CLEI Electronic Journal, Volume 27, Issue 3, paper 5, August 2024

Monte Carlo Tree Search Algorithm for SSPs Under the GUBS Criterion

Gabriel Nunes Crispino, Valdinei Freire, Karina Valdivia Delgado
Universidade de São Paulo,
São Paulo, Brasil,
gcrispino@alumni.usp.br, {valdinei.freire@, kvd@}usp.br

Abstract

The Stochastic Shortest Path (SSP) is a formalism widely used for modeling goal-oriented probabilistic planning problems. When dead ends, which are states from which goal states cannot be reached, are present in the problem and cannot be avoided, the standard criterion for solving SSPs is not well defined in these scenarios. Because of that, several alternate criteria for solving SSPs with unavoidable dead ends have been proposed in the literature. One of these criteria is GUBS (Goals with Utility-Based Semantics), a criterion that makes trade-offs between probability-to-goal and cost by combining goal prioritization with Expected Utility Theory. GUBS is a good choice for these problems because it is one of the only criteria that are known to maintain the α-strong probability-to-goal priority property, a property that provides guarantees on how a decision criterion can choose policies without having to preprocess any specific SSP problem. Although there already exist two exact algorithms for solving GUBS, eGUBS-VI and eGUBS-AO*, both are offline and there is no algorithm for solving GUBS in an online manner. In this paper we propose UCT-GUBS, an online approximate algorithm based on UCT (a Monte Carlo tree search algorithm) that solves SSPs under the GUBS criterion. We provide an analysis of an empirical evaluation performed on three probabilistic planning domains (Navigation, River, and Triangle Tireworld) to observe how the probability-to-goal and utility values of the resulting policies compare to the optimal values, and also how the time performance of UCT-GUBS compares to the ones of eGUBS-VI and eGUBS-AO*. Our conclusion is that, like other algorithms, the usage of UCT-GUBS has to be evaluated considering the application requirements and of the problem being solved. Depending on these factors, it can be a good alternative for obtaining policies in an online fashion while, for some problems, also being able to have better time performance than other algorithms.

Keywords: Markov Decision Processes, Stochastic Shortest Path, Sequential decision making, Probabilistic planning, Monte Carlo tree search.

1 Introduction

In the field of probabilistic planning, the Stochastic Shortest Path (SSP) [1] formalism is widely used to model goal-oriented problems with stochastic actions. In these problems, an agent is at a specific state at each time step, and can then apply a set of stochastic actions that will transition the agent to a next state. The transition to the next state happens according to a probability distribution based on the current state, the executed action and the next state, and there is also a cost associated apply this action at the current state. A non-empty subset of states that are absorbing goal states is assumed to exist.

In the standard criterion for solving SSPs, the goal is to find a policy, a mapping from states to actions, that has the minimum expected cost to a goal state when executed. If it is not possible to reach a goal state from a certain state s, we say that s is a dead end. If dead ends exist in the SSP and they are unavoidable, the expected cost to goal diverges. This means that, in the standard criterion for solving SSPs, it is not possible to solve these problems when these states exist and are unavoidable. For this reason, several alternate criteria have been proposed to solve SSPs with unavoidable dead ends [2, 3, 4, 5, 6, 7].

Figure 1 depicts an example of an SSP with an unavoidable dead end. In this example, the agent represents a person that needs to get from their home at 11 AM to the airport at 11 PM. There is one goal

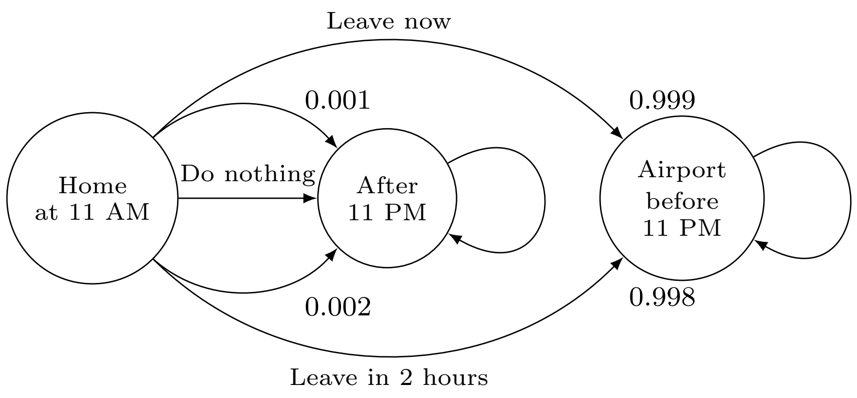


Figure 1: Example of an SSP with an unavoidable dead end: the agent is a person that needs to arrive at the airport at 23h, and there is a dead end representing the case in which the person arrives at the airport after this time.

state, representing the case where the person is at the airport before 11 PM, and one dead-end state, which represents the person not being in the airport after 11 PM. At the initial state, there are 3 actions: to leave now (top one) with probability 0.999 of arriving on time, or to leave in 2 hours (bottom one), which has probability 0.998 of arriving on time. The dead end is unavoidable because all possible histories in this SSP have a non-zero probability of reaching this state.

The GUBS (Goals with Utility-Based Semantics) criterion [8, 9] is one of the criteria that solve SSPs with unavoidable dead ends. It combines Expected Utility Theory with goal prioritization, and it is one of the two criteria known to maintain the α-strong probability-to-goal priority property [9] for every value of $0 \leq \alpha \leq 1$. Maintaining this guarantee means that, for every SSP problem and value of $0 \leq \alpha \leq 1$, under certain conditions, GUBS guarantees that the ratio of the probability-to-goal value of this policy and the one of any other policy is always greater than α. This means that we could tune the parameters of GUBS such that we have a certain value of α, which will dictate which policies could be optimal in different problems. Considering the example of Figure 1, this would mean that, by tuning GUBS' parameters for different values of α, we can get any of the three available actions as optimal. For example, by choosing parameters such that $\alpha > 0$, we can guarantee that, for this problem, the action that does nothing will never be chosen as optimal, since it has a probability-to-goal value of 0.

There are two exact algorithms proposed in the literature for solving GUBS: eGUBS-VI [8, 9], a value iteration algorithm, and eGUBS-AO*[9], that uses value iteration with heuristic search. Having exact algorithms is important and mandatory to obtain optimal solutions to the problem. However, since current exact algorithms to solve GUBS are offline, that is, they need to first compute a policy defined for the initial states and all states reachable from it before being able to make a decision, they can not be suited for certain applications. This is especially true if planning needs to happen with a time constraint and if we can compromise quality in favor of less planning time. In these cases having available online algorithms can be useful, as they can, at each step of acting in an environment, do its processing for a given time and then choose an action with the best estimate given the processing done in this time frame.

For this reason, in the present work we propose UCT-GUBS (Upper Confidence Bounds Applied to Trees Under GUBS), an algorithm based on the UCT algorithm [10] that uses Monte Carlo tree search to solve GUBS in an online and approximate manner. We perform experiments with it and compare to eGUBS-VI and eGUBS-AO*, and also compare how much some of the values obtained from UCT-GUBS compare to optimal values.

This work is structured as follows: Section 2 outlines the formalism for Stochastic Shortest Path problems used throughout the paper; Section 3 contains the definition of the GUBS criterion; Section 4 introduces the UCT-GUBS algorithm; Section 5 explains the empirical evaluation done for UCT-GUBS and provides an analysis of the results that were obtained; Finally, Section 6 contains the conclusion of the present work.

This manuscript extends a preliminary version of this work that appeared in [11] with some corrections in the definition of UCT-GUBS and additional experiments. More specifically, the present work provides experiments in three domains instead of one, and analyzes more result metrics than the preliminary referenced work.

2 Stochastic Shortest Path (SSP) Problems

The Stochastic Shortest Path (SSP) model is commonly used to formalize probabilistic planning problems in which an agent is set in an environment and can interact with it by applying actions with stochastic outcomes. This model is defined as follows:

Definition 1. *An SSP [1] is a tuple $\mathcal{M} = \langle \mathcal{S}, s_0, \mathcal{A}, P, c, \mathcal{G} \rangle$, where:*

- *$\mathcal{S}$ is a finite set of states;*
- *$s_0 \in \mathcal{S}$ is the initial state;*
- *$\mathcal{A}$ is a finite set of actions;*
- *$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function, such that $P(s, a, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$;*
- *$c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ is the cost function, which assigns a cost $c(s, a)$ for taking action a at state s;*
- *$\mathcal{G} \subset \mathcal{S}$ is the set of absorbing goal states, i.e. $c(g, a) = 0$ and $P(g, a, g) = 1, \forall g \in \mathcal{G}$ and $a \in \mathcal{A}$. Also, $c(s, a) > 0$ for $s \in \mathcal{S} \setminus \mathcal{G}$.*

Thus, in an SSP, the agent is at time step t at state s_t. She can apply an action a_t by paying cost $c(s_t, a_t)$, leading to a new state s_{t+1} with probability $P(s_t, a_t, s_{t+1})$. The objective is to reach an absorbing goal state $s \in \mathcal{G}$.

The solution to an SSP is a policy π, a function that maps states to actions. This policy should minimize the expected cost to goal:

$$V^\pi(s) = \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=0}^{T-1} c_t \mid \pi, s_0 = s\right].$$

The probability-to-goal of a policy π at a state s is the probability of reaching a goal from s when following π, and is defined by the function:

$$P_G^\pi(s) = \lim_{t\to\infty} \Pr(s_t \in \mathcal{G} \mid \pi, s).$$

A policy is Markovian if its choice of an action depends on the current state only. If this is not true (for example, if the policy needs to maintain a history to choose an action) the policy is non-Markovian.

As in Figure 1, an SSP can be represented by a hypergraph, in which its nodes represent the states, and its hyperedges represent the actions with the transition probabilities. To aid in the exemplification, we will refer to the states "Home at 11 AM", "After 11 PM" and "Airport before 11 PM" as s_h, s_d, and s_g, and to the actions "Leave now", "Leave in 2 hours" and "Do nothing" as a_n, a_h, and a_d, respectively. Thus, in the example provided in Figure 1, the initial state s_0 is s_h; the set of states is $\mathcal{S} = \{s_h, s_d, s_g\}$; the set of actions is $\mathcal{A} = \{a_n, a_h, a_d\}$; the subset of goal states is $\mathcal{G} = \{s_g\}$; the cost function c is defined as $c(s, a) = 1, \forall s \in \mathcal{S}$ such that $s \neq s_g$ and $\forall a \in \mathcal{A}$, and $c(s_g, a) = 0, \forall s \in \mathcal{G}$ and $\forall a \in \mathcal{A}$; the transition function P for s_h is specified on Table 1, while for the other states it is defined as $P(s_d, a, s_d) = 1, \forall a \in \mathcal{A}$ and $P(s_g, a, s_g) = 1, \forall a \in \mathcal{A}$ (i.e. states s_d and s_g are both absorbing).

Next State / **Action**	s_h	s_d	s_g
a_n	0	0.001	0.999
a_h	0	0.002	0.998
a_d	0	1	0

Table 1: Transition probabilities from state s_h from the example SSP of Figure 1.

2.1 SSPs with dead ends

An SSP can have states such that the probability-to-goal from these states is 0. They are referred to as *dead-end* states. If they exist and are unavoidable, that is, no policy can be taken from the initial state such that a dead-end state is reached with probability equal to 0, the expected cost to goal $V^\pi(s_0)$ diverges, for any policy π. In other words, when dead ends are unavoidable, the conventional criterion of expected cost to goal in SSPs does not work anymore.

To solve SSPs in the presence of unavoidable dead ends, we then need alternate criteria that have cost measures that do not diverge in these cases. Several criteria were proposed in the literature for this matter. Some of them consider a lexicographic approach, in which a given cost measure is minimized considering policies that maximize probability-to-goal [4, 6]. In the example of Figure 1, these criteria would always choose the action of leaving now, which results in the maximum probability of reaching the goal. Other criteria use cost measures that trade off probability-to-goal and cost by evaluating both at the same time [3, 5, 7, 8, 9]. In the example, these criteria could choose any of the three available actions. This choice depends on each criterion's parameters used to make a trade-off between probability-to-goal and cost. From these criteria, only GUBS [8] and α-MCMP [7] are able to guarantee the α-strong probability-to-goal priority property [9]. This property is defined as follows:

Definition 2 (α-strong probability-to-goal priority [9]). *Consider $0 \leq \alpha \leq 1$, we say that a decision criterion has α-strong probability-to-goal priority if for all SSPs $\mathcal{M}$ and all pairs of policies $\pi, \pi' \in \Pi$, the following condition is true:*

$$\pi \succeq \pi' \implies \frac{P_G^{\pi}(s_0)}{P_G^{\pi'}(s_0)} \geq \alpha,$$

where $P_G^{\pi'}(s_0) > 0$. If π^* is an optimal policy then for all $\pi' \in \Pi$ the following equation holds:

$$\pi^* \succeq \pi' \implies \frac{P_G^{\pi^*}(s_0)}{P_G^{\pi'}(s_0)} \geq \alpha.$$

Note that $\frac{P_G^{\pi}(s_0)}{P_G^{\pi'}(s_0)} \geq \alpha$ is a necessary condition for $\pi \succeq \pi'$ to be true.

Lexicographic criteria have 1-strong probability-to-goal priority, since they guarantee that, considering that a policy π is preferred over another policy π', π necessarily has a larger value of probability-to-goal than π' (i.e. $\frac{P_G^{\pi}(s_0)}{P_G^{\pi'}(s_0)} \geq 1$) [9]. As mentioned, GUBS and α-MCMP guarantee α-strong priority for $0 \leq \alpha \leq 1$, while other trade-off criteria from the literature only have such guarantees for $\alpha = 0$ (only-0-strong) [9].

Considering the example at Figure 1, as already mentioned, 1-strong criteria would always choose the action that maximizes probability-to-goal. Criteria that are only-0-strong can only guarantee that a policy with probability-to-goal equal to 0, such as the one that chooses the action of doing nothing in the given example, can be chosen as optimal. In the case of GUBS, we can choose its parameters such that it is 0.998-strong, which would mean that it could choose either the action that leaves now to the airport (with probability-to-goal 0.999), or the action that leaves after two hours (with probability-to-goal 0.998), but not the action that does nothing (with probability-to-goal 0).

Next section will define and cover GUBS in more detail.

3 Goals with Utility-based Semantics (GUBS)

The GUBS criterion defines preferences of policies by combining goal prioritization over histories with Expected Utility Theory. It is formally defined as follows:

Definition 3 (**GUBS criterion** [8]). *The GUBS criterion evaluates a history by the utility function U:*

$$U(C_T, \beta_T) = u(C_T) + K_g \beta_T, \tag{1}$$

where C_T is the accumulated cost of the history at time T, and β_T is a variable that indicates whether a goal has been reached at time T. Formally, if $s_T \in \mathcal{G}$ then $C_T = \sum_{t=0}^{T-1} c(s_t, a_t)$ and $\beta_T = 1$, otherwise $C_T = \sum_{t=0}^{T-1} c(s_t, a_t)$ and $\beta_T = 0$.

An agent follows the GUBS criterion if she evaluates a policy π under the following value function:

$$V_{GUBS}^{\pi}(s) = \lim_{T \to \infty} \mathbb{E}[U(C_T, \beta_T) | \pi, s_0 = s]. \tag{2}$$

A policy π^* is optimal under the GUBS criterion if it maximizes the function V_{GUBS}^{π} for all $s \in \mathcal{S}$, that is, $V_{GUBS}^{\pi^*}(s) \geq V_{GUBS}^{\pi}(s), \forall \pi \in \Pi$ and $\forall s \in \mathcal{S}$. For simplicity, we also refer to $V_{GUBS}^{\pi^*}$ as V_{GUBS}^{*} in this work.

Two algorithms have been proposed in the literature to find optimal solutions to eGUBS, a specific case of GUBS in which the optimal function is an exponential function with a negative risk factor [12, 9]: eGUBS-VI and eGUBS-AO*. Both do this by computing a finite non-Markovian policy from a Markovian policy of a lexicographic risk-sensitive criterion, in an offline fashion. The main difference between them is

Symbol	Description
$\mathcal{M}$	An SSP
$\mathcal{S}$	Set of states
s_0	Initial state
$\mathcal{A}$	Set of actions
P	Transition function
c	Cost function
$\mathcal{G}$	Set of goal states
N	Tree node
h_u	Utility heuristic function
h_p	Probability-to-goal heuristic function
u	GUBS' utility function over accumulated cost of a history
K_g	GUBS' goal utility constant
c_{ucb}	Exploration constant from the UCB1 equation [13]
H	Horizon
C'	Cost obtained from a recursive call to the Search subroutine
C'_N	Cost obtained in a rollout starting from tree node N
C_{total}	Total cost of a rollout in the Search subroutine
k	Stores 0 if no goal has been reached after calling the Search subroutine recursively, or the value of K_g otherwise (equivalent to the $K_g\beta_T$ term in Definition 3)
g	Boolean value that indicates if a goal has been reached during a rollout or round
n_r	Number of rounds
n_s	Maximum number of steps to run the RunRound algorithm for
q	Maximum utility of a tree node after executing the RunRound subroutine in the Simulate algorithm
Δ_T	Time taken to execute the RunRound subroutine in the Simulate algorithm
$\mathcal{U}$	Array of utility values obtained at s_0 after running multiple rounds
$\mathcal{T}$	Array containing values indicating the time that was taken to execute each of multiple rounds
G	Array of boolean values that indicate if a goal has been reached during the execution of a round
d	Current tree depth

Table 2: List of symbols used in the algorithms defined throughout the paper.

that while the former only uses value iteration procedures, the latter also uses heuristic search in the step of computing the non-Markovian policy from the Markovian lexicographic policy.

Next section goes over UCT-GUBS, an algorithm we propose to approximately solve the GUBS criterion in an online fashion by performing Monte Carlo tree search.

Table 2 lists the symbols used in the algorithms defined in the following sections.

4 UCT-GUBS

Although there already exist exact algorithms to solve the GUBS criterion, both return a solution offline, that is, no acting is performed in the environment while planning. In other words, offline algorithms plan and return a complete policy, valid for all states reachable from s_0, before executing this policy.

Another approach is the one used by online algorithms, which both plan and act in the planning environment at the same time. Examples of such algorithms for solving SSPs under their conventional criterion are RTDP (Real-Time Dynamic Programming) [14] and UCT (Upper Confidence Bounds Applied to Trees) [10]. They aim to plan by quickly evaluating the current state to then be able to select an action that is good enough, although not necessarily optimal. This action is then selected for the agent to execute, leading to a next state, and then the process is repeated until a stop criterion is met (e.g. reaching a maximum number of steps or a goal state).

UCT applies a search procedure in the tree of outcomes from a specific state s in the process. The search tree is built as a result of applying Monte Carlo rollouts from s, and estimates of the value of each action in each tree node can be computed from the average of the costs observed in each rollout. Action selection in the rollout steps is done based on the UCB1 equation [13], defined by the following equation:

$$UCB1(s, a, d, c_{ucb}, n_{s,d}, n_{s,a,d}) = Q(s, a, d) + c_{ucb}\sqrt{\frac{\ln n_{s,d}}{n_{s,a,d}}}, \quad (3)$$

where $Q(s, a, d)$ is the quality value of action a when applied at state s from tree depth (or horizon) d, estimated from the accumulated costs obtained from executing each rollout so far; $n_{s,d}$ is the number of times the state s at depth d has been visited; $n_{s,a,d}$ is the number of times action a has been executed at state s and depth d; and c_{ucb} is the exploration constant that, depending on its value, will weight unexplored nodes of the search tree. After a established stop criterion, the rollouts stop being executed and the action with the highest value of $Q(s, a, d)$ is chosen as the best one.

To solve GUBS in an online fashion, we can adapt UCT with this goal in mind by applying a very similar approach, except for the computation of the estimates of states. Instead of using the average accumulated cost of applying rollouts from a tree node as its value, we can use the utility $U(C_T, \beta_T)$ defined by GUBS (see Definition 3), considering summaries (C_T, β_T) of the histories generated by these rollouts.

Thus, we propose UCT-GUBS (Upper Confidence Bounds Applied to Trees Under GUBS), an algorithm inspired on UCT that can be used to compute approximate policies under the GUBS criterion online. UCT-GUBS performs various rollouts from a state tree node until a stop condition is met. When this happens, each action from this state will have a utility estimate obtained based on the results of the rollouts. Finally, the action that maximizes this utility is returned and can then be used by the agent to act in the environment. The key difference between UCT-GUBS and a conventional implementation of UCT is in the computation of these estimations. As already mentioned, it needs to reflect GUBS' utility function U, as defined in Definition 3.

The pseudo-code of UCT-GUBS is shown in Algorithm 1.

For it, we first consider that each tree node N is a tuple $\{s, C, d, n_{s,d}, n_{s,a,d}, q, ch\}$, in which:

- s is the current state;
- C is the current accumulated cost;
- d is the node depth in the tree;
- $n_{s,d}$ is the number of times node N has been reached when performing rollouts;
- $n_{s,a,d}$ is a map in which the key is an action a and the value is the number of times action a has been executed in node N when performing rollouts;
- q is a map in which the key is an action a and the value is the average utility value of a from N computed when performing rollouts;
- ch is a map in which the key is an action a and the value is another map, whose key is an augmented state (s, C) and the value is a tuple $\{p, N'\}$. In this tuple, p is the probability of reaching the state at node N' from N by applying a.

Throughout the paper we use the notation of $T.p$ to denote accessing property p of tuple T, $M[k]$ to denote access of the value of key k in the map M, and $\{\}$ to denote an empty tuple.

Algorithm 1: UCT-GUBS

```
  Input: SSP $\mathcal{M} = \langle \mathcal{S}, s_0, \mathcal{A}, P, c, \mathcal{G} \rangle$, tree node $N$, heuristic functions $h_u$ and $h_p$, utility function over
         cost $u$, goal constant utility $K_g$, exploration constant $c_{ucb}$, and horizon $H$
  Output: $a$
1 begin
2     repeat
3         SEARCH($\mathcal{M}, N, h_u, h_p, u, K_g, c_{ucb}, H$)
4     until stop condition;
5     $a \leftarrow \arg\max_{a' \in \mathcal{M}.\mathcal{A}} N.q[a']$
6 end
7 return $a$
```

UCT-GUBS (Algorithm 1) takes a tree node N, heuristic functions h_u and h_p (respectively for estimating values of utility and probability-to-goal), GUBS' parameters function u and constant K_g, exploration constant c_{ucb} and horizon H.

It then simply runs the SEARCH subroutine from N repeatedly until a specified stop condition, such as a limit of time and/or number of iterations is met. In the end, it returns an action a, which is the action that maximizes the value of $N.q$.

Algorithm 2: SEARCH

```
Input: SSP $\mathcal{M} = \langle \mathcal{S}, s_0, \mathcal{A}, P, c, \mathcal{G} \rangle$, node $N$, heuristic functions $h_u$ and $h_p$, utility function over cost
       $u$, goal constant utility $K_g$, exploration constant $c_{ucb}$, and horizon $H$
Output: Cost $C'_N$ obtained in the current rollout from $N$ and variable $g$ that is true if a goal has
        been reached in this rollout
1  begin
2      $s \leftarrow N.s$
3      if $s \in \mathcal{G}$ then
4          return $0, true$
5      end
6      if $N.d = H - 1$ then
7          return $0, false$
8      end
9      if $N$ is a leaf node then
10         $N.n_{s,d} \leftarrow 0$
11         foreach $a \in \mathcal{A}$ do
12             $N.q[a] \leftarrow h_u(s) + h_p(s)K_g$
13             $N.n_{s,a,d}[a] \leftarrow 0$
14             $C' \leftarrow c(s, a) + N.C$
15             foreach $s' \in \{s' \mid P(s, a, s') > 0\}$ do
16                 $N' \leftarrow \{s', C', N.d + 1, 0, \{\} \{\}, \{\}\}$
17                 $N.ch[a][(s', C')] \leftarrow \{p, N'\}$
18             end
19         end
20     end
21     $a \leftarrow \arg\max_{a' \in \mathcal{A}} N.q[a'] + c_{ucb}\sqrt{\frac{\ln N.n_{s,d}}{N.n_{s,a',d}}}$
22     $children \leftarrow N.ch[a]$
23     $N' \leftarrow$ sample next node according to $P(N.s, a, s')$, $\forall s' \in children$
24     $C', g \leftarrow$ SEARCH$(\mathcal{M}, N', h_u, h_p, u, K_g, c_{ucb}, H)$
25     $C'_N \leftarrow c(s, a) + C'$
26     $C_{total} \leftarrow C + C'_N$
27     if $g$ then
28         $k \leftarrow K_g$
29     else
30         $k \leftarrow 0$
31     end
32     $N.q[a] \leftarrow \frac{N.n_{s,a,d} N.q[a] + u(C_{total}) + k}{N.n_{s,a,d}[a] + 1}$
33     $N.n_{s,d} \leftarrow N.n_{s,d} + 1$
34     $N.n_{s,a,d}[a] \leftarrow N.n_{s,a,d}[a] + 1$
35 end
36 return $C'_N, g$
```

In short, the SEARCH subroutine recursively executes a rollout from a node until a goal state or the horizon H has been reached, and updates the values of N at the end. More specifically, SEARCH returns two values: the cost (C'_N) obtained in the performed rollout from node N, and a variable g such that g's value is 0 if a goal has been reached in this rollout, or 1 otherwise. As already mentioned, if a goal state or the maximum horizon has been reached, the routine is terminated. In this case, the rollout has accumulated cost 0.

If N is a leaf node (i.e. a node with no children), it needs to be initialized (lines 9–20). For each action a, the utility value of $N.q[a]$ is initialized based on heuristic functions h_u and h_p; $N.n_{s,a,d}$ is initialized with 0; and, for each state (s', C') reachable by (s, C) with probability p and $N.ch[a][(s', C')]$ is initialized with p, N', such that N' takes the value of $\{s', C', N.d + 1, 0, \{\} \{\}, \{\}\}$.

Then, the action selection is performed by choosing the action a that maximizes the $UCB1$ equation

Domain	**Instance**	**Number of states reachable from s_0**
Navigation	1	13
	2	16
	3	21
River	1	25
	2	40
	3	75
Triangle Tireworld	1	42
	2	946
	3	19562

Table 3: Information for each instance of the domains used in the experiments.

(line 21). The next node N' is sampled based on applying this action at the current state (line 23), and Search is recursively called from this node (line 24).

After that, the values of N are updated based on the rollout performed from it. The total cost of the rollout, C_{total}, is computed from the sum of the current accumulated cost C and C'_N (line 26), which is in turn the accumulated cost obtained in the rollout from N. The estimated utility of applying a at N, $N.q[a]$, is then updated (line 32) from the utility of C_{total} and of the variable k, which holds the value of K_g if a goal state has been reached from N, or 0 otherwise. This is the main difference from the conventional version of UCT, as this computes the GUBS utility value of the history obtained after executing a rollout, as opposed to just maintaining an average of the accumulated cost obtained from it. Finally, the values of the counters $N.n_{s,d}$ and $N.n_{s,a,d}[a]$ are incremented.

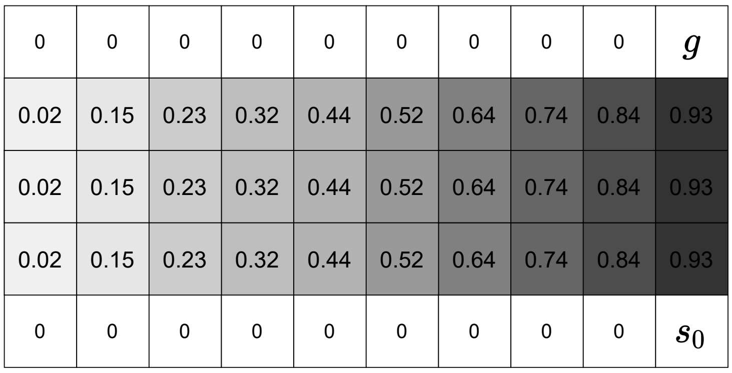
5 Experiments

To empirically evaluate UCT-GUBS, we conducted experiments in three different domains: Navigation [15], River [8] and Triangle Tireworld [16] (figures 2a, 2b, and 2c illustrate these domains, respectively).

In Navigation, the environment is a grid world, in which the agent is a robot that needs to get from one side of the grid to the other. The robot can cross the grid at any point, but inside it, actions have a positive probability of making it disappear, thus resulting in a dead end. These probabilities decrease the further the agent goes from the initial state, configuring a trade-off between cost and probability-to-goal. Figure 2a represents an instance of this domain. Each cell in the grid represented in the figure is a state of the problem. The initial state is represented by the cell labeled as s_0, and the the goal state, by the cell labeled as g. The numbers in each cell represent the probability that the robot disappears when executing any action at these cells. Also, there is one more implicit dead-end state, representing the case in which the robot disappeared.

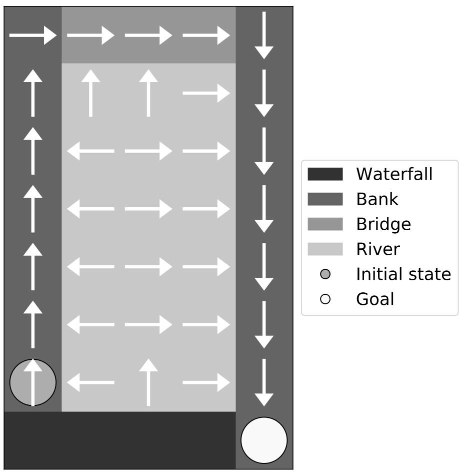
In the River domain, the goal is also to go from an initial location to a goal location in a grid world environment. The agent starts at a river bank, needing to cross the river to the other bank in order to get to the goal location. She can go all the way up to get to the bridge and cross it to get to the other bank or cross the river directly. The actions in the bank and in the bridge are deterministic, but the actions in the river have a 0.4 probability of having the agent fall down one level vertically. The bottom-most states are all dead ends representing the waterfall. Thus, there is a decision of how long to go through the bank before crossing the river (or the bridge), which involves a trade off between a higher probability-to-goal when moving up through the bank, with the caveat of paying a higher expected cost. In Figure 2b, an instance of this domain is illustrated, in which each cell of the grid is a possible state. The different types of states (waterfall, bank, bridge, river, and the initial and goal states) are represented by different shades of gray specified in the figure legend. The arrows represent an example policy that maximizes probability-to-goal, by going all the way up to the bridge to cross the river.

In the Tireworld domain, the agent represents a car that needs to go from a source location to a destination location. Every time the car moves, there is a probability of it having a flat tire. Some states have spare tires where the car can have a possible flat tire replaced. The instances are designed such that locations with no spare tire are generally contained in smaller paths, and longer paths are safer. Figure 2c illustrates three different instances of the Tireworld domain. Each of them is represented by a directed graph, where each node is a location that the car can be in. The start and goal locations are labeled in the illustrations. The state in these instances is represented not only by the location, as in the other domains explained so far, but also by the fact of whether it has a spare tire or not, which is represented by the black and white nodes, respectively, and if the car has a flat tire at the moment.

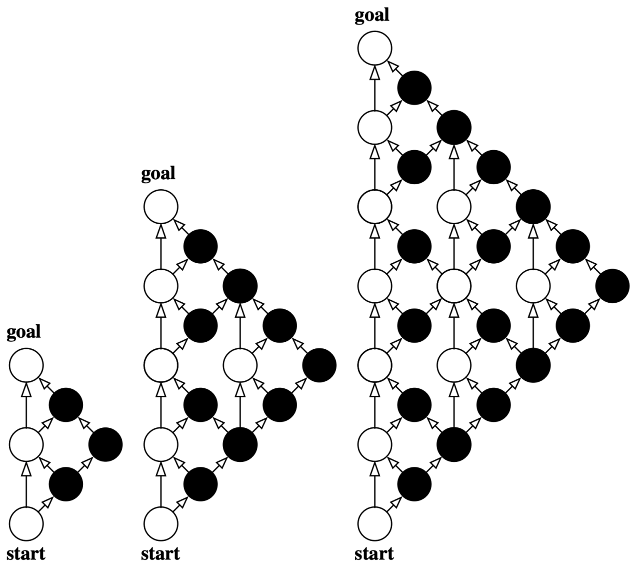


0	0	0	0	0	0	0	0	0	g
0.02	0.15	0.23	0.32	0.44	0.52	0.64	0.74	0.84	0.93
0.02	0.15	0.23	0.32	0.44	0.52	0.64	0.74	0.84	0.93
0.02	0.15	0.23	0.32	0.44	0.52	0.64	0.74	0.84	0.93
0	0	0	0	0	0	0	0	0	s_0

(a) Navigation domain (image extracted from [9]).



(b) River domain (image extracted from [9]).



(c) Triangle Tireworld domain (image extracted from [16]).

Figure 2: Illustration of the domains used in the experiments.

The experiments were performed on 3 instances of each domain. The domains and instances were defined in the PPDDL language [17] by using the PDDLGym [18] environments, and the code of the implementation is available at `https://github.com/GCrispino/uct_gubs_pddlgym/tree/clei-journal`. The number of states reachable from s_0 for all of the instances are depicted in Table 3. These are the states that can be reached from the initial state considering any possible policy.

5.1 Setup

The simulation procedure used to evaluate UCT-GUBS is specified in the Simulate routine (Algorithm 3). It takes a number of rounds n_r as parameter (we set $n_r = 30$). Thus, the RunRound subroutine (Algorithm 4) is called inside a loop (line 4). It runs a round from the initial state s_0 and returns the root node N created, the best action from s_0 after the round is run, and g, that is true if a goal state has been reached during it. Then, the 3 following arrays are returned for data to be computed from them:

- $\mathcal{U}$ stores the utility value of the best action at s_0 after each round is executed;
- $\mathcal{T}$ stores the time that took for each RunRound procedure to be executed;
- G stores whether each round reached a goal state or not. It is then used to compute an average value of the probability-to-goal from the rounds that were executed.

More specifically, the RunRound procedure begins from the initial state s_0 and depth $d = 0$ and runs a loop. For each iteration of this loop, the process is at a certain augmented state (s, C) and depth, and UCT-GUBS is run at this state to obtain the current best action a according to the rollouts that were executed (line 16). The stop condition used in UCT-GUBS is to stop when 100 rollouts (i.e. when the Search procedure is called from UCT-GUBS algorithm on line 3) have been performed. Then, the next node (representing the augmented state $(s', C + c(N.s, a))$) at depth $d + 1$ is sampled according to the transition function P based on $N.s$ and a (line 18). This procedure continues until the maximum number of steps n_s (we set $n_s = 50$) is reached (line 6), a goal state is reached (line 9), or a dead end is detected using the suboptimal dead end detection as described in Section 5.2 (line 13). The utility function used was the eGUBS exponential utility function [12] $u(C_T) = e^{\lambda C_T}$. The values used for the risk factor λ for the Navigation, River and Triangle Tireworld domains were respectively -0.01, -0.1, and -0.3. The value used for the exploration constant c_{ucb} was $1.414 \approx \sqrt{2}$.

Algorithm 3: SIMULATE

```
Input: SSP $\mathcal{M} = \langle \mathcal{S}, s_0, \mathcal{A}, P, c, \mathcal{G} \rangle$, number of rounds $n_r$, maximum number of steps $n_s$, node $N$,
       heuristic functions $h_u$ and $h_p$, utility function over cost $u$, goal constant utility $K_g$,
       exploration constant $c_{ucb}$, and horizon $H$
Output: Array of values obtained at each round: $\mathcal{U}$ for utility values at $s_0$, $\mathcal{T}$ for time taken to
        execute each round, and $G$ for boolean values indicating if each round has reached a goal
        state
1  begin
2      Let $\mathcal{U}, \mathcal{T}$ and $G$ be arrays of size $n_r$.
3      for $i = 1$ to $n_r$ do
4          $N, g \leftarrow$ RunRound($\mathcal{M}, n_s, h_u, h_p, u, K_g, c_{ucb}, H$)
5          $q \leftarrow \max_{a \in \mathcal{A}} N.q[a]$
6          $\Delta_T \leftarrow$ Record time taken to execute line 4
7          Add $q$ to $\mathcal{U}$
8          Add $\Delta_T$ to $\mathcal{T}$
9          Add $g$ to $G$
10     end
11 end
12 return $\mathcal{U}, \mathcal{T}, G$
```

The heuristic function $h_{sp}(s) = e^{\lambda d(s)}$ is used for the utility values, where $d(s)$ is the distance from (x_s, y_s) to (x_g, y_g) in G_d; G_d is a graph in which each node (x, y) represents a location in a problem and there is an edge from (x, y) to (x', y') if (x', y') is adjacent to (x, y) in this problem; (x_s, y_s) is the location of the agent in state s, and (x_g, y_g), the goal location. $h_{sp}(s)$ represents the exponential utility of the shortest path between the location of state s and the goal, if all actions were deterministic.

For the Navigation domain, a heuristic function $h_{pn}(s)$ is used to provide an estimate of probability to

Algorithm 4: RunRound

```
Input: SSP $\mathcal{M} = \langle \mathcal{S}, s_0, \mathcal{A}, P, c, \mathcal{G} \rangle$, maximum number of steps $n_s$, heuristic functions $h_u$ and $h_p$,
       utility function over cost $u$, goal constant utility $K_g$, exploration constant $c_{ucb}$, and horizon
       $H$
Output: $s_0$'s tree node $N$ and boolean value $g$ that indicates if a goal state has been reached as
        part of this round
1  begin
2      $N \leftarrow \{s_0, 0, 0, 0, \{\}, \{\}, \{\}\}$
3      $d \leftarrow 0$
4      $g \leftarrow false$
5      while true do
6          if $d = n_s$ then
7              break
8          end
9          if $N.s \in \mathcal{G}$ then
10             $g \leftarrow true$
11             break
12         end
13         if N.s is detected as a dead end then
14             break
15         end
16         $a \leftarrow$ UCT-GUBS($\mathcal{M}, N, h_u, h_p, u, K_g, c_{ucb}, H$)
17         $children \leftarrow N.ch[a]$
18         $N \leftarrow$ sample next node according to $P(N.s, a, s')$, $\forall s' \in children$
19     end
20 end
21 return $N, g$
```

goal from s:

$$h_{pn}(s) = \begin{cases} 0, & \text{if } (x_s, y_s) \text{ is a dead end} \\ 1 - p(y_s), & \text{if } (x_s, y_s) \text{ is a state where actions are probabilistic} \\ 1, & \text{otherwise,} \end{cases} \tag{4}$$

where $p(y_s)$ is the probability that the robot disappears in column y_s when applying an action in the states where actions are probabilistic.

For the River domain, the function $h_{pr}(s)$ is used to provide an estimate of probability-to-goal values:

$$h_{pr}(s) = \begin{cases} 0, & \text{if } (x_s, y_s) \text{ is in waterfall} \\ 1 - 0.4^{y_s - 1}, & \text{if } (x_s, y_s) \text{ is in river} \\ 1, & \text{otherwise.} \end{cases} \tag{5}$$

Note that the second case, in which $h_{pr}(s) = 1 - 0.4^{y_s-1}$, represents the probability-to-goal from s as if it was a state from the river's leftmost or rightmost side (next to either bank), which can be used as an estimate for river states that is better than using 0 or 1.

For the Triangle Tireworld domain, the function h_{pt} is used to estimate probability to goal values for new states:

$$h_{pt}(s) = \begin{cases} 0, & \text{if tire is flat and there is no spare} \\ & \text{tire in location } (x_s, y_s) \\ 0.5, & \text{if there is no spare tire} \\ & \text{in neighbors of } (x_s, y_s) \\ 1, & \text{otherwise.} \end{cases} \tag{6}$$

The rationale for the case where $h_{pt}(s) = 0$ is that if s does not have a spare tire and the car's tire is flat, then s is a dead end. For the second case, in which $h_{pt}(s) = 0.5$, the explanation is that if there is no spare tire in the neighboring locations of s's location (x_s, y_s), this means that the probability-to-goal from s is precisely the probability of getting a flat tire when moving to the next location. This probability is 0.5.

5.2 Search optimizations

To aid the search in UCT-GUBS, we use the following procedures:

- Action pruning – actions that are obviously not the better choices, similar to what is defined in [19], can be pruned form the search. This includes actions that are obviously worse than others at the same node (i.e., when this action leads to the same set of states as the other but at a higher cost), and actions that lead to the same state when actions that lead to other states exist. This helps reducing the search space;
- Suboptimal dead end detection – in the last step, actions that lead to the same state are verified. If at a certain node there are only actions of this type, then we can guarantee that this node's state is a dead end. In such cases, we can end the rollouts by computing the remaining cost of this them by multiplying the cost of applying this action for the remaining steps considering the horizon;
- Exploration constant normalization – since the first term of the UCB1 equation (Equation 3) is related to the utility values of the given node and the second term is responsible for weighing exploration in the action selection, the same value of c_{ucb} can have different weights for different settings (e.g. domains, instances, GUBS parameters), if these settings reasonably change the range of the utility values. Because of that, when computing the UCB1 values in UCT-GUBS' line 21, we multiply c_{ucb} with the node's current maximum utility value ($\max_{a \in \mathcal{A}} N.q[a]$).

5.3 Results

This section contains the analysis about the results obtained in the experiments. It is separated into three subsections that will respectively cover the following metrics: probability-to-goal and utility; count of actions taken in the initial state at each round; and time.

Note that UCT-GUBS solves a finite horizon version of the SSPs provided as an input, since it has a maximum horizon value that is considered. This version of the problem is an approximate version of the indefinite horizon original version of the SSP problem. Thus, one of the goals of the experiments is to observe how close the solutions obtained by UCT-GUBS can get from the optimal solutions of the instances analyzed.

5.3.1 Probability-to-goal and utility

Figure 3 shows the average values of probability-to-goal in the initial state obtained on each round. This value is obtained by dividing the number of times a goal state was reached in a round by the number of rounds. These values are compared to the probability-to-goal of GUBS' optimal policy for each value of K_g in the given instance. Figure 4 shows the average values of the best utility values of the initial state on each round. This value is the maximum utility value of the initial state node, that is, $\max_{a \in \mathcal{A}} N.q[a]$. They are compared to $V^*_{GUBS}(s_0)$, the optimal value under the GUBS criterion at the initial state.

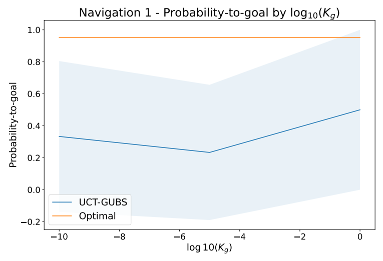
For the Tireworld domain, UCT-GUBS was able to reach the values of probability-to-goal close to optimal well enough in average. For the Navigation and River domains, this was not the case. This could be explained by the different structure of these two domains compared to Tireworld, or by the choice of parameters.

For utility values, the results obtained by UCT-GUBS in the Tireworld were also closer to optimal than in the Navigation and River domains. This is expected from the probability-to-goal values since these are correlated with utility values. However, it can be noted, mainly in the Tireworld domain, that as instances get bigger, the values obtained by UCT-GUBS get further away from the optimal. This is probably due to the algorithm having to explore the problem in much greater detail in bigger domains in order to reach values close to optimal, when compared to smaller instances.

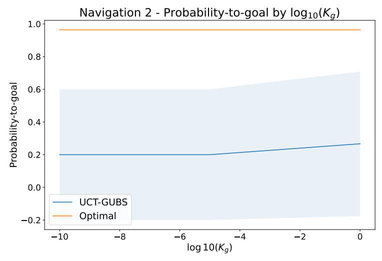
5.3.2 Action choices

Figure 5 shows the actions that have been taken at the first step of each round of UCT-GUBS, and indicates how many times each action has been chosen considering all rounds for each setting.

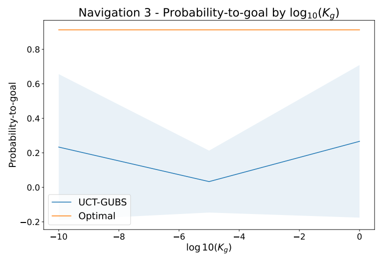
Even though values of probability-to-goal and utility were not as close to optimal as in the Tireworld experiments, for the Navigation and River domains, the optimal actions at the initial state were taken at least as many times or more often than other actions in 6 out of 9 cases ($\approx 67\%$), and in 8 out of 9 cases ($\approx 89\%$). For Tireworld, this happened for all cases, which is a good result that correlates well with the probability-to-goal and utility values obtained that were near-optimal.



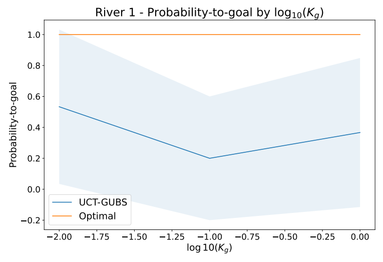
(a)



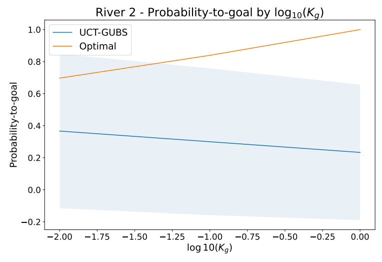
(b)



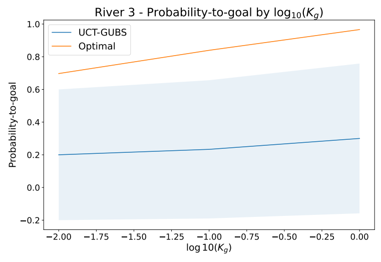
(c)



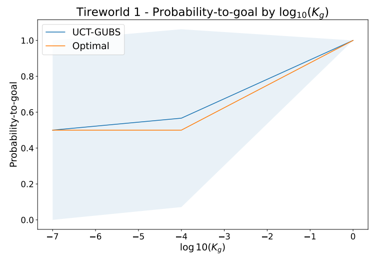
(d)



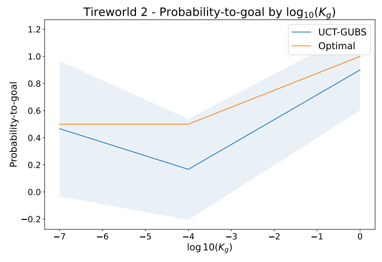
(e)



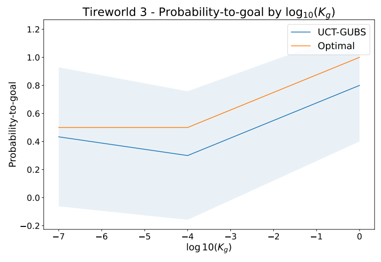
(f)



(g)

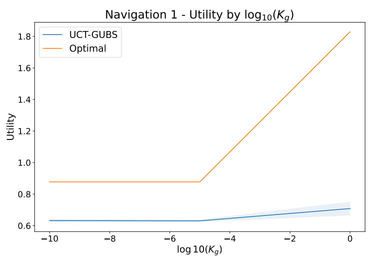


(h)

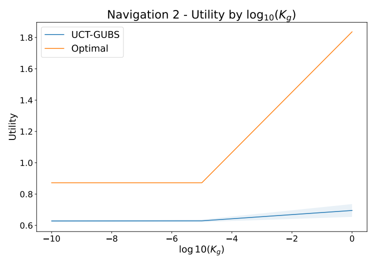


(i)

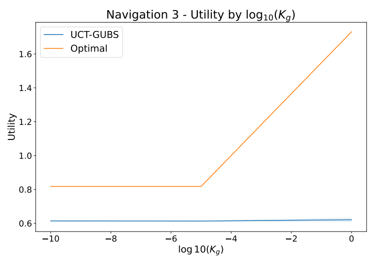
Figure 3: Average probability-to-goal value at s_0 gotten from executing UCT-GUBS per value of K_g (displayed in $\log_{10}$ scale), compared with optimal.



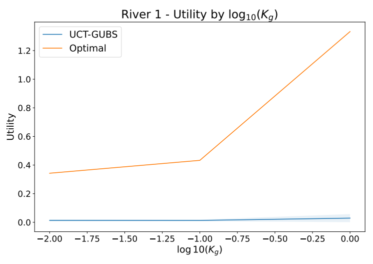
(a)



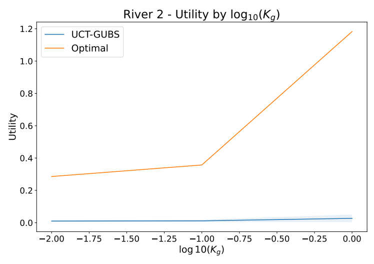
(b)



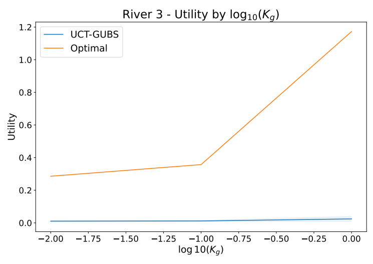
(c)



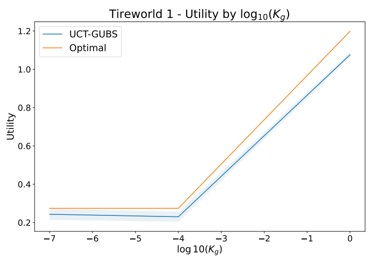
(d)



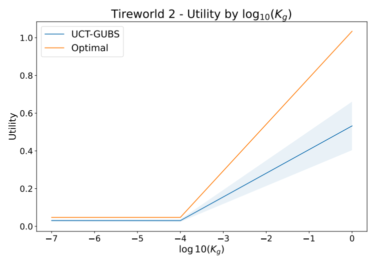
(e)



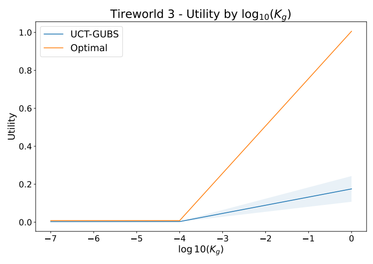
(f)



(g)



(h)



(i)

Figure 4: Average best utility value at s_0 gotten from executing UCT-GUBS per value of K_g (displayed in $\log_{10}$ scale), compared with optimal.

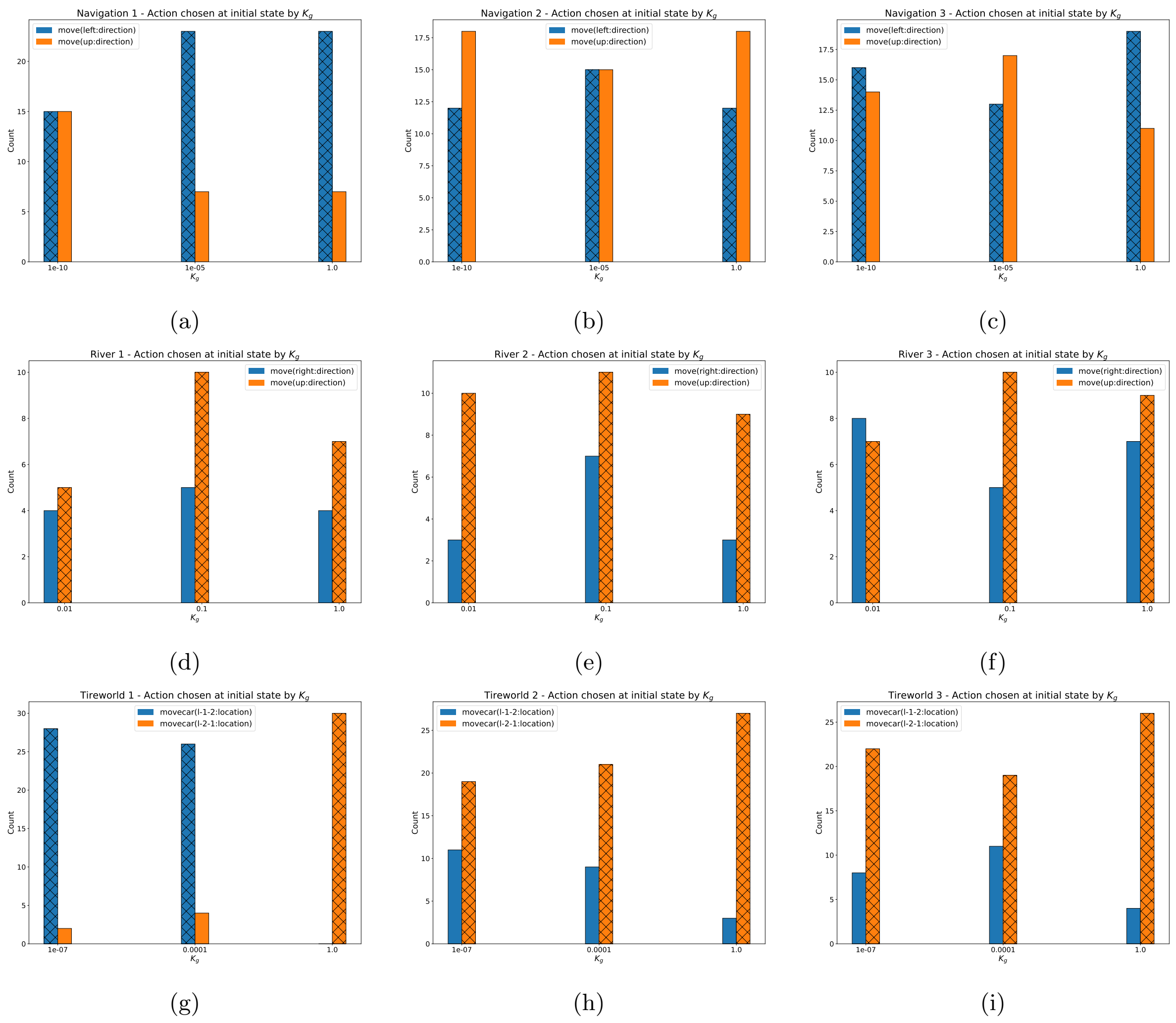
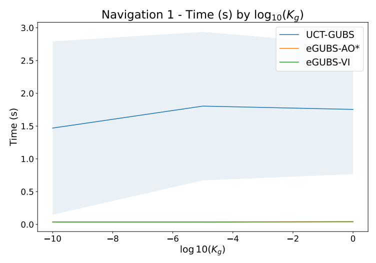
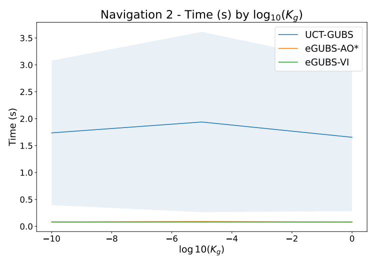


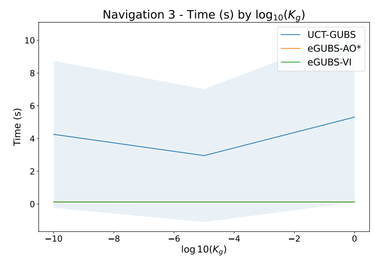
Figure 5: Count of actions chosen at initial state by UCT-GUBS per value of K_g. The bars that reference the optimal action are highlighted.



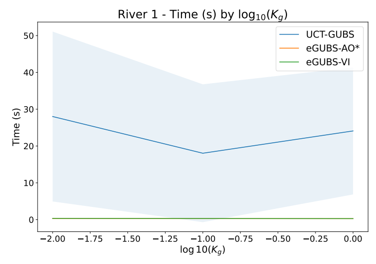
(a)



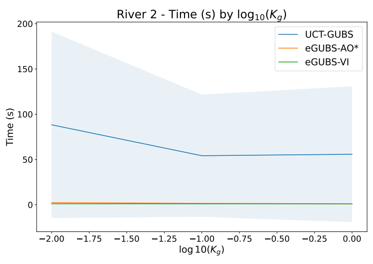
(b)



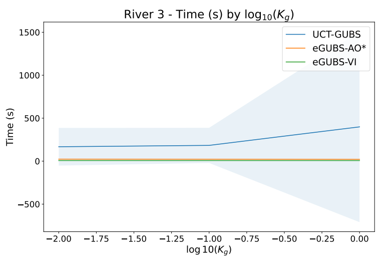
(c)



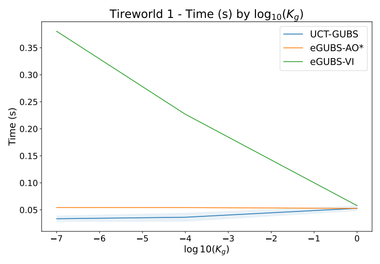
(d)



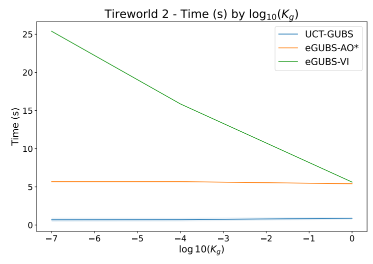
(e)



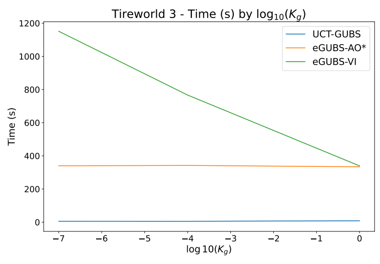
(f)



(g)



(h)



(i)

Figure 6: Time (s) taken to execute each round on UCT-GUBS per value of K_g (displayed in $\log_{10}$ scale), compared with time average time to execute eGUBS-VI and eGUBS-AO*.

5.3.3 Time

Figure 6 summarizes the results of the average time in seconds taken to execute each of the 30 rounds of UCT-GUBS, compared to the average time of running algorithms eGUBS-VI and eGUBS-AO* 10 times each.

UCT-GUBS achieved better time performance for Tireworld than optimal algorithms eGUBS-VI and eGUBS-AO*. In the third and bigger instance experimented, for example, the average time to execute each round was lower than 10 seconds, while for eGUBS-AO* it is around 300 seconds.

For the Navigation domain, as for the other metrics experimented, performance was worse than in other algorithms. Something that contributes to it is that it was observed that, in this domain, in certain cases the best action for each round is going back to a state that was just visited, and that this can happen more than once for a round, which can lead to longer round histories and total round planning time. For example, if the agent is at one of the states where actions are deterministic on Navigation, then it chooses to go to the left (to another state where actions are deterministic). Sometimes, likely because of not having sample data that is good enough after performing rollouts, it can happen that the result of it is to choose to move to the right, going back to the same state it was in the previous step, with no clear advantage. Also note that since the policy is non Markovian, depending on the accumulated cost and horizon, the choices in the same state can be different, which can also contribute to this behavior.

The time results in the experiments in the River domain were similar to the ones from the Navigation domain, in which UCT-GUBS is worse than other algorithms. This can be explained due to these two domains having a similar structure, especially regarding the tree of outcomes resulting from the cases in which the agent enters the river. The fact that the structure of the River domain is more elaborate than the Navigation one can also make it harder to obtain good estimates of values, as well as taking more time to run rollouts, resulting in a longer time to execute (this can be tuned when configuring the algorithm by reducing the number of rollouts, with the cost of losing accuracy in the utility and probability-to-goal estimates).

6 Conclusion

GUBS is a criterion for solving SSP problems with unavoidable dead ends, while being one of the few of such criteria that maintain the α-strong probability-to-goal priority property for any $0 \leq \alpha \leq 1$. In the present work we introduce UCT-GUBS, an online Monte Carlo tree search algorithm based on UCT for finding approximate policies under the GUBS criterion. Since it is an online algorithm, unlike other exact algorithms for GUBS eGUBS-VI and eGUBS-AO*, it can find an action at the same time of planning. Also, these algorithms work for eGUBS, a particular case of GUBS, while UCT-GUBS works for any utility function used as a parameter.

We performed experiments on UCT-GUBS on the Navigation, River, and Triangle Tireworld domains, to evaluate how its solutions approximate optimal solutions, and how its time performance compares to the ones of eGUBS-VI and eGUBS-AO*. It has been observed that in one of the domains (Triangle Tireworld), UCT-GUBS was able to significantly outperform eGUBS-VI and eGUBS-AO* in time while its resulting policies had values close to optimal. On the other domains (Navigation and River), UCT-GUBS performed worse than other algorithms, and could not approximate optimal values well enough.

Thus, this can indicate that UCT-GUBS might be a good choice depending on the environment of the application. In any case, however, its stop criterion can be modified such that less planning time is spent, with the caveat of compromising the quality of the resulting policy.

Acknowledgments

This study was supported in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) – Finance Code 001, by the São Paulo Research Foundation (FAPESP) grant #2018/11236-9 and the Center for Artificial Intelligence (C4AI-USP), with support by FAPESP (grant #2019/07665-4) and by the IBM Corporation.

References

[1] D. Bertsekas, *Dynamic Programming and Optimal Control.* Belmont, Mass: Athena Scientific, 1995.

[2] A. Kolobov, D. S. Weld, and H. Geffner, "Heuristic search for generalized stochastic shortest path MDPs," in *Proceedings of the Twenty-First International Conference on International Conference on Automated Planning and Scheduling*, 2011, pp. 130–137.

[3] A. Kolobov, D. Weld *et al.*, "A theory of goal-oriented MDPs with dead ends," in *Uncertainty in artificial intelligence : proceedings of the Twenty-eighth Conference [on uncertainty in artificial intelligence] (2012)*, 2012, pp. 438–447.

[4] F. Teichteil-Königsbuch, "Stochastic safest and shortest path problems," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012, pp. 1825–1831.

[5] F. Teichteil-Königsbuch, V. Vidal, and G. Infantes, "Extending classical planning heuristics to probabilistic planning with dead-ends," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011, pp. 1017–1022.

[6] F. W. Trevizan, F. Teichteil-Königsbuch, and S. Thiébaux, "Efficient solutions for stochastic shortest path problems with dead ends." in *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence (UAI) (2017)*, 2017, pp. 1–10.

[7] G. N. Crispino, V. Freire, and K. V. Delgado, "α-MCMP: Trade-offs between probability and cost in SSPs with the MCMP criterion," in *Brazilian Conference on Intelligent Systems.* Springer, 2023, pp. 112–127.

[8] V. Freire and K. V. Delgado, "GUBS: a utility-based semantic for goal-directed Markov decision processes," in *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*, 2017, pp. 741–749.

[9] G. N. Crispino, V. Freire, and K. V. Delgado, "GUBS criterion: Arbitrary trade-offs between cost and probability-to-goal in stochastic planning based on expected utility theory," *Artificial Intelligence*, vol. 316, p. 103848, 2023.

[10] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *European conference on machine learning.* Springer, 2006, pp. 282–293.

[11] G. Crispino, V. Freire, and K. V. Delgado, "Algoritmo de Monte Carlo Tree Search para SSPs sob o critério GUBS," in *Anais do XVII Encontro Nacional de Inteligência Artificial e Computacional.* Porto Alegre, RS, Brasil: SBC, 2020, pp. 402–413.

[12] V. Freire, K. V. Delgado, and W. A. S. Reis, "An exact algorithm to make a trade-off between cost and probability in SSPs," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 146–154.

[13] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[14] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1, pp. 81–138, 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/000437029400011O

[15] S. Sanner and S. Yoon, "IPPC results presentation," in *International Conference on Automated Planning and Scheduling*, 2011, accesses. [Online]. Available: http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/IPPC_2011_Presentation.pdf

[16] I. Little, S. Thiebaux *et al.*, "Probabilistic planning vs. replanning," in *ICAPS Workshop on IPC: Past, Present and Future*, 2007, pp. 1–10.

[17] H. L. Younes and M. L. Littman, "PPDDL1. 0: The language for the probabilistic part of ipc-4," 2004.

[18] T. Silver and R. Chitnis, "PDDLGym: Gym environments from PDDL problems," in *International Conference on Automated Planning and Scheduling (ICAPS) PRL Workshop*, 2020, pp. 1–6. [Online]. Available: https://github.com/tomsilver/pddlgym

[19] T. Keller and P. Eyerich, "PROST: Probabilistic planning based on UCT," in *Proceedings of the Twenty-Second International Conference on International Conference on Automated Planning and Scheduling*, 2012, pp. 119–127.