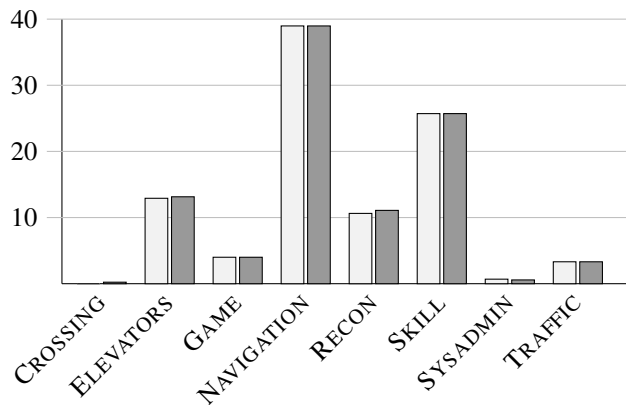
Figure 2: Average number of remaining steps in the first solved root state of a run for DP-UCT (light gray) and UCT* (dark gray).

Figure 3: Average total runtime of DP-UCT (light gray) and UCT* (dark gray) in percent of the runtime of UCT.

als, the given probabilities are directly used to calculate estimates in DP-UCT. As it takes several trials until an action-value has converged with averaging over outcomes, one would expect that DP-UCT has the edge over UCT especially early on, and Figure 1 shows that the difference between the two is indeed highest with lower timeouts and a significant improvement regardless of the timeout.

Besides the RECON domain, where DP-UCT also suffers from the comparably bad heuristic, it is always at least on par with UCT, and it vastly outperforms the baseline algorithm in the NAVIGATION domain. Figure 2 is a good indicator for why this is the case. The first root state that is labeled as solved is a state with 39.0 remaining steps in average. In the IPPC 2011 setting with a horizon of 40, it is thereby an optimal solver for most of the 10 problems (in average, only the first decision is not a provably optimal choice). The solve labeling also allows DP-UCT to save time compared to UCT. Not only does it yield a significantly higher reward, Figure 3 shows that it does so in only 14.7% of the time.

Especially the improvement in the NAVIGATION domain makes DP-UCT an algorithm with superior total results. Solve labeling not only leads to better results, but also allows for more than 20% time savings in total by computing provably optimal policies. The results clearly show that considering probabilities in the backup function pays off.

UCT* The results of our main algorithm, UCT*, impressively reflect the theoretical results: it outperforms all other algorithms, including PROST, on all domains but RECON, almost always yielding the best score among our algorithms. 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 SYSADMIN, GAME OF LIFE, and TRAFFIC, where states close to the root are most influential, 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
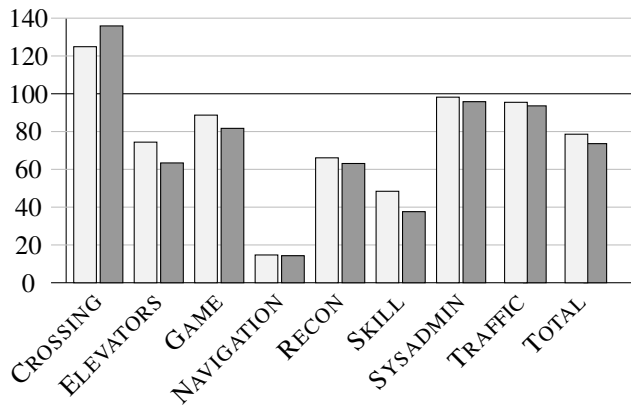
horizon must be considered. This is a clear sign that the asymmetric form of the search tree with the majority of visits in the promising parts of the search space is preserved.

As UCT* updates all visited states in a trial whenever a node is expanded, it must perform more backup steps than DP-UCT. Figure 2 and 3 indicate that the overhead of additional backups is negligible. In fact, the total time savings of 26.4% are even higher for UCT*. We believe that this is because "good" states are reached more often in execution due to the improved policy. Last but not least, UCT* also shines regarding the anytime behavior. It achieves the best result among all algorithms regardless of the time per decision. Figure 1 shows that UCT* provided with only 0.125 seconds already achieves better results than the PROST planner with 2 seconds (i.e. in $\frac{1}{16}$ of the time), and better results even than DP-UCT, the second best performing algorithm, if provided with 0.25 seconds.

Conclusion

We have presented a novel algorithmic framework, Trial-based Heuristic Tree Search, which subsumes Monte-Carlo Tree Search, 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. We described several existing algorithms within this framework and derived three novel ones.

By combining Monte-Carlo and Full Bellman backups to Partial Bellman backups, we were able to derive the Max-UCT 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 DP-UCT and UCT* not only perform significantly better on the benchmarks of IPPC 2011, but also in less time than any other considered algorithm, including the winner of IPPC 2011, PROST.

Acknowledgments

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47:235–256.

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

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

Berry, D., and Fristedt, B. 1985. *Bandit Problems*. Chapmann and Hall.

Bertsekas, D., and Tsitsiklis, J. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control*. Athena Scientific.

Bonet, B., and Geffner, H. 2003a. Faster Heuristic Search Algorithms for Planning with Uncertainty and Full Feedback. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 1233–1238.

Bonet, B., and Geffner, H. 2003b. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, 12–21.

Bonet, B., and Geffner, H. 2006. Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and Its Application to MDPs. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, 142–151.

Bonet, B., and Geffner, H. 2012. Action Selection for MDPs: Anytime AO* Versus UCT. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*. To Appear.

Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions Computational Intelligence and AI in Games* 4(1):1–43.

Childs, B. E.; Brodeur, J. H.; and Kocsis, L. 2008. Transpositions and Move Groups in Monte Carlo Tree Search. In *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games (CIG)*, 389–395.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-Quality Policies for the Canadian Traveler's Problem. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, 51–58.

Feldman, Z., and Domshlak, C. 2012. Online Planning in MDPs: Rationality and Optimization. *CoRR* abs/1206.3382.

Gelly, S., and Silver, D. 2007. Combining Online and Offline Knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 273–280.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence (AIJ)* 129(1–2):35–62.

Howard, R. 1960. *Dynamic Programming and Markov Processes*. MIT Press.

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

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

Kolobov, A.; Dai, P.; Mausam; and Weld, D. 2012. Reverse Iterative Deepening for Finite-Horizon MDPs with Large Branching Factors. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 146–154.

Kolobov, A.; Mausam; and Weld, D. 2012. LRTDP vs. UCT for Online Probabilistic Planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*, 1786–1792.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers Inc.

Papadimitriou, C. H., and Yannakakis, M. 1991. Shortest Paths Without a Map. *Theoretical Computer Science* 84(1):127–150.

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

Saffidine, A.; Cazenave, T.; and Méhat, J. 2012. UCD : Upper Confidence bound for rooted Directed acyclic graphs. *Knowledge Based Systems* 34:26–33.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.