Action Selection for MDPs: Anytime AO* vs. UCT

Blai Bonet
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Hector Geffner
ICREA & Universitat Pompeu Fabra
08003 Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

In the presence of non-admissible heuristics, A* and other best-first algorithms can be converted into anytime optimal algorithms over OR graphs, by simply continuing the search after the first solution is found. The same trick, however, does not work for best-first algorithms over AND/OR graphs, that must be able to expand leaf nodes of the explicit graph that are not necessarily part of the best partial solution. Anytime optimal variants of AO* must thus address an exploration-exploitation tradeoff: they cannot just "exploit", they must keep exploring as well. In this work, we develop one such variant of AO* and apply it to finite-horizon MDPs. This Anytime AO* algorithm eventually delivers an optimal policy while using *non-admissible random heuristics that can be sampled*, as when the heuristic is the cost of a base policy that can be sampled with rollouts. We then test Anytime AO* for action selection over large infinite-horizon MDPs that cannot be solved with existing off-line heuristic search and dynamic programming algorithms, and compare it with UCT.

Introduction

One of the natural approaches for selecting actions in very large state spaces is by performing a limited amount of lookahead. In the contexts of discounted MDPs, Kearns, Mansour, and Ng have shown that near to optimal actions can be selected by considering a *sampled* lookahead tree that is sufficiently *sparse*, whose size depends on the discount factor and the suboptimality bound but not on the number of problem states (**?**). The UCT algorithm (**?**) is a version of this form of Monte Carlo planning, where the lookahead trees are not grown depth-first but 'best-first', following a selection criterion that balances 'exploration' and 'exploitation' borrowed from the UCB algorithm for multi-armed bandit problems (**?**).

While UCT does not inherit the theoretical properties of either Sparse Sampling or UCB, UCT is an *anytime optimal algorithm* for discounted or finite horizon MDPs that eventually picks up the optimal actions when given sufficient time. The popularity of UCT follows from its success in the game of Go where it outperformed all other approaches (**?**); a success that has been replicated in other models and tasks such as Real-Time Strategy Games (**?**), General Game Playing (**?**), and POMDPs (**?**).

An original motivation for the work reported in this paper was to get a better understanding of the success of UCT and related Monte-Carlo Tree Search (MCTS) methods (**?**).[1] It has been argued that this success is the result of *adaptive sampling methods*: sampling methods that achieve a good exploration-exploitation tradeoff. Yet, adaptive sampling methods like Real-Time Dynamic Programming (RTDP) (**?**) have been used before in planning. For us, another important reason for the success of MCTS methods is that they address a slightly different problem; a problem that can be characterized as:

1. anytime *action selection* over MDPs (and related models) given a time window, resulting in good selection when the window is short, and near to optimal selection when window is sufficiently large, along with
2. non-exhaustive search combined with the ability to use informed *base policies* for improved performance.

From this perspective, an algorithm like RTDP fails on two grounds: first, RTDP does not appear to make best use of short time windows in large state spaces; second, and more importantly, RTDP can use admissible heuristics but not informed base policies. On the other hand, algorithms like Policy Iteration (**?**), deliver all of these features except one: they are exhaustive, and thus even to get started, they need vectors with the size of the state space. At the same time, while there are non-exhaustive versions of (asynchronous) Value Iteration such as RTDP, there are no similar 'focused' versions of Policy Iteration ensuring anytime optimality. Rollouts and nested rollouts are two practical and focused versions of Policy Iteration over large spaces (**?**; **?**), yet neither one aims at optimality.[2]

In this work, we introduce a new, simple heuristic search algorithm designed to address points 1 and 2 above, and compare it with UCT. We call the new algorithm *Anytime AO**, because it is a very simple variation of the classical AO* algorithm for AND/OR graphs (**?**) that is optimal even

[1] For a different attempt at understanding the success of UCT, see (**?**).

[2] Nested rollouts can deliver optimal action selection but for impracticable large levels of nesting.

in the presence of non-admissible heuristics. Anytime AO* is related to recent anytime and on-line heuristic search algorithms for OR graphs (**?**; **?**; **?**; **?**). It is well known that A* and other best-first algorithms can be easily converted into anytime optimal algorithms in the presence of non-admissible heuristics by simply continuing the search after the first solution is found. The same trick, however, does not work for best-first algorithms over AND/OR graphs that must be able to expand leaf nodes of the explicit graph that are not part of the best partial solution. Thus, Anytime AO* differs from AO* in two main points: first, with probability p, it expands leaf nodes that are not part of the best partial solution; second, the search finishes when time is up or there are no more leaf nodes to expand at all (not just in the best solution graph). Anytime AO* delivers an optimal policy eventually and can also use *random heuristics that are not admissible and can be sampled*, as when the heuristics result from rollouts of a given *base policy*.

MDPs

Markov Decision Processes are fully observable, stochastic state models. In the discounted reward formulation, an MDP is given by a set S of states, sets $A(s)$ of actions applicable in each state s, transition probabilities $P_a(s'|s)$ of moving from s into s' when the action $a \in A(s)$ is applied, real rewards $r(a, s)$ for doing action a in the state s, and a discount factor γ, $0 < \gamma < 1$. A solution is a policy π selecting an action $\pi(s) \in A(s)$ in each state $s \in S$. A policy is optimal if it maximizes the expected accumulated discounted reward.

Undiscounted finite horizon MDPs replace the discount factor γ by a positive integer horizon H. The policies for such MDPs are functions mapping nodes (s, d) into actions $a \in A(s)$, where s is a state and d is the horizon to go, $0 < d \leq H$. Optimal policies maximize the total reward that can be accumulated in H steps. Finite horizon MDPs are acyclic as all actions decrement the horizon to go by 1.

Undiscounted infinite horizon MDPs are like discounted MDPs but with discount $\gamma = 1$. For such MDPs to have well-defined solutions, it is common to assume that rewards are negative, and thus represent positive costs, except in certain goal states that are cost-free and absorbing. If these goal states are reachable from all the other states with positive probability, the set of optimal policies is well defined.

The MDPs above are reward-based. AO* is used normally in a cost setting, where rewards $r(a, s)$ are replaced by costs $c(a, s)$, and maximization is replaced by minimization. The two points of view are equivalent, and we will use one or the other when most convenient.

We are interested in the problem of selecting the action to do in the current state of a given infinite horizon MDPs, whether discounted or undiscounted. This will be achieved by a lookahead that uses a limited time window to run an anytime optimal algorithm over the version of the MDP that results from fixing the horizon.

AO*

A finite horizon MDP defines an *implicit AND/OR graph* that can be solved by the well-known AO* algorithm (**?**). The root node of this graph is the pair (s_0, H) where s_0 is the initial state and H is the horizon, while the terminal nodes are of the form (s, d) where s is a state and d is 0, or s is a terminal state (goal or dead-end). The children of a non-terminal node (s, d) are the triplets (a, s, d) where a is an action in $A(s)$, while the children of a node (a, s, d) are the nodes $(s', d-1)$ where s' is a possible successor state of a in s; i.e., $P_a(s'|s) > 0$. Non-terminal nodes of the form (s, d) are OR-nodes where an action needs to be selected, while nodes (a, s, d) are non-terminal AND-nodes.

The solutions graphs to these AND/OR graphs are defined in the standard way; they include the root node (s_0, H), and recursively, one child of every OR-node and all children of every AND-node. The value of a solution is defined recursively: the leaves have value equal to 0, the OR-nodes (s, d) have value equal to the value of the selected child, and the AND-nodes (a, s, d) have value equal to the cost of the action a in s plus the sum of the values for the children $(s', d-1)$ weighted by their probabilities $P_a(s'|s)$. The *optimal solution graph* yields a minimum value to the root node (cost setting), and can be computed as part of the evaluation procedure, marking the best child (min cost child) for each OR-node. This procedure is called *backward induction*. The algorithm AO* computes an optimal solution graph in a more selective and incremental manner, using a heuristic function h over the nodes of the graph that is *admissible* or *optimistic* (does not overestimate in the cost setting).

AO* maintains a graph G that *explicates* part of the implicit AND/OR graph, the *explicit graph*, and a second graph G^*, the *best partial solution*, that represents an optimal solution of G under the assumption that the tips n of G are the terminal nodes with values given by the heuristic $h(n)$. Initially, G contains the root node of the implicit graph only, and G^* is G. Then, iteratively, a non-terminal leaf node is selected from the best partial solution G^*, and the children of this node in the implicit graph are explicated in G. The best partial solution of G is then revised by applying an incremental form of backward induction, setting the values of the leaves in G to their heuristic values. The procedure finishes when there are no leaf nodes in the best partial graph G^*. If the heuristic values are optimistic, the best partial solution G^* is an optimal solution to the implicit AND/OR graph, which is partially explicated in G. In the best case, G ends up containing no more nodes than those in the best solution graph; in the worst case, G ends up explicating all the nodes in the implicit graph. Code for AO* is shown in Fig. 1. The choice of which non-terminal leaf node in the best partial graph to expand is important for performance but is not part of the algorithm and it does not affect its optimality.

UCT

UCT has some of the flavour of AO* and it is often presented as a best-first search algorithm too.[3] Indeed, UCT maintains an explicit partial graph that is expanded incre-

[3]This is technically wrong though, as a best-first algorithm just expands nodes in the best partial solution in correspondence to what is called 'exploitation'. Yet UCT as the other MCTS algorithms, and Anytime AO* below, do 'exploration' as well.

AO*: G is explicit graph, initially empty; h is heuristic function.

Initialization

1. Insert node (s, H) in G where s is the initial state
2. Initialize $V(s, H) := h(s, H)$
3. Initialize best partial graph to G

Loop

4. Select non-terminal tip node (s, d) in best partial graph. If there is no such node, **Exit**.
5. Expand node (s, d): for each $a \in A(s)$, add node (a, s, d) as child of (s, d), and for each s' with $P_a(s'|s) > 0$, add node $(s', d-1)$ as child of (a, s, d). Initialize values $V(s', d-1)$ for new nodes $(s', d-1)$ as 0 if $d - 1 = 0$, or $h(s', d-1)$ otherwise.
6. Update ancestors AND and OR nodes of (s, d) in G, bottom-up as:

$$Q(a, s, d) := c(a, s) + \gamma \textstyle\sum_{s'} P_a(s'|s) V(s', d-1),$$
$$V(s, d) := \min_{a \in A(s)} Q(a, s, d).$$

7. Mark best action in ancestor OR-nodes (s, d) to any action a such that $V(s, d) = Q(a, s, d)$, maintaining marked action if still best.
8. Recompute best partial graph by following marked actions.

Figure 1: AO* for Finite Horizon Cost-based MDPs.

UCT(s, d): G is explicit graph, initially empty; π is base policy; C is exploration constant.

1. If $d = 0$ or s is terminal, Return 0
2. If node (s, d) is not in explicit graph G, then
   - Add node (s, d) to explicit graph G
   - Initialize $N(s, d) := 0$ and $N(a, s, d) := 0$ for all $a \in A(s)$
   - Initialize $Q(a, s, d) := 0$ for all $a \in A(s)$
   - Obtain sampled accumulated discounted reward $r(\pi, s, d)$ by simulating base policy π for d steps starting at s
   - Return $r(\pi, s, d)$
3. If node (s, d) is in explicit graph G,
   - Let $Bonus(a) = C\sqrt{2 \log N(s, d)/N(a, s, d)}$ if $N(a, s, d) > 0$, else ∞, for each $a \in A(s)$
   - Select action $a = \text{argmax}_{a \in A(s)}[Q(a, s, d) + Bonus(a)]$
   - Sample state s' with probability $P_a(s'|s)$
   - Let $nv = r(s, a) + \gamma$UCT$(s', d-1)$
   - Increment $N(s, d)$ and $N(a, s, d)$
   - Set $Q(a, s, d) := Q(a, s, d) + [nv - Q(a, s, d)]/N(a, s, d)$
   - Return nv

Figure 2: UCT for Finite-Horizon Reward-based MDPs.

mentally like the graph G in AO*. The main differences are in how leaf nodes of this graph are selected and expanded, and how heuristic values for leafs are obtained and propagated. The code for UCT, which is more naturally described as a recursive algorithm, is shown in Fig. 2.

UCT consists of a sequence of stochastic simulations, like RTDP, that start at the root node. However, while the choice of successor states is stochastic, the choice of the actions is not greedy on the action Q-values as in RTDP, but greedy on the sum of the action Q-values and a bonus term $C\sqrt{2 \log N(s, d)/N(a, s, d)}$ that ensures that all applicable actions are tried in all states infinitely often at suitable rates. Here, C is an exploration constant, and $N(s, d)$ and $N(a, s, d)$ are counters that track the number of simulations that had passed through the node (s, d) and the number of times that action a has been selected at such node.

The counters $N(a, d)$ and $N(a, s, d)$ are maintained for the nodes in the explicit graph that is extended incrementally, starting like in AO*, with the single root node (s, H). The simulations start at the root and terminate at a terminal node or at the first node (s, d) that is not in the graph. In between, UCT selects an action a that is greedy on the stored value $Q(a, s, d)$ plus the bonus term, samples the next state s' with probability $P_a(s'|s)$, increments the counters $N(s, d)$ and $N(a, s, d)$, and generates the node $(s', d-1)$. When a node (s, d) is generated that is not in the explicit graph, the node is added to the explicit graph, the registers $N(s, d)$, $N(a, s, d)$, and $Q(a, s, d)$ are allocated and initialized to 0, and a total discounted reward $r(\pi, s, d)$ is sampled, by simulating a base policy π for d steps starting at s, and propagated upward along the nodes in the simulated path. These values are not propagated using full Bellman backups as in AO* or Value Iteration, but through Monte-Carlo backups that extend the current average with a new sampled value (**?**); see Fig. 2 for details.

It can be shown that UCT eventually explicates the whole finite horizon MDP graph, and that it converges to the optimal policy asymptotically. Unlike AO*, however, UCT does not have a termination condition, and moreover, UCT does not necessarily add a new node to the graph in every iteration, even when there are such nodes to explicate. Indeed, while in the worst case, AO* converges in a number of iterations that is bounded by the number of nodes in the implicit graph, UCT may require an exponential number of iterations (**?**; **?**). On the other hand, AO* is a *model-based approach* to planning that assumes that the probabilities and costs are known, while UCT is a *simulation-based approach* that just requires a simulator that can be reset to the current state.

The differences between UCT and AO* are in the leafs of the graphs selected for expansion, the way they are expanded, the values to which they are initialized, and how the values are propagated. These dimensions are the ones used to define a family of Monte Carlo Tree Search (MCTS) methods that includes UCT as the best known member (**?**). The feature that is common to this family of methods is the use of Monte Carlo simulations to evaluate the leafs of the graph. The resulting values are heuristic, although not necessarily optimistic as required for the optimality of AO*. One way to bridge the gap between AO* and MCTS methods is by modifying AO* to accommodate *non-admissible heuristics*, and moreover, *random heuristics that can be sampled* such as the cost of a base policy.

Anytime AO*

Anytime AO* involves two small changes from AO*. The first, shown in Fig. 3, is for handling non-admissible heuristics: rather than always selecting a non-terminal tip node from the explicit graph that is IN the *best partial graph*, Anytime AO* selects with probability p a non-terminal tip node from the explicit graph that is OUT of the best partial graph. The probability p is a given parameter between 0 and

Anytime AO* with possibly non-admissible heuristic h: same code as AO* except for extra parameter p, $0 \leq p \leq 1$, and line 4 in Loop for tip selection replaced by 4.1 and 4.2 below.

4.1. Choose IN with probability $1 - p$, else Choose OUT:

- IN: Select non-terminal tip node (s, d) from explicit graph G that is IN the best partial graph.
- OUT: Select non-terminal tip node (s, d) from explicit graph G that is NOT in the best partial graph.

4.2. If there is no node (s, d) satisfying the choice condition, make the other choice. If there is no node satisfying either choice condition, **Exit** the loop.

Figure 3: Anytime AO*. If the heuristic $h(s, d)$ is random, as when representing the cost of a given base policy; see text.

1, by default $1/2$. Of course, if a choice for an IN node is decided (a tip in the best partial graph) but there is no such node, then an OUT choice is forced, and vice versa. Anytime AO* terminates when neither IN or OUT choices are possible; i.e., when no tip nodes in the explicit graph are left, or when the time is up.

It is easy to see that Anytime AO* is optimal, whether the heuristic is admissible or not, because it terminates when the implicit graph has been fully explicated. In the worst case, the complexity of Anytime AO* is not worse than AO*, as AO* expands the complete graph in the worst case too.

The second change in Anytime AO*, not shown in the figure, is for dealing with random heuristics h. Basically, when the value $V(s, d)$ of a tip node (s, d) is set to a heuristic $h(s, d)$ that is a *random variable*, such as the reward obtained by following a base policy d steps from s, Anytime AO* uses *samples* of $h(s, d)$ until the node (s, d) is expanded. Until then, a 'fetch for value' $V(s, d)$, which occurs each time that a parent node of (s, d) is updated, results in a new sample of $h(s, d)$ which is averaged with the previous ones. This is implemented in standard fashion by incrementally updating the value $V(s, d)$ using a counter $N(s, d)$ and the new sample. These counters are no longer needed when the node (s, d) is expanded, as then the value of the node is given by the value of its children in the graph.

Choice of Tip Nodes in Anytime AO*

AO* and Anytime AO* leave open the criterion for selecting the tip node to expand. For the experiments, we use a selection criterion aimed at selecting the tip nodes that can have the biggest potential impact in the best partial graph. For this, a function $\Delta(n)$ is introduced that measures the change in the value of the node n that is needed in order to produce a change in the best partial graph. The function Δ is defined top-down over the explicit graph as:

1. For the root node, $\Delta(s, H) = \infty$.
2. For children $n_a = (a, s, d)$ of node (s, d) in the best solution graph, $\Delta(n_a)$ is $[V(n) - Q(n_a)]$ if a is *not* the best action in n, else is $\min_b[\Delta(n), Q(n_b) - V(n)]$, for $b \neq a$.
3. For children $n_a = (a, s, d)$ of node (s, d) that is *not* in the best solution graph, $\Delta(n_a)$ is $\Delta(n) + V(n) - Q(n_a)$.
4. For children $n_{s'} = (s', d - 1)$ of node $n_a = (a, s, d)$, $\Delta(n_{s'})$ is $\Delta(n_a)/\gamma P_a(s'|s)$.

The tip nodes (s, d) that are chosen for expansion are the ones that minimize the values $|\Delta(s, d)|$. Since this computation is expensive, as it involves a complete traversal of the explicit graph G, we select $N \geq 1$ tip nodes for expansion at a time. This selection is implemented by using two priority queues during the graph traversal: one for selecting the best N tips in the solution graph (IN), and one for selecting the best N tips OUT. The first selected tip is the one with min $|\Delta(s, d)|$ in the IN queue with probability $1 - p$ and in the OUT queue otherwise. Once selected, the tip is removed from the queue, and the process is repeated N times.

Experimental Results

We have evaluated Anytime AO* (abbreviated AOT) vs. UCT as an action selection mechanism over a number of MDPs. In each case, we run the planning algorithm for a number of iterations from the current state s, apply the best action according to the resulting $Q(a, s, H)$ values, where H is the planning horizon, and repeat the loop from the state that results until the goal is reached. The *quality profile* of the algorithms is given by the average cost to the goal as a function of the time window for action selection. Each data point in the profiles (and table) is the average over 1,000 sampled episodes that finish when the goal is reached or after 100 steps. For the profiles, the x-axis stands for the average time per action selected, and the y-axis for the average cost to the goal. AOT was run with parameter $p = 0.5$ (i.e., equal probability for IN and OUT choices), and UCT with the exploration 'constant' C set to current $Q(a, s, d)$ value of the node; a choice that appears to be standard (**?**; **?**). The N parameter for the computation of the Δ's was set as a fraction k of the number of iterations, $k = 1/10$.[4] The actual codes for UCT and AOT differ from the ones shown in that they deal with graphs instead of trees; i.e., there are duplicates (s, d) with the same state s and depth d. The same applies to the computation of the Δ's. All experiments were run on Xeon 'Woodcrest' computers of 2.33 GHz and 8 Gb of RAM.

CTP. The Canadian Traveller Problem is a path finding problem over a graph whose edges (roads) may be blocked (**?**). Each edge is blocked with a given prior probability, and the status of an edge (blocked or free) can be sensed noise-free from either end of the edge. The problem is an acyclic POMDP that results in a belief MDP whose states are given by the agent position, and edge beliefs that can take 3 values: prior, known to be blocked, and known to be free. If the number of nodes is n and the number of edges in the graph is m, the number of MDP states is $n \times 3^m$. The problem has been addressed recently using a domain-specific implementation of UCT (**?**). We consider instances with 20 nodes from that paper and instances with 10 nodes obtained from

[4] The number of tip nodes in UCT is bounded by the number of iterations (rollouts). In AOT, the number of tips is bounded in the worst case by the number of iterations (expansions) multiplied by $|A||S|$, where $|A|$ and $|S|$ are the number of actions and states.

		br. factor		UCT–CTP		random base policy			optimistic base policy		
prob.	P(bad)	avg	max	UCTB	UCTO	direct	UCT	AOT	direct	UCT	AOT
10-1	19.9	8.5	32	114.8 ± 3	$\mathbf{101.3 \pm 3}$	324.7 ± 5	108.4 ± 1	103.1 ± 1	102.8 ± 1	106.1 ± 1	102.2 ± 1
10-2	45.6	6.5	64	102.5 ± 2	99.9 ± 2	254.8 ± 4	101.1 ± 1	98.3 ± 1	145.9 ± 2	$\mathbf{97.9 \pm 1}$	99.1 ± 1
10-3	21.9	8.9	16	127.2 ± 3	$\mathbf{115.2 \pm 3}$	313.2 ± 4	127.1 ± 1	125.3 ± 1	125.0 ± 1	118.7 ± 1	116.5 ± 1
10-4	1.4	11.4	32	58.0 ± 2	53.4 ± 2	276.4 ± 5	55.9 ± 1	54.4 ± 1	53.0 ± 1	53.6 ± 1	$\mathbf{52.5 \pm 1}$
10-5	22.7	7.2	16	89.3 ± 2	$\mathbf{86.1 \pm 2}$	224.4 ± 3	91.3 ± 1	90.0 ± 1	86.7 ± 1	89.3 ± 1	94.4 ± 1
10-6	24.9	7.1	32	111.5 ± 3	$\mathbf{92.6 \pm 2}$	225.4 ± 3	121.4 ± 1	102.9 ± 1	105.0 ± 1	105.4 ± 1	96.0 ± 1
10-7	4.1	11.0	32	83.3 ± 2	$\mathbf{66.7 \pm 1}$	244.8 ± 4	83.9 ± 1	69.9 ± 1	118.4 ± 2	74.5 ± 0	69.0 ± 0
10-8	14.1	8.0	32	87.0 ± 1	$\mathbf{74.6 \pm 1}$	230.9 ± 3	107.1 ± 1	78.3 ± 0	75.0 ± 0	81.8 ± 0	76.1 ± 0
10-9	28.1	12.7	32	72.0 ± 3	66.5 ± 2	177.6 ± 4	71.5 ± 1	66.7 ± 1	$\mathbf{63.7 \pm 1}$	68.1 ± 1	66.7 ± 1
10-10	31.1	9.4	32	76.7 ± 2	75.4 ± 2	200.0 ± 4	79.5 ± 1	$\mathbf{74.5 \pm 1}$	76.9 ± 1	75.2 ± 1	82.7 ± 1
total				922.3	**831.7**	2472.2	947.2	863.4	952.4	870.6	855.2
20-1	17.9	13.5	128	210.7 ± 7	169.0 ± 6	1000.3 ± 1	216.4 ± 3	187.4 ± 3	191.8 ± 0	180.7 ± 3	$\mathbf{163.8 \pm 2}$
20-2	9.5	15.7	64	176.4 ± 4	$\mathbf{148.9 \pm 3}$	676.6 ± 1	178.5 ± 2	167.4 ± 2	202.7 ± 0	160.8 ± 2	156.4 ± 1
20-3	14.3	15.2	128	150.7 ± 7	$\mathbf{132.5 \pm 6}$	571.7 ± 1	169.7 ± 4	140.7 ± 3	142.1 ± 0	144.3 ± 3	133.8 ± 2
20-4	78.6	11.4	64	264.8 ± 9	235.2 ± 7	861.4 ± 1	264.1 ± 4	261.0 ± 4	267.9 ± 0	238.3 ± 3	$\mathbf{233.4 \pm 3}$
20-5	20.4	15.0	64	123.2 ± 7	111.3 ± 5	586.2 ± 1	139.8 ± 4	128.3 ± 3	163.1 ± 0	123.9 ± 3	$\mathbf{109.4 \pm 2}$
20-6	14.4	13.9	64	165.4 ± 6	$\mathbf{133.1 \pm 3}$	670.7 ± 1	178.0 ± 3	160.0 ± 2	193.5 ± 0	167.8 ± 2	135.5 ± 1
20-7	8.4	14.3	128	191.6 ± 6	148.2 ± 4	862.9 ± 1	211.8 ± 3	170.2 ± 2	171.3 ± 0	174.1 ± 2	$\mathbf{145.1 \pm 1}$
20-8	23.3	15.0	64	160.1 ± 7	$\mathbf{134.5 \pm 5}$	704.0 ± 1	218.5 ± 4	154.6 ± 3	167.9 ± 0	152.3 ± 3	135.9 ± 2
20-9	33.0	14.6	128	235.2 ± 6	173.9 ± 4	783.4 ± 1	251.9 ± 3	213.7 ± 2	212.8 ± 0	185.2 ± 2	$\mathbf{173.3 \pm 1}$
20-10	12.1	15.3	64	180.8 ± 7	167.0 ± 5	825.7 ± 1	185.7 ± 3	180.5 ± 3	173.2 ± 0	178.5 ± 3	$\mathbf{166.4 \pm 2}$
total				1858.9	1553.6	7542.9	2014.4	1763.8	1886.3	1705.9	**1553.0**

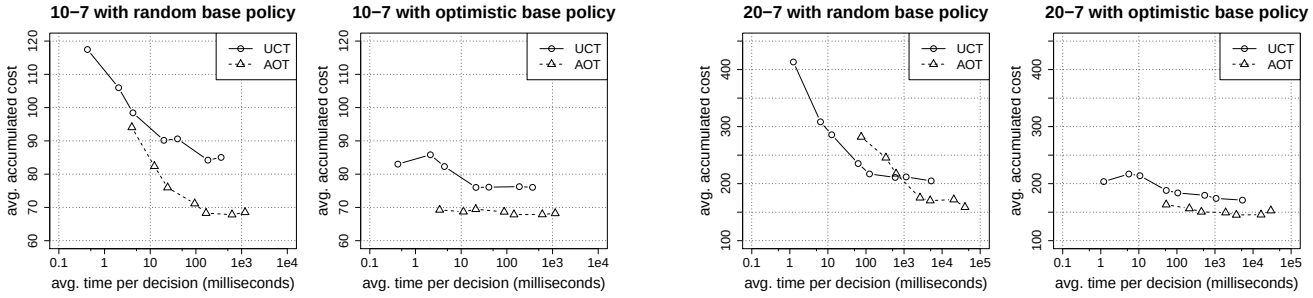
Table 1: CTP instances with 10 and 20 nodes. P(bad) is probability (in percentage) of the instance not being solvable, and max branching factor is 2^m where $m+1$ is the number of edges with a common node in CTP graph. UCTB and UCTO are two domain-dependent UCT implementations (**?**), π refers to the base policy in our domain-independent UCT and AOT algorithms, whose base performance is shown as well. UCT run for 10,000 iterations and AOT for 1,000. Boldface figures are best in whole table; gray cells show best among domain-independent implementations.

the authors. Following Eyerich, Keller and Helmert, the actions are identified with moves to any node in the frontier of the known graph. The horizon H can then be set to the number of nodes in the graph.

Quality profiles are shown in Fig. 4 for instances 10-7 and 20-7. On each panel, results on the left are for UCT and AOT using the base *random policy*, while results on the right are for the base *optimistic policy* that assumes that all edges of unknown status are traversable. The points for UCT correspond to running 10, 50, 100, 500, 1k, 5k, 10k and 50k iterations (rollouts), resulting in the times shown. The points for AOT, correspond to running 10, 50, 100, 500, 1k, 5k, 10k iterations (expansions). Points that time out are not displayed. The curves show that AOT performs better than UCT except up to the time window of 1 second in the instance 20-7 for the random base policy. We have computed the curves for the 20 instances and the pattern is similar (but not shown for lack of space). A comprehensive view of the results is shown in Table 1. As a reference, we include also the results for two specialized implementations of UCT, UCTB and UCTO (**?**), that take advantage of the specific MDP structure of the CTP, use a more informed base policy, and actually solve a slightly simpler version of the problem where the given CTP is solvable. While all the algorithms are evaluated over solvable instances, AOT and our UCT solve the harder problem of getting to the target location if the problem is solvable, and determining otherwise that the problem is unsolvable. In some instances, the edge priors are such that the probability of an instance not being solvable is high. This is shown as the probability of 'bad weather' in the second column of the table. In spite of these differences, AOT improves upon the domain-specific implementation of UCT in several cases, and practically dominates the domain-independent version. Since UCTO is the state of the art in CTP, the results show that our domain-independent implementations of both AOT and UCT are good enough, and that AOT in particular appears to be competitive with the state of the art in this domain.

Sailing and Racetrack. The Sailing domain is similar to the one in the original UCT paper (**?**). The profile for a 100×100 instance with 80,000 states is shown in the left panel of Fig. 5 for a random base policy. The problem has a discount $\gamma = 0.95$ and the optimal value is 26.08. UCT and AOT are run with horizon $H = 50$. Profiles for other instances show the same pattern: AOT is slower to get started because of the more expensive expansions, but then learns faster. The profile for the instance 'barto-big' of the Racetrack domain (**?**) is shown in the right panel of Fig. 5. In this case, AOT converges much faster than UCT.

Base Policies, Heuristics, and RTDP. We also performed experiments comparing the use of heuristics h vs. base policies π for initializing the value of tip nodes in AOT. We refer to AOT using π as a base policy as AOT(π), and to AOT using the heuristic h as AOT(h). Clearly, the overhead per iteration is smaller in AOT(h) than in AOT(π) that must do a full rollout of π to evaluate a tip node. Results of AOT(π) vs. AOT(h) on the instances 20-1 and 20-4 of CTP are shown in Fig. 6 with both the zero heuristic and the min-min heuristic (**?**). The base policy π is the policy π_h that is greedy with



(a) 10-7 (b) 20-7

Figure 4: Quality profiles for CTP instances 10-7 and 20-7.

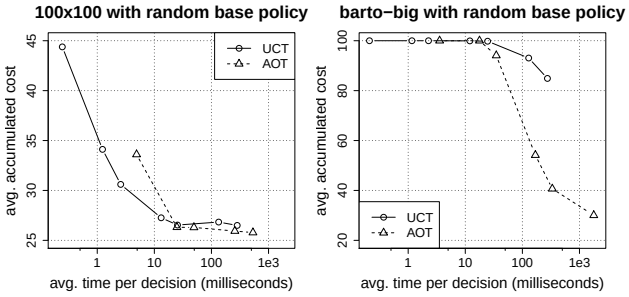


Figure 5: Quality profiles for Sailing and Racetrack.

respect to h with π_h being the random policy when $h = 0$. The curves also show the performance of LRTDP (**?**), used as an *anytime action selection mechanism* that is run over the same finite-horizon MDP as AOT and with the same heuristic h. Interestingly, when LRTDP is used in this form, as opposed to an *off-line* planning algorithm, LRTDP does rather well on CTP too, where there is no clear dominance between AOT(π_h), AOT(h), and LRTDP(h). The curves for the instance 20-1 are rather typical of the CTP instances. For the zero heuristic, AOT(h) does better than LRTDP(h) which does better than AOT(π_h). On the other hand, for the min-min heuristic, the ranking is reversed but with differences in performance being smaller. There are some exceptions to this pattern where AOT(π_h) does better, and even much better than both AOT(h) and LRTDP(h) for the two heuristics. One such instance, 20-4, is shown in the bottom part of Fig. 6.

Figure 7 shows the same comparison for a Racetrack instance and three variations h_d of the min-min heuristic h_{min}, $h_d = d \times h_{min}$ for $d = 2, 1, 1/2$. Notice that multiplying an heuristic h by a constant d has no effect on the policy π_h that is greedy with respect to h, and hence no effect on AOT(π_{h_d}). On the other hand, these changes affect both AOT(h_d) and LRTDP(h_d). As expected the performance of LRTDP(h_d) deteriorates for $d < 1$, but somewhat surprisingly, improves for $d > 1$ even as the resulting heuristic is not (necessarily) admissible. For small time windows, AOT(h_d) does worse than LRTDP(h_d), but for larger windows AOT(h_d) catches up, and in two cases surpasses LRTDP(h_d). In these cases, AOT with the greedy base policy π_h appears to do best of all.

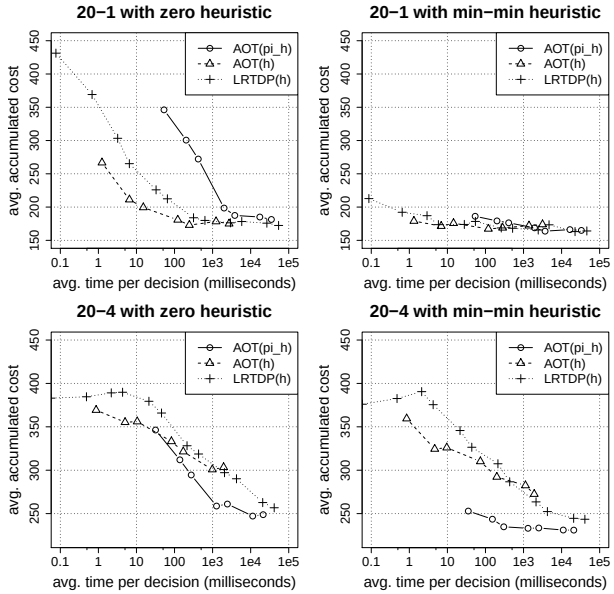


Figure 6: AOT(π_h), AOT(h), and LRTDP(h) on two CTP instances for $h = 0$ and h_{min}. Policy π_h is greedy in h.

Leaf Selection. We have also tested the value of the *leaf selection method* used in AOT by comparing the Δ-based selection of tips vs. random tip selection. It turns out that the former pays off, in particular, when the base policies are not informed. Some results are shown in Fig. 8.

The results obtained from the experiments above are not conclusive. They show however that the new AOT algorithm, while a simple variation of the standard AO* algorithm, appears to be competitive with both UCT and LRTDP, while producing state-of-the-art results in the CTP domain.

Conclusions

The algorithm UCT addresses the problem of anytime action selection over MDPs and related models, combining a non-exhaustive search with the ability to use informed base policies. In this work, we have developed a new algorithm for this task and showed that it compares well with UCT. The new algorithm, Anytime AO* (AOT) is a very small variation of AO* that retains the optimality of AO* and its worst

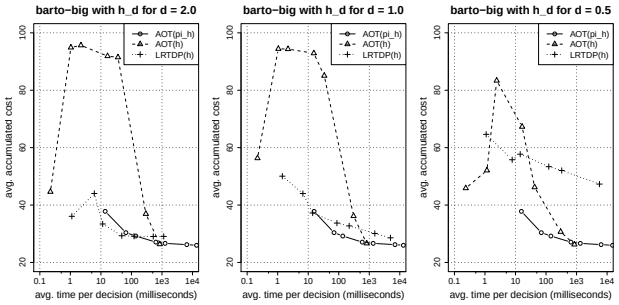


Figure 7: AOT(π_h), AOT(h), and LRTDP(h) on Racetrack instance for $h = d \times h_{min}$ and $d = 2, 1, 1/2$.

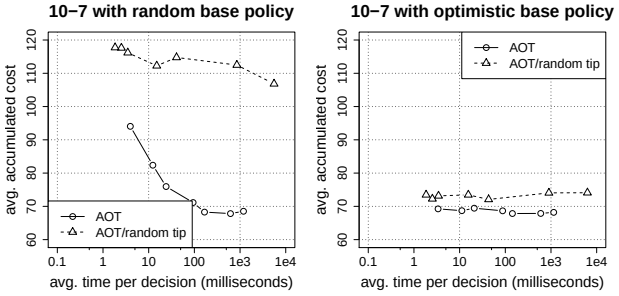


Figure 8: Random vs. Δ-based tip selection. Problem 10-7.

case complexity, yet it does not require admissible heuristics and can use base policies. The work helps to bridge the gap between Monte-Carlo Tree Search (MCTS) methods and *anytime* heuristic search methods, both of which have flourished in recent years, the former over AND/OR Graphs and Game Trees, the latter over OR graphs. The relation of Anytime AO* to both classes of methods suggest also a number of extensions that are worth exploring in the future.

Acknowledgments. H. Geffner is partially supported by grants TIN2009-10232, MICINN, Spain, and EC-7PM-SpaceBook.

Molestiae voluptatum veritatis, alias quisquam ex culpa hic quia sequi sed eveniet, deserunt esse distinctio voluptate cum laboriosam similique quo expedita, aut voluptatibus id vel, temporibus itaque alias?Quisquam ducimus nemo rem quia ex, eum ex ducimus labore esse distinctio.Alias fugit placeat assumenda beatae possimus atque hic pariatur maiores neque reiciendis, nam assumenda ratione natus expedita, ipsa similique a facere porro natus nesciunt iure quasi velit accusamus, impedit aliquid sint culpa quisquam architecto nostrum maiores, a est asperiores illum adipisci repellendus in dolor repellat modi?Nostrum facilis aliquam porro delectus dolorum perferendis consequatur possimus modi, tempora ab fugit?Ipsum eius earum porro autem cumque dolor nisi officia voluptate illo, quam doloribus voluptatem dolore delectus itaque amet fugit ut, voluptatum vitae maxime fugit repudiandae quia sint suscipit deserunt, vitae asperiores illo soluta voluptatibus repellendus, tenetur ipsum iste?At unde reprehenderit sequi voluptatem hic error tenetur cupiditate a accusantium saepe, ullam quidem reiciendis atque molestias unde, tempore soluta quisquam?Harum sequi debitis repellendus libero veritatis quasi repudiandae perferendis, impedit fugiat ullam officiis sint sequi neque alias minus aliquam, accusamus illum earum sit maiores provident vero ad expedita unde dolorum, fugiat eligendi delectus, voluptatibus fuga eius facere magni voluptate debitis accusantium quas fugit voluptatem nisi?Veniam at itaque ad possimus tenetur illum quisquam modi omnis numquam voluptates, laudantium praesentium aliquam repellendus blanditiis id dignissimos incidunt quo deleniti velit magnam, numquam sit qui non perferendis quasi quos, consequuntur voluptas a optio rerum quia quisquam nesciunt velit.Ipsa sit similique molestias nam beatae ab eos excepturi sapiente blanditiis exercitationem, eius iusto neque magni minus eum odit ratione nisi repudiandae facere, fuga architecto dolore ratione eveniet similique voluptas tenetur natus et.Ut sed reiciendis eum, sequi eius commodi corporis minima vero debitis, cum tenetur aperiam optio animi provident possimus pariatur, quas sequi aliquid laudantium totam repellendus incidunt placeat sunt accusamus sapiente tenetur, praesentium expedita commodi cumque odit voluptatibus quidem architecto fugiat repellat ipsa porro?Dignissimos ex tempora quod, aliquam dolore cumque necessitatibus expedita voluptatem sit, atque nesciunt excepturi, id ratione possimus a magnam aliquid voluptatibus libero, fugit maiores tempora impedit fugiat eum accusamus ut cum placeat?Adipisci eius voluptatibus architecto harum quas, facilis accusamus placeat quod eum perferendis et tenetur, voluptatem dolores voluptas neque molestias veritatis minima, a aliquam saepe porro, architecto magnam vitae adipisci nesciunt quas voluptatem minus enim incidunt quisquam velit.Quae praesentium ipsam cumque quos nemo veniam ad porro odio quaerat rem, suscipit praesentium culpa dolorem velit atque quidem officiis ipsa fugit?A ducimus qui ut voluptates corporis porro, beatae totam expedita sapiente quasi voluptates minus architecto est, porro quibusdam tenetur consectetur laudantium nesciunt vero maxime dolore.Similique ullam iste adipisci beatae ipsam sed laborum doloremque quo fuga provident, tempora quod deserunt totam rem nulla consequatur obcaecati libero dolor unde quaerat, necessitatibus sit nihil fugiat molestias ut excepturi quos quibusdam laborum error, numquam dolorem architecto sit illo laboriosam harum enim, numquam laborum iusto quod voluptas placeat dolorum dicta saepe quibusdam odit.Atque perspiciatis commodi harum illum ullam tempore aliquam molestias possimus optio tenetur, dolorum natus eveniet a incidunt omnis officiis, error illum rem.Laborum ex ea laboriosam sapiente hic esse nemo rem incidunt ad, commodi inventore voluptatem minus dolor voluptate similique modi est quaerat delectus excepturi, aliquam consectetur suscipit doloribus recusandae porro dolore placeat vero eum quaerat.Nisi fugiat nobis suscipit, asperiores quo aliquam eos minima?Alias facilis repellendus exercitationem totam aperiam et rem, obcaecati dolor nisi exercitationem sit magni distinctio provident unde, sapiente a officia quidem impedit est molestiae fuga.Commodi explicabo nostrum eligendi quis quam harum eius amet quisquam quos modi, earum et ea provident autem quaerat eius, nostrum repudiandae quisquam veniam eaque perspiciatis voluptate ab amet nam veritatis, ullam incidunt dicta esse quas autem facilis

at maxime magni, nihil at quidem.Fuga facere quisquam soluta, optio modi veritatis accusantium, qui ex delectus suscipit eveniet in