Metareasoning for Deliberation Time Distribution in the PROST Planner

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
http://ai.cs.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Thomas Keller

Ferdinand Badenberg
ferdinand.badenberg@unibas.ch
14-055-206

10.12.2017

Acknowledgments

I would like to offer my thanks to Dr. Thomas Keller for all the support, insight and guidance he provided me with and to Prof. Dr. Malte Helmert for the opportunity to write this thesis in his research group. Calculations were performed at sciCORE (http://scicore.unibas.ch/) scientific computing core facility at University of Basel and I also want to thank them for their resources.

Abstract

Probabilistic planning expands on classical planning by tying probabilities to the effects of actions. Due to the exponential size of the states, probabilistic planners have to come up with a strong policy in a very limited time. One approach to optimising the policy that can be found in the available time is called metareasoning, a technique aiming to allocate more deliberation time to steps where more time to plan results in an improvement of the policy and less deliberation time to steps where an improvement of the policy with more time to plan is unlikely.
This thesis aims to adapt a recent proposal of a formal metareasoning procedure from Lin. et al. for the search algorithm BRTDP to work with the UCT algorithm in the Prost planner and compare its viability to the current standard and a number of less informed time management methods in order to find a potential improvement to the current uniform deliberation time distribution.

Table of Contents

1 Introduction

Probabilistic planning differs from classical planning by including non-deterministic actions with effects that take place with certain known probabilities. A probabilistic planning problem is typically modeled by a Markov decision process for which a planner will then develop a solution called a policy. Planners can be divided into online planners, i.e. planners which alternate between steps of thinking (planning, finding a good action to be executed next) and acting (executing the best action) and offline planners which come up with a complete policy before executing any actions.

Popular algorithms used by such online planners are RTDP (Real-Time Dynamic Programming) [2] and its variations like BRTDP (Bounded RTDP) [3] or LRTDP [4] but also UCT (UCB applied to trees) [7]. They evaluate the current state of the Markov decision process and provide an estimation of the expected reward for the applicable actions in this state (often simply called a Q-value), based on which the next action will be chosen. The more often these algorithms can evaluate the actions, the larger the part of the following actions that is guaranteed to be correctly estimated. In fact, given an infinite amount of resources, they will come up with the optimal solution. Therefore it would be best to let them evaluate for as long as possible, resulting in optimal action choices.

However, in all applications of planning time is limited as it is generally not possible to simply give them all the time they need. In this somewhat theoretical application of planning we assume that there is no time cost to acting, which is different from some real world applications where deciding a robot's next action should be walking two meters takes some time to execute. In both cases however there is an explicit cost to thinking as opposed to acting, different from the action costs, in that thinking expends the limited time (and in our case, acting does not). Consequently it would be preferable to only continue thinking when there is a high chance that the action that is regarded as best will change after the next thinking cycle. In "Metareasoning for Planning Under Uncertainty", Lin et al. [1] propose a formal way to make such a decision that only continues thinking if the Q-values are likely to improve in the environment of BRTDP.

This thesis adapts their concept and applies it to the Prost planner [8] that uses UCT, comparing the results to both the current standards for planning under uncertainty and a selection of intuitive, hand made functions distributing the time to think about each action without further knowledge of the problem. The general idea of the metareasoning procedure is to gather the Q-values generated by the search algorithm for all actions and use the latest change in the Q-values to appraise their next values, based on the reasonable assumption that the next change will generally not be larger than the previous change. Using these next values, an expected value for the next thinking cycle can be calculated and compared to the currently best action.

In order to do this, we first declare the formal background needed for the procedures. In a second part we will talk about the different approaches for metareasoners, outline the formal approach of Lin et al. and its application to the UCT setting as well as present some ideas for improvement. We will then evaluate the different approaches based on our experiments and compare and contrast the different results. Lastly we will summarise the thesis and discuss both flaws and promising aspects of the procedures, providing an overview for potential future work.

2 Background

In the following chapter, a brief overview over the environment in which this project is set will be given and the concepts needed to understand the context of the work and the choices that were made will be formally introduced.

2.1 Probabilistic Planning

In contrast to classical planning where only deterministic problems are considered, probabilistic planning deals with non-deterministic problems where the effects of actions have a certain probability tied to them. Therefore probabilistic planning tasks differ from classical planning tasks and are often modeled as Markov decision processes. Probabilistic planning is then finding a policy for such a planning task formalised as a Markov decision processes.

2.1.1 Markov Decision Drocess

Probabilistic planning tasks are formalised as a Markov decision process (MDP) which is a 6-tuple $M = \langle S, s_I, A, T, R, H \rangle$, where

- S is a finite set of states;
- $s_I \in S$ is the initial state;
- A is a finite set of actions;
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function where $T(s, a, s')$ models the probability of reaching a state s' given action a is applied in state s;
- $R : S \times A \rightarrow \mathbb{R}$ is the reward function on a;
- $H \in \mathbb{N}$ is a finite horizon.

Such an MDP represents a model for decision processes with the Markov property that new states are selected only with regard to the current state and the selected action and without the influence of previous decisions.

2.2 Anytime Setting

2.2.1 Metareasoning Problem

The MDP describing our planning task is now expanded to a time-limited problem, the metareasoning problem $\mathcal{M} = \langle M, \mathcal{R}, \mathcal{T}, X \rangle$, where

- M is an MDP describing the planning task
- $\mathcal{R} \in \mathbb{N}$ is a finite amount of rounds to be executed
- $\mathcal{T} \in \mathbb{R}^+$ is the time given to solve M and execute the policy ρ times
- X is an anytime algorithm that provides a Q-value estimate $\hat{x}(a) \in \mathbb{R}$ for each action a and stops in sufficiently small time intervals.

2.2.2 Metareasoner

Let $\mathcal{M}$ be a metareasoning problem. Then a metareasoner $\chi^{\mathcal{M}}$ is a function that maps a 6-tuple $\langle s_0, \rho, \sigma, \tau, t, \hat{x} \rangle$ to "act" or "think", where

- $s_0 \in S$ is the current state of the decision process;
- $\rho \in \{1, ..., \mathcal{R}\}$ is a finite amount of rounds remaining;
- $\sigma \in \{1, ..., H\}$ is a finite amount of steps remaining in this round with the total number of steps per round given by the horizon;
- $\tau \in [0, ..., \mathcal{T}]$ is the time remaining to solve M and execute the policy ρ times;
- t is the smaller value of the time since the last decision to act or the time since the very first thinking cycle;
- $\hat{x} \in \mathbb{R}^{|A|}$ is the current state of algorithm X.

This decision to act or to think will lead to another metareasoning decision step, where the input depends on the selected decision.

Let μ be the time the metareasoning decision step took and $a \in A$ be the action with the highest $\hat{x}(a)$.
Then, if the decision is to act, first the action a is applied and the next metareasoning decision step will take $\langle s_0', \rho', \sigma', \tau', t', \hat{x}' \rangle$ where

- $s_0' \in S$ with $s_0' \sim T(s_0, a, \cdot)$ so s_0' is the resulting state when applying the chosen action a to s_0 determined by sampling a successor state according to T
- $\rho' = \begin{cases} \rho - 1 & \text{if } \sigma = 0 \\ \rho & \text{otherwise} \end{cases}$
- $\sigma' = \begin{cases} H & \text{if } \sigma = 0 \\ \sigma - 1 & \text{otherwise} \end{cases}$

- $\tau' = \tau - \mu$
- $t' = 0$
- $\hat{x}'$ is the next state of algorithm X

as an input. Notably, the execution of the action itself takes up no time.

If the decision is to think, rather than applying an action the expected reward for all actions will be reevaluated and another metareasoning decision step will follow, taking $\langle s_0', \rho', \sigma', \tau', t', \hat{x}' \rangle$ where

- $s_0' = s_0$
- $\rho' = \rho$
- $\sigma' = \sigma$
- $\tau' = \tau - \mu$
- $t' = t + \mu$
- $\hat{x}'$ is the next state of algorithm X

as an input.

2.3 Search Algorithms: UCT and BRTDP

Regarding search algorithms for finding a policy for an MDP, the major contenders are: Dynamic Programming (e.g. RTDP, BRTDP), Heuristic Search (e.g. AO*) and Monte-Carlo Tree Search (e.g. UCT). In the following part we will take a closer look at both BRTDP and UCT and discuss their key differences in regard to their usage as a search algorithm for metareasoning, more specifically as the algorithm X we described before. To do so we take a look at both of them under the steps that are commonly used to describe Monte-Carlo Tree Search [5] but very similar to the more general THTS framework [6]. These are the four steps: selection, expansion, simulation (or heuristic in a general case) and backpropagation.

2.3.1 UCT

The most popular version of MCTS algorithms is UCT, Upper Confidence Bounds applied to trees [7], which applies the multi-armed bandit algorithm UCB1 to Monte-Carlo planning trying to balance exploitation (further abusing the best results) and exploration (looking for other options). The important thing about the UCT algorithm for this project is that it can evaluate the problem and can give us reward estimations for all the applicable actions in a state by estimating their optimal value function.

Regarding the aforementioned steps, UCT handles them in the following ways:

- Selection: the UCB1 algorithm is applied to select a child.
- Expansion: chance nodes for all available actions are generated.
- Heuristic: UCT uses a random walk heuristic to estimate the optimal value.
- Backpropagation: Monte-Carlo Backups, meaning the average of the simulation results is backpropagated.

2.3.2 BRTDP

Bounded Real-Time Dynamic Programming (BRTDP) [3] uses both an upper and a lower bound of the optimal value function, allowing it to make anytime performance guarantees while acting greedily.

BRTDP handles the steps above in the following way:

- Selection: BRTDP uses a greedy policy to select the child resulting in the highest valued reward function.
- Expansion: both chance nodes for all available actions and their child nodes (who are again decision nodes) needed for the Bellman Backups are generated.
- Heuristic: both an admissible heuristic and a heuristic that never underestimates the actual cost are used to maintain this lower/upper bound on the optimal value.
- Backpropagation: Bellman Backups, meaning the best result is backpropagated.

3 Metareasoners

3.1 Hand Made Functions

A rather naive approach to metareasoning is to completely disregard the search algorithm with its current Q-values and only base the decision on the remaining steps, rounds and time. From these three parameters a time t is allocated for the current step and the decision to act will be made if the time used to think in this step so far exceeds t, otherwise the decision is to think again. Formally, these hand made metareasoners can be defined as follows:

$$\chi^{\mathcal{M}}(\langle s_0, \rho, \sigma, \tau, t, \hat{x} \rangle) = \begin{cases} \text{act} & \text{if } t > t' \\ \text{think} & \text{otherwise} \end{cases}$$

where t' is calculated differently for each function. Most of the calculations use a the total number of remaining steps $s = \sigma + \rho * H$. Additionally, some of these metareasoners include an additional parameter α in their calculation of t'.

As this approach does not need a Q-value for each applicable action to make its decision, it can be evaluated after every single trial. The current standard approach of the Prost planner for time management is actually a version of this which uses a uniform distribution of the remaining time over the remaining steps. In addition to this existing one, we implemented several versions of this concept that try to bias the time distribution towards steps that intuitively seem to have more influence on the solution quality without having further information about the specific problem. In the following we will talk about the way they calculate t'. Figure 3.1 showcases an example for the time distribution of these functions using the results on a problem instance.

3.1.1 Uniform

This time management method is the current default version for the Prost planner, the idea is to uniformly distribute the remaining time over the remaining steps. Thus,

$$t' = \frac{\tau}{s}$$

This is based on the very simple assumption that every step is equally important and aims to make sure that every planning step has enough time to make a good choice.

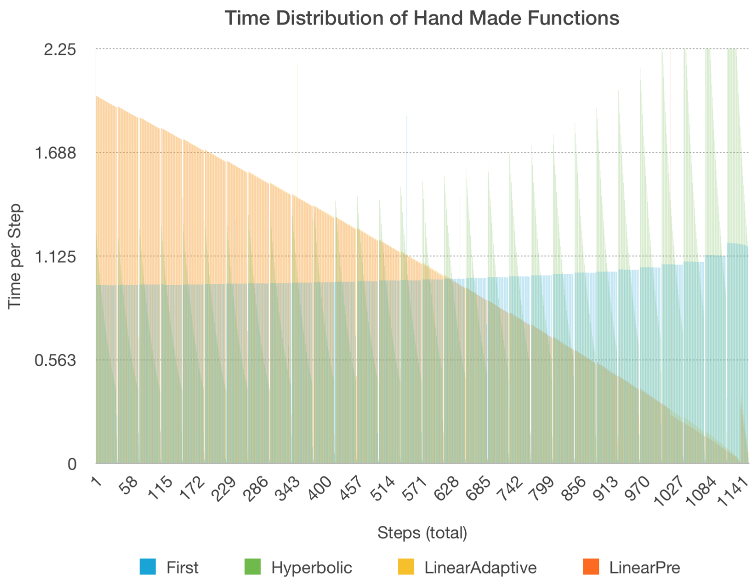

Figure 3.1: Example time distribution of the hand made functions for the wildfire 9 instance.

3.1.2 First

Coming from a uniform deliberation time distribution, a reasonable first idea is to prioritise the first step. For one, it is the step we are guaranteed to reach in every problem meaning the time allocated there is always put to use. Secondly, the planner can reuse some of the calculations of previous steps for the following steps, meaning the time spent early one can also be useful down the line. This time management method allocates a fraction of the remaining time to the first step depending on the parameter α, and uniformly distributes the time over all other steps. Formally,

$$t' = \begin{cases} \frac{\tau}{\alpha} & \text{if } s = \mathcal{R}H \\ \frac{\tau}{s} & \text{otherwise} \end{cases}$$

3.1.3 LinearPre

The next step from a distribution favouring the very first step to a distribution valuing the early steps in general as the second step will have the second highest probability of being reached every time and has almost as many steps following where its calculations could be useful. Hence this time management method aims for a linearly descending time distribution by calculating the sum over all steps and then divides the time for each step by this sum and multiplies it with the remaining steps for all rounds, resulting in a linearly descending distribution of the time over all the steps. We can describe t' as follows:

$$t' = \frac{\tau s}{c}$$

where $c = \frac{(\mathcal{R}H)^2 + \mathcal{R}H}{2}$ is calculated in the first step and decreased by s in each following step.

3.1.4 LinearAdaptive

This time management method acts similarly to the LinearPre version, but recomputes the sum over the values of the remaining steps over all rounds in each step and uses this for the division, resulting in a smooth curve instead. The formal definition of this time management method is:

$$t' = \frac{\tau s}{\frac{s^2+s}{2}} = \frac{2\tau}{s+1}$$

3.1.5 Hyperbolic

The next idea was to firstly use a hyperbolic function for the distribution instead of a linear one with the same reasoning as before, but also to apply this to each round instead of the whole problem so the first step in each round will have a similar deliberation time and so on. This assumes again a simple uniform distribution of time over all rounds. The Hyperbolic time management method divides the remaining time for this round by α in each step, meaning it allocates a similar amount of time for any step i in all rounds and decreasing the time for each step as a round goes on. Thus,

$$t' = \frac{c}{\alpha}$$

where $c = \frac{\tau}{\rho}$ is calculated at the beginning of every round and decreased after each step by t'. Intuitively, this is the most advanced of the hand made functions as it allocates more time not simply to the beginning of the calculation but to the start of each new round, hopefully supporting exploration early in a round and exploitation later on.

3.2 Metareasoner of Lin et al.

In the paper "Metareasoning for planning under uncertainty", Lin. et al. [1] present a formal solution for the aforementioned metareasoning decision step together with a BRTDP [3] algorithm. The general idea behind their procedure is to calculate values we call Q^{think} and Q^{act} representing how useful thinking and acting is respectively in the current state. Based on these values we make our decision: if Q^{think} is larger than Q^{act} we continue to think, else we execute our currently best actions. Formally, we can say that:

$$\chi^{\mathcal{M}} = \begin{cases} \text{act} & \text{if } Q^{act} \geq Q^{think} \\ \text{think} & \text{otherwise} \end{cases}$$

3.2.1 Calculating Q^{think}

Of the two values, Q^{think} is more difficult to compute: This value is essentially represented by the expected value of our policy after another thinking cycle. In order to compute this expected value we would need both the values of meta-states describing the configuration of our algorithm after thinking again and the probabilities that this state will be reached for each combination of Q-values for all actions. Unfortunately, neither the successor states nor their probabilities are know to us at that time, hence we need a way to estimate them. However, we are not actually interested in estimating the complete configuration of our

algorithm, as we really only need to know which action the algorithm will consider to be the best. This means that for each action, it suffices to estimate the values and probabilities of the successor states where this action is considered to be the best. This makes the problem much more manageable since we only need one probability and state for each action.
The idea of Lin et al. is now to estimate this next change in Q-values for each action based on the previous change in Q-values. This relies on the assumption that the change in Q-values becomes smaller in each step (we will use their notation of $\Delta Q(a)$ for the next change in the Q-value of action a and $\hat{\Delta} Q(a)$ for the last previous change in the Q-value). As the values given by the upper bound of the BRTDP algorithm actually converges toward the optimal solution and it is monotonously falling meaning the estimated Q-values are only getting smaller and closer to the actual Q-value, it is certainly reasonable albeit not guaranteed to assume that the ΔQ-values are also getting smaller with each step.
Of the two versions to calculate the next Q-values suggested by the paper we have decided to only pursue the more promising variant: we pick a single $\rho \sim$ Uniform$[0, 1]$ and calculate for all actions the next $\Delta Q(a)$-value as the last delta Q-value $\hat{\Delta} Q(a) * \rho$.
With this we can construct line segments for each action a on the unit interval with $l_a(0) = Q_{s,a}$ and $l_a(1) = Q_{s,a} + \Delta Q(a)$ where $Q_{s,a}$ is simply the Q-value given by our algorithm for the action a in the state s. This allows us to estimate the probability of action a being considered to be the best action after the next thinking cycle denoted by $P(A_{s'} = a)$ by calculating the percentage of the unit interval where l_a lies above all other line segments.
Now the only thing missing to compute our Q^{think}, which was the expected value of our policy, is the expected Q-value in the next state for all actions, given that action a was chosen, called $E[Q_{s',a}|A_s = a]$. This value is simply given by the mean of the section where l_a is considered to be the best action, i.e. above all other line segments.

3.2.2 Calculating Q^{act}

The calculation of Q^{act} is much simpler. For this value we use the mean of the values over the complete line segment of the action we currently consider to be the best action, using the values from the line segments as calculated above. At a first glance, it seems counterintuitive not to simply use the Q-value of our current best action, but it is actually very important to use the mean. This is because we do not want to think again if after the next thinking cycle we still consider the same action to be the best, even if its Q-value has improved over the thinking cycle, as then thinking again would not result in a policy change. Conveniently, when using the mean, our Q^{act} value will the the same as the Q^{think} value if the probability of our current best action also being considered the best action after the next thinking cycle is one, meaning we stop thinking if we think that our policy will not change.

Figure 3.2 and 3.3 show two examples of how Q^{think} and Q^{act} are determined using these line segments. In Figure 3.2 we can see a case where our currently best action a_1 is projected to become even better, meaning it is above a_2 and a_3 for the complete unit interval. Therefore the probability of a_1 being considered to be the best action after another thinking cycle is 1. The expected Q-value in the next state for a_1, $E[Q_{s',a}|A_s = a]$ (depicted as e_1), is now the

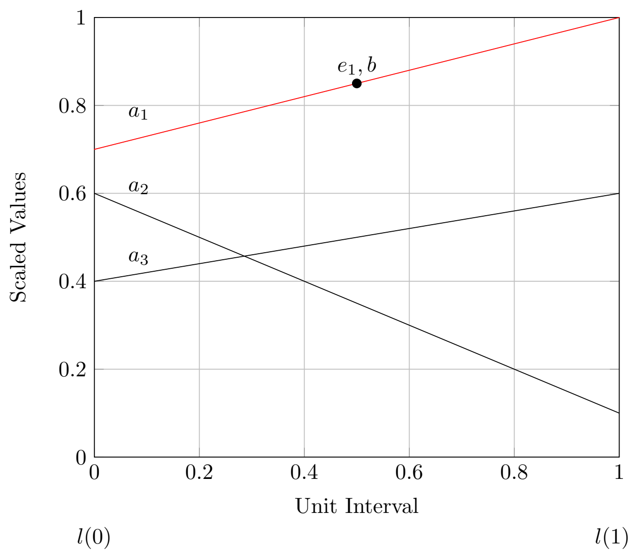

Figure 3.2: Example Line Segments with $Q^{think} = Q^{act}$

mean of the values of a_1 over the complete unit interval which is the same as the mean of the values of the current best action over the complete unit interval (depicted as b). Thus, the value of Q^{think} and Q^{act} will be the same, leading to the decision to act. This is also what we intuitively would want to do as there is no change in our policy.

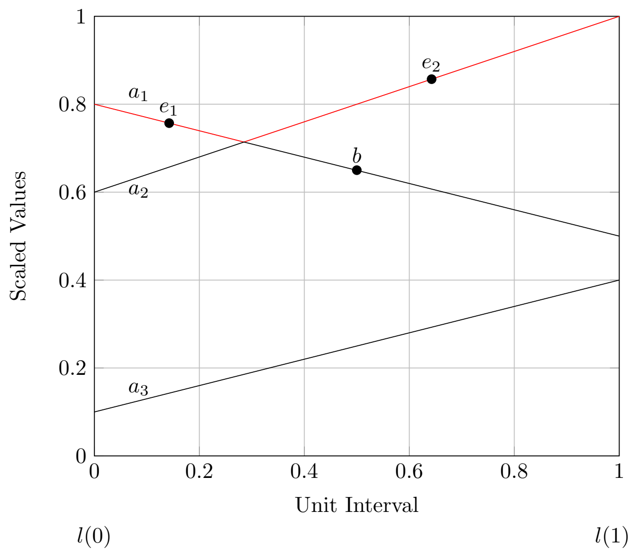

Figure 3.3: Example Line Segments with $Q^{think} > Q^{act}$

Figure 3.3 shows an example where the line segment of the projection of our action a_1 currently considered best is intersected by the line segment of a_2. This means that the probability of a_1 being considered the best action again after the next thinking cycle is the x value of the point of intersection and the probability of a_2 is 1 minus the x value of the point of intersection. $E[Q_{s',a_1}|A_s = a_1]$ is now the average over part of l_{a_1} from $l_{a_1}(0)$ to l_{a_1}(point of intersection) and $E[Q_{s',a_2}|A_s = a_2]$ is the average over l_{a_2} from l_{a_2}(point of intersection) to $l_{a_2}(1)$ (depicted as e_1 and e_2 respectively). Consequently,

$$Q^{think} = P(A_{s'} = a_1) * E[Q_{s',a_1}|A_s = a_1] + P(A_{s'} = a_2) * E[Q_{s',a_2}|A_s = a_2]$$

will be larger than Q^{act}, the average of l_{a_1} (depicted as b).

3.3 BRTDP and UCT in Metareasoning

A version of UCT is used in the Prost planner, while the original paper used BRTDP for their search algorithm. As both of these algorithms are able to stop in sufficiently small time steps, their only other aspect relevant to a metareasoning problem are their Q-value estimates. Hence, their most important difference for the metareasoning procedure lies in the way they estimate the optimal value function of the Q-values: While BRTDP only overestimates the optimal value with the guarantee of being monotonously falling, UCT only provides a balanced estimation of the optimal value function that can be lower or higher. Both algorithms however converge toward the optimal value. Figure 3.4 illustrates the possible differences in the Q-value estimation functions of the two algorithms. Another difference between the two algorithms is that the policy of BRTDP is greedy with respect to its upper bound while UCT is greedy with respect to the Q-value estimate.

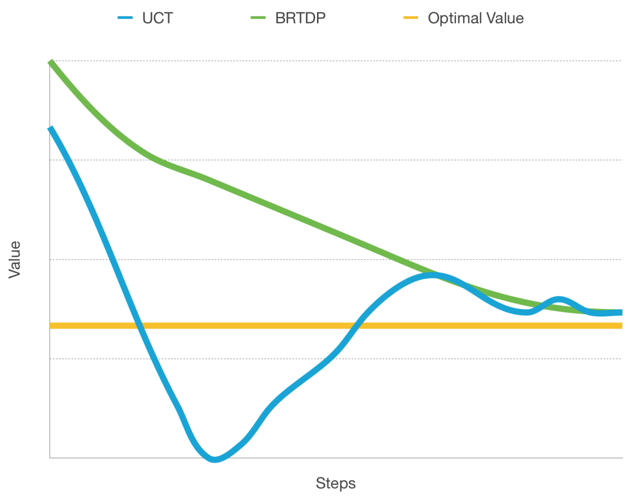

Figure 3.4: Possible UCT and BRTDP value functions together with the optimal value function.

Regarding the metareasoning procedure, the monotonously falling overestimation of BRTDP guarantees that any change in Q-values will result in a better estimation of Q-values than before. As UCT offers no such guarantee using it in combination with this metareasoner means that it is entirely possible that we correctly predict a change in our policy causing us to follow up with another thinking cycle only for this change to lead to a worse policy because later Q-value estimations in UCT can temporarily be less accurate than previous ones.

3.4 Adaptation from BRTDP to UCT

As mentioned before, the procedure presented by Lin et al. was constructed with the framework of a BRTDP algorithm in mind, which means several points have to be adapted for it to be usable with UCT, the search algorithm used by the Prost planner. While the basic assumption of the next ΔQ-values being no larger than the previous $\hat{\Delta} Q$-values is still reasonable, UCT is lacking the guarantee of being monotonically falling and only converges toward the actual value. This means that our Q-values can be both negative and positive, so technically the corresponding assumption would be to say that the absolute value of the ΔQ-values will be no larger than the absolute value of the $\hat{\Delta} Q$-values.

What does this mean for the metareasoning procedure? Firstly, for the probabilities and expected values we are interested in the sections where the line segment is above all other line segments (as described previously) instead of below (as in their paper), since our Q-values are estimations and not an upper bound and we want to increase our Q-values in this reward based setting instead of lowering them. Secondly, the $\rho \sim \text{Uniform}[0, 1]$ would technically have to be $\rho \sim \text{Uniform}[-1, 1]$ as the next ΔQ-value could be either positive or negative. This however was not changed in our implementation of the metareasoner as for one it is often rather likely that the next change will change its sign but more importantly this would increase the dependance of the outcome of our decision on the choice of ρ even more than it already does. This seems very problematic as being dependant on the choice of ρ to decide if it is better to act or to think means that the outcome is more a random chance than an actual estimation. Therefore we chose to not further increase this random aspect and keep sampling ρ from Uniform$[0, 1]$ as they did.

3.5 Metareasoner with Minimum Thinking Time

A problem we have encountered with the metareasoning procedure as given by Lin et al. is that the way it only continues to think if Q^{think} is larger than Q^{act} the implicit assumption made is actually stronger than $\Delta Q \leq \hat{\Delta} Q$. This is because in actuality we want to continue thinking if after any following thinking cycle we would pick another action, meaning by acting if $Q^{act} \geq Q^{think}$ we assume that an arbitrarily high number of thinking cycles would not lead to a better policy. Therefore we implicitly assume that the sum of all following ΔQ-values is no larger than $\hat{\Delta} Q$.
While this can still hold, it certainly does only rarely during the first few trials, especially when considering the explorative nature of UCT. In fact, the more thinking cycles have been already, the likelier it is for this assumption to hold true. For this reason we also implemented a version of the metareasoning that only follows the metareasoning procedure when a certain amount of time was already spend thinking about the current step to make our assumption more applicable for the problems and overcome the trials where it is somewhat unreasonable.

3.6 Metareasoner with a Cost of Thinking

Another problem we had with the given metareasoning procedure is that while it obviously aims to reduce the time spent thinking redundantly, there is no explicit statement of the cost that actually comes with thinking. Of course this cost of thinking we call C^{think} varies depending on the framework, but nonetheless we wanted to introduce a version of the procedure that considers an explicit cost to thinking with the environment of the PROST planner in mind. Since we are given a specific amount of time to execute several rounds of planning and acting and aim to achieve the best solution we can during this time, we are in a situation where we can have a rather high cost to thinking if we have a lot of steps still left to plan but very little time, and on the other hand if we have a lot of time left but only few steps to go, there is no gain by being done preemptively so the cost of thinking is very low.

Therefore, we suggest the cost of thinking:

$$C^{think}(\tau, s) = \begin{cases} 0 & \text{for } \frac{\tau}{s} > T_{max} \\ 1 & \text{for } \frac{\tau}{s} < T_{min} \\ T_{max} + T_{min} - \frac{\tau}{s} & \text{otherwise} \end{cases}$$

where τ is the total time remaining given as input to the metareasoner and $s = \sigma + \rho * H$ is the total number of steps remaining. We use this in another modified version where we now subtract C^{think} from Q^{think} before comparing it to Q^{act}, so formally this third metareasoner looks like this:

$$\chi^{\mathcal{M}} = \begin{cases} \text{act} & \text{if } Q^{act} \geq Q^{think} - C^{think} \\ \text{think} & \text{otherwise} \end{cases}$$

In order to make Q^{think} and C^{think} comparable in this manner we scaled the values of the line segments and thus also the values of Q^{think} and Q^{act} to lie between zero and one.

Furthermore we mentioned previously that there is no reason to stop thinking while we have still time left, i.e. $\tau > 0$. And while the aforementioned function C^{think} of the third metareasoner does not punish thinking if we have plenty of time left, it also does nothing to bias the decision towards thinking in this scenario. For this reason it could make sense to completely remove the lower bound for the cost function so that it will become negative if we have enough time left just keep thinking and we will spend the remaining time thinking rather than being done prematurely. This new cost function

$$C^{think'}(\tau, s) = \begin{cases} 1 & \text{for } \frac{\tau}{s} < T_{min} \\ T_{max} + T_{min} - \frac{\tau}{s} & \text{otherwise} \end{cases}$$

will be used in a fourth metareasoner otherwise identical to the third one.

4 Evaluation

This version of the metareasoning procedure was adapted to work in the state of the art Prost planner [8] and hence was implemented and tested within this framework. The problem domains and instances used to test on are from the International Probabilistic Planning Competition (IPPC) 2011 [9] and 2014 [10]. The results are averages over 30 runs of each of the problems as is the standard in the IPPC. They are then assigned a relative value from 0 to 100 based on their performance which is given by the scaled value of the official IPPC score.

4.1 Hand Made Functions

First is an evaluation of the hand made functions. Keep in mind that these have no information of the current state of the search algorithm and operate solely on the time and steps remaining.

4.1.1 Parameter for First

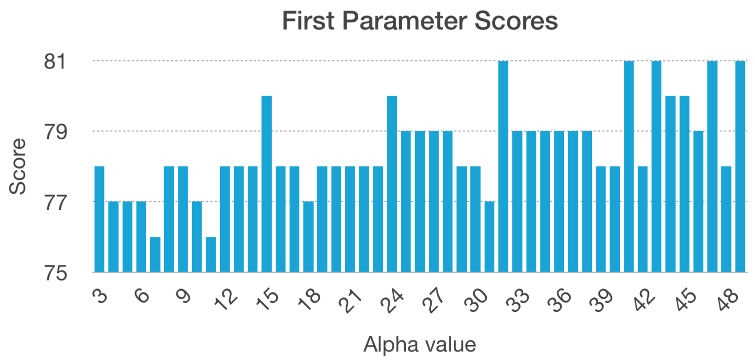

Figure 4.1: Average First scores for different parameter values on all problems.

As some of the hand made functions contain an additional parameter α, we tested different values for this parameter to find a good value for the future comparisons. Regarding the parameter for the time management method First, we tried values ranging from 2 to 50. The results can be found in Figure 4.1, ranging from 76 to 81 so only a difference of 5 in performance. Nevertheless we used an α of 32 in the following tests for this function.

4.1.2 Parameter for Hyperbolic

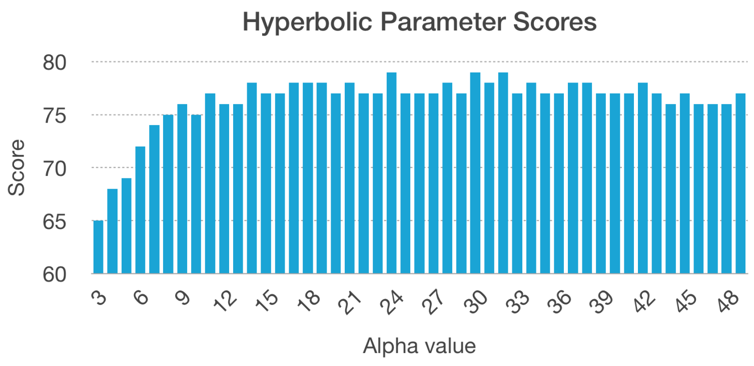

Figure 4.2: Average Hyperbolic scores for different parameter values on all problems.

For the time management method Hyperbolic, we again tested α-values from 2 to 50. The resulting performances seen in Figure 4.2 range from 65 to 79 so already a larger difference of 14 compared to results for the function First. Interestingly enough, one of the parameter values resulting in the best performance score is again an α-value of 32. Other than that we can see that performance for small α-values was the worst as the discrepancy in time allocated between the first few steps and all consecutive steps in each round is very high.

4.1.3 Comparison of the Hand Made Functions

Table 4.1 shows results for the different hand made functions, where the Uniform results are our baseline to compare them to as this, while being a (very simple) metareasoner under our previous definition, is the current default of the PROST planner and does not share the same intent of the other metareasoners trying to allocate time based on importance of the calculation step. As previously noted, First and Hyperbolic use α-values of 32 for this calculation.

Looking at Table 4.1, we can see that all hand made metareasoners but Hyperbolic actually performed better than Uniform, although only by a very small margin. In fact, all of the total scores are very close together with LinearAdaptive being the best. When looking at the totals for the different problems, LinearAdaptive is always rather strong and does not have the same drop to a bad performance the way for example Hyperbolic has in the navigation domain or LinearPre has in the triangle domain, leading to its strong overall performance. Interestingly enough, Hyperbolic performed rather poorly compared to the other functions, especially when considering that we initially thought it to be one of the more reasonable time allocations based on our intuition.

The metareasoners First, LinearAdaptive and LinearPre on the other hand show an especially large improvement over the Uniform distribution on both the Crossing and the

Table 4.1: Total IPPC scores for the different hand made metareasoners where Uniform is the current default.

Problem	Uniform	Hyperbolic	First	LinearAdaptive	LinearPre
Wildfire	74	71	80	81	**90**
Triangle	72	65	72	**75**	54
Academic	37	37	34	**45**	33
Elevators	93	93	91	**94**	**94**
Tamarisk	93	**94**	92	91	88
Sysadmin	**94**	**94**	90	91	91
Recon	97	**99**	97	96	96
Game	97	93	94	93	**98**
Traffic	**97**	**97**	96	96	**97**
Crossing	87	89	91	99	**100**
Skill	91	91	88	**93**	92
Navigation	65	58	83	82	**84**
Total	83	82	84	**86**	85

Navigation domain. All of these heavily favour the early steps and these domains can actually realistically be solved by the planner if he invests enough time, so it is likely that the early focus of these time management methods leads to just enough time for the planner to do so and then play the optimal action afterwards.

4.2 Metareasoner of Lin et al. and Improvements

The complete results for all informed metareasoners, i.e. the adaption of the metareasoner of Lin et al. and our proposed improvements for it, can be seen in Table 4.2. The Uniform result represents the control result using the same THTS configuration with the same ingredients as all of the metareasoners, the only different part is that it uses the default uniform time management method instead of the metareasoning. The THTS configuration used the following ingredients:

- Action selection: UCB1
- Outcome selection: Monte Carlo
- Backup function: Partial Bellman
- Heuristic function: IDS of Prost

Standard stands for the basic metareasoner of Lin et al. adapted to UCT and the following versions are our various potential improvements. Minimum is the metareasoner with a minimum thinking time so we increase the likelihood of our assumption being true. We used a T_{min} value of 0.1 in all of these calculations, and all of the following metareasoners will also use this minimum thinking time in addition to their other changes. Going forward, both other metareasoners are using the suggested explicit formulation of the cost to thinking C^{think} with the difference being the actual cost function: Cthink^{+} uses C^{think} as initially stated while Cthink uses the version $C^{think'}$ that omits the lower bound and allows for the cost to become negative, biasing towards thinking.

Table 4.2: Total IPPC scores for the formal versions of the metareasoner.

Problem	Uniform	Standard	Minimum	Cthink$^+$	Cthink
Wildfire	60	90	86	**95**	68
Triangle	**78**	67	62	59	68
Academic	**39**	32	36	35	38
Elevators	**98**	71	83	83	97
Tamarisk	**96**	68	86	90	92
Sysadmin	**100**	36	67	74	82
Recon	**98**	56	75	75	97
Game	**97**	64	82	86	96
Traffic	**99**	85	90	87	98
Crossing	88	58	78	83	**89**
Skill	**100**	25	71	69	86
Navigation	82	26	25	28	**83**
Total	**86**	56	70	72	83

When looking at the scores shown in Table 4.2, we can immediately see that there is a very large discrepancy between the current standard and the straight adaptation from the metareasoner of Lin et al. with the current standard being much better. Despite this bad performance however we can see that the other versions of the metareasoner with different small improvements are indeed relevant improvements regarding the performance compared to the standard version. Still, even the version of the improved metareasoners Cthink with the best results does not outperform the current standard, despite being pretty close to it. We can also see that the two big improvements over the Standard metareasoner are for one the fixation of a minimum thinking time and removing the lower bound of the cost function C^{think} so can also promote thinking if there is spare time instead of only punishing it. The performance improvement coming with Cthink$^+$ introducing the first version of C^{think} only pushing towards acting interestingly is very small compared to these other two improvements.

This suggests that the Standard metareasoner likely tends to underestimate the expected reward of thinking so it is already too quick to act, meaning a balancing towards thinking shows more improvement than a balancing towards acting.

But where does that underestimation come from? When we suggested the implementation of a minimum thinking time, we did so because we said that the way this metareasoning procedure is constructed, the assumption we make is actually more severe than $\Delta Q(a)$ being no larger than $\hat{\Delta} Q(a)$. The reason behind this is that we only consider the very next ΔQ-value to make our decision to act or to think, but the action we consider to be the best might very well only change after multiple consecutive thinking cycles.

Therefore, if we base our decision only on the next $\Delta Q(a)$, we are actually assuming that the sum over all of the subsequent $\Delta Q(a)$-values will also be no larger than $\hat{\Delta} Q(a)$. And this much stronger assumption seems indeed to be false in a lot of cases, leading to this underestimation of Q^{think}. But despite both of our ways to mitigate this underestimation improving the result, it is hard to say how precise Q^{think} is in the end.

Moreover, although the improvement going along with Cthink proves to be very relevant in practice, placing it very close to the current standard, it means that the underlying

Table 4.3: Scoring results for the problem instances of the time allocation figures.

Problem instance	Uniform	Standard	Minimum	Cthink$^+$	Cthink
Wildfire 9	33	50	59	**72**	40
Game 9	**100**	58	72	81	94

metareasoning strategy is less than ideal, since we want to allocate the time where we need it so the fact that simply using up more time to think just because we can leads to such a big improvement shows that the time budgeting is suboptimal. Also if the previous metareasoners are done with time left over it should mean that they are reasonably certain that no improvement could have been made by thinking more. Thus this large performance increase by using this spare time again shows the underestimation of the value of thinking.

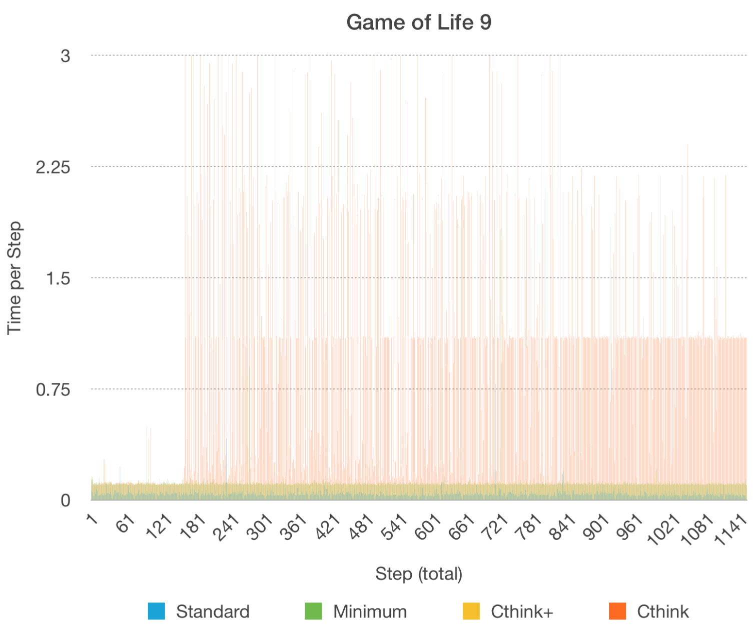

Figure 4.3: Time allocation per step in seconds for the formal metareasoners on the game of life instance 9.

Figure 4.3 shows us the time the four metareasoners as well as the uniform time distribution have allocated to each step during the solution of instance nine of the game of life domain. This domain is a good example for the average performance of these metareasoners, since as seen in Table 4.3 the scores for it are similar to the total of all domains. The almost constant uniform distribution of the control configuration would be around one in this graph, which Metareasoner Cthink exceeds with some larger spikes.

In Figure 4.4 we can see a section of the previous figure, allowing us to better make out the trends for the other metareasoners: Standard varies quite a bit depending on the step, but has very low values overall. Minimum and Cthink$^+$ are both very close to the minimum thinking time of 0.1. Cthink however shows a pattern somewhat similar to Standard but with a much larger amplitude.

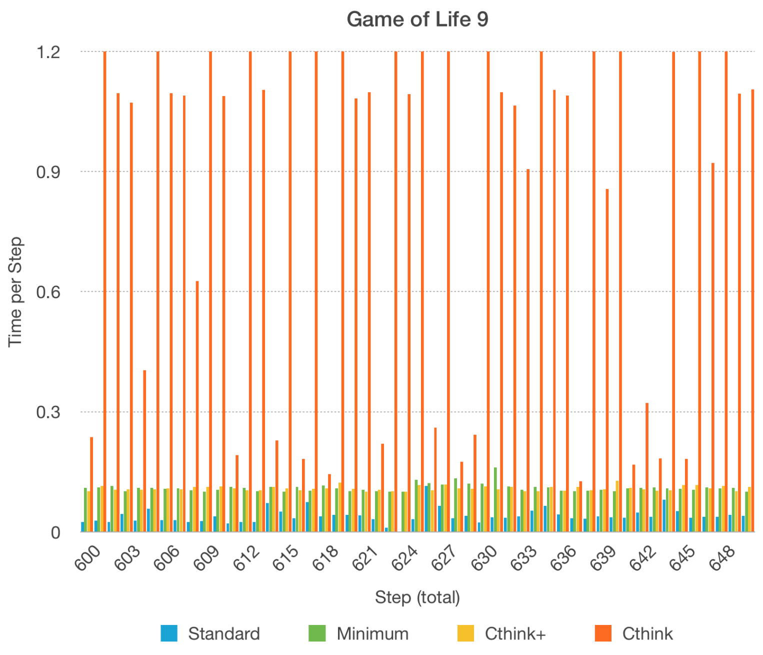

Figure 4.4: Subsection of 4.3.

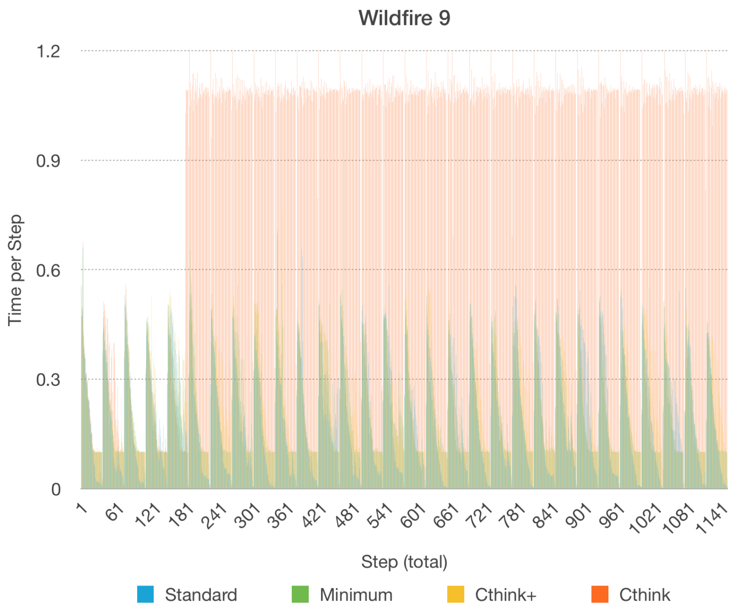

Figure 4.5: Time allocation per step in seconds for the formal metareasoners on the wildfire instance 9.

Looking at Figure 4.5 shows us another such example but for the wildfire domain. Especially when looking again at a zoomed section as seen in Figure 4.6, we can see that while Cthink and Uniform are again similar with higher values, the values for the first three metareasoners show a very similar trend resembling a linear function over a certain interval. This is very close to something one might expect to be a good time allocation intuitively, and indeed, when looking at the results, these three metareasoners actually beat the performance of Cthink and the Uniform on this instance.

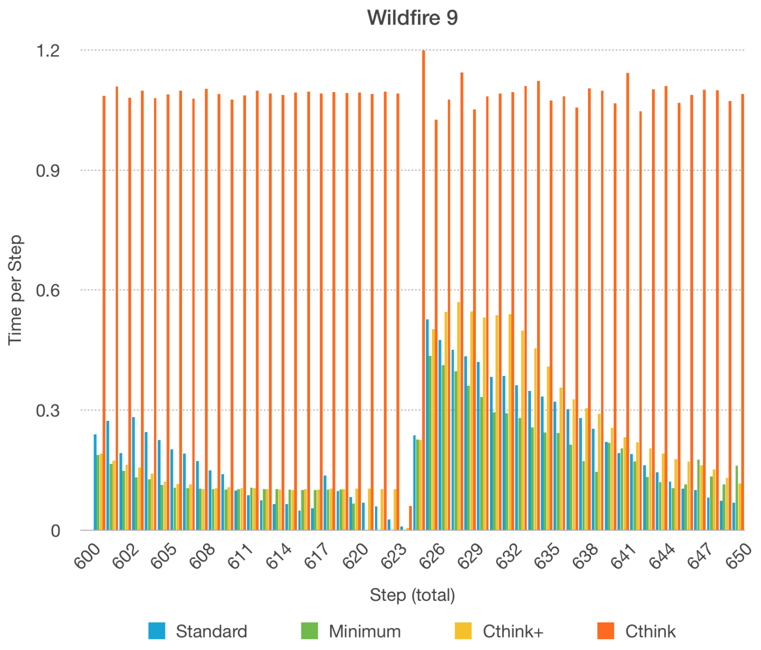

Figure 4.6: Subsection of 4.5.

This shows that there are some problems where the assumption of the formal metareasoning procedure seems to work very well and manages to outperform the current standard, but this is clearly not the case in general. It shows however that there is some dependance of a good time allocation on the specific problem, meaning there is certainly some merit to a metareasoner taking the state of the search algorithm into account.

Furthermore we tested configurations for Cthink$^+$ and Cthink using the calculations of the hand made functions for the T_{max} value as a first attempt to combine the two metareasoning concepts, however they performed strictly worse than the original versions. This is because C^{think} is modelled around the assumption of a uniform time distribution as a guideline for how much time we should still have and the standard of 1 for T_{max} fits this ratio rather well for our problems. Consequently only changing T_{max} and not $\frac{\tau}{s}$ was not a enough to combine both concepts.

5 Conclusion

The goal of this thesis was to examine and analyse different possibilities for metareasoning for the deliberation time distribution of the time given to the Prost planner. This was achieved by comparing simple hand made approaches to the metareasoning procedure presented by Lin et al., a formal way to make a decision based on the estimation of an expected reward for thinking and acting, after implementing both in the Prost planner. Furthermore, in addition to adapting the formal procedure from its BRTDP environment to a UCT environment, we proposed several possible improvements based on certain assumptions and choices of the given procedure we identified as suboptimal in the setting of the Prost planner.

The results of our experiments using the problems and score of the IPPC showed that the very simple, hand made time management functions performed very well and three of the four functions were able to improve overall performance compared to the current standard. The formal metareasoning procedure however performed very badly compared to the current standard, but all three of our main suggestions for improvement managed to increase the solution quality, in fact so far that the best solution is again very close to the current standard, but unfortunately did not manage to beat it.
The two of our improvements who ended up being very useful are the introduction of a minimum thinking time, which makes it more likely that the assumption underlying the metareasoning procedure of Lin et al. matches the actual situation, and the statement of an explicit cost to thinking weighing the algorithm towards thinking or acting depending on the time and steps left.

During this adaptation of the metareasoning of Lin et al. to UCT, we encountered some weaknesses of the procedure, partly due to the use with UCT, partly due to the procedure itself. The most important problem we encountered very severely was one they already mention themselves [1]: Due to this procedure only projecting changes for one more thinking cycle when in fact there can be many more thinking cycles if thinking is chosen, the value of thinking is in many cases underestimated. While they suggested projecting the change in Q-values over multiple thinking cycles but ultimately dismissed it as impractical due to being too imprecise, we managed to improve this problem with a minimum thinking time based on the assumption that the sum of all future ΔQ-values being no larger than the single previous ΔQ-values is more likely the more the Q-values have already been explored. A possible reason for multiple projections being so imprecise could be that the actual projection of the future ΔQ-values for all actions based on a single, randomly chosen ρ between zero and one is a rather weak way of estimating the next ΔQ-values based on the previous ones.

5.1 Future Work

Based on these problems and the solutions we used for some of them, there are still several possibilities both regarding this procedure of Lin et al. and metareasoning in general we could not yet explore. One such idea would be to base the estimation of the next ΔQ on the history of previous ΔQ-values and not just the last one as this estimation seemed to be the weakest point of their procedure. A more accurate estimation of the future ΔQ-values might also help with the underestimation of the value of thinking. Additionally, with our metareasoner formulating an explicit cost to thinking being reasonably close to the current standard, a more elaborate cost function might allow the metareasoning approach to surpass it. Based on the promising results of the simple and high level hand made approaches, one such idea was the combination of the hand made functions with the metareasoning based on the search algorithm by integrating the suggested time allocation of these functions into the cost function C^{think}: the cost function biases based on how much time should be left for the remaining steps. The current version of this function uses the assumption of a uniform distribution for the default budgeting of the remaining time, but with other functions showing better results than the uniform distribution, there is likely an improvement to be made by testing their distribution for this estimation. Moreover, it is very likely that there are still better hand made functions to be found. We realised during this project that these hand made functions and several other configurations could easily be described based on few parameters: either using one function (Linear, step, hyperbolic, constant or other function) describing the time distribution for all steps, or a combination of two such functions with one describing the time distribution for each round and the other describing the time for each step in a round. With such a parametrisation of these hand made functions, it would be easily possible to test many more configurations to likely find an even better function.

In conclusion, despite our improvements not the formal procedure based on the estimations of the search algorithms but the simple, more general time management functions outperformed the current standard method of deliberation time distribution, proving the viability of approaches different to the current uniform distribution. Additionally, these results show that there are multiple promising options for further research connecting to this work.

Bibliography

[1] Lin, C. H., Kolobov, A., Kamar, E., and Horvitz, E. Metareasoning for Planning Under Uncertainty. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 1601–1609 (2015).

[2] Barto, A. G., Bradtke, S. J., and Singh, S. P. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81 – 138 (1995).

[3] McMahan, H. B., Likhachev, M., and Gordon, G. J. Bounded Real-Time Dynamic Programming: RTDP with Monotone Upper Bounds and Performance Guarantees. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML 2005)*, pages 569–576 (2005).

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

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

[6] Keller, T. and Helmert, M. Trial-Based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, pages 135–143 (2013).

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

[8] Keller, T. and Eyerich, P. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pages 119–127 (2012).

[9] Coles, A. J., Coles, A., Olaya, A. G., Celorrio, S. J., Linares López, C., Sanner, S., and Yoon, S. A Survey of the Seventh International Planning Competition. *AI Magazine*, 33(1):83–88 (2012).

[10] Vallati, M., Chrpa, L., Grzes, M., McCluskey, T. L., Roberts, M., and Sanner, S. The 2014 International Planning Competition: Progress and Trends. *AI Magazine*, 36(3):90–98 (2015).

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Ferdinand Badenberg

Matriculation number — Matrikelnummer
14-055-206

Title of work — Titel der Arbeit
Metareasoning for Deliberation Time Distribution in the Prost Planner

Type of work — Typ der Arbeit
Bachelor Thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 10.12.2017

[illegible]

Signature — Unterschrift