University of Basel

Online Knowledge Enhancements for Monte Carlo Tree Search in Probabilistic Planning

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 and Cedric Geissmann

Marcel Neidinger
m.neidinger@unibas.ch
2014-050-280

09/02/2017

Acknowledgments

I'd like to thank Dr. Thomas Keller and Cedric Geissmann for supervising this thesis as well as Prof. Dr. Helmert for providing me with the posibility, to do the thesis in his group. Calculations were performed at sciCORE (http://scicore.unibas.ch/) scientific computing core facility at University of Basel.

Abstract

Monte Carlo Tree Search Algorithms are an efficient method of solving probabilistic planning tasks that are modeled by Markov Decision Problems. MCTS uses two policies, a tree policy for iterating through the known part of the decission tree and a default policy to simulate the actions and their reward after leaving the tree. MCTS algorithms have been applied with great success to computer Go.
To make the two policies fast many enhancements based on online knowledge have been developed. The goal of *All Moves as First* enhancements is to improve the quality of a reward estimate in the tree policy. In the context of this thesis the, in the field of computer Go very efficient, α-AMAF, Cutoff-AMAF as well as Rapid Action Value Estimation enhancements are implemented in the probabilistic planner PROST.
To obtain a better default policy, *Move Average Sampling* is implemented into PROST and benchmarked against it's current default policies.

Table of Contents

1 Introduction

Playing is the highest form of research

- Albert Einstein

Playing board games is an activity traditionally associated with humans and for a long time, humans deemed themselves unbeatable in this abstract type of problem where a player has to "think outside the box", and show "real intelligence" to beat their opponent. However when IBM's Deep Blue defeated chess champion Garri Kasparow in 1997 this dominant role came to an end. Almost 20 years later, in 2016, Google's AlphaGo software won a best of five game of Go against South Korean Go master Lee Sedol under tournament settings. Go is a highly complex strategy game that was, due to its complexity, considered to be one of the last frontiers in AI game playing.

For a computer to be able to beat a human player it needs a *plan*, a sequence of actions that lead to a desired goal state. While there are deterministic planning problems, the game of Go is based around probabilities, for example about an opponent's move. This leads to *probabilistic planning problems* that can be described by a *Markov Decision Process.* MDPs provide a framework for describing problems with a partially random outcome. In the domain of Go this means that, while the computer has full control over its own moves, the opponent's moves are subject to a probability distribution.

While MDPs can be used to formalize probabilistic planning tasks we need an algorithm to solve them. For AlphaGo Google used the class of *Monte Carlo Tree Search* algorithms to solve the Markov Decision process. MCTS algorithms are a technique that has an increasing popularity among researchers, especially in the domain of computer Go.

The goal of a planning algorithm is to find a series of actions that, once applied from a given start state, lead to a goal state. To find such a plan we need to build a search tree. In many cases this search tree can branch out very fast. MCTS overcomes this problem by building a partial search tree. To build this tree a MCTS algorithm consists of four phases and has two main components. In the first phase, a **selection**, the known region of the search tree is traversed and the next, most urgent node is selected by a *tree policy.* Here, nodes that have been visited more often in previous runs are selected more likely while still choosing actions that have not been selected too often. In the following **expansion** phase the encountered leaf node is expanded. Leaving the known regions of the tree in the **simulation** phase a *default policy* is used to simulate the run further down until either a final state is reached or the algorithm runs out of computational budget. The last part is the **backpropagation** phase in which all statistics of the tree are updated according to the results from the current simulation.

With the MCTS algorithm being a framework there is no definitive way to implement the different components. One widely used implementation of MCTS methods is the *Upper Confidence Bound applied to Trees*(UCT)[?]. For action selection within the tree policy, it uses the Bandit-based model of *Upper Confidence Bounds*(UCB) that was first introduced in 2002[?]. Within UCB, in every state, the selection of the next action is modelled as a k-armed bandit from which one arm has to be chosen. UCT applies this concept to trees, inheriting UCBs desirable property of balancing exploration and exploitation.

With the wide application of MCTS in game playing agents, especially in the field of computer Go, research has focused on finding improvements that yield better results in the field of AI game playing, modifying standard algorithms to find the best game move faster.

In this thesis, first the MCTS framework as well as the UCB and UCT algorithms for action selection are introduced in Chapter 2.

Chapters 3 and 4 then summarize different ways to increase the execution speed of Monte Carlo Tree Search methods. Chapter 3 focuses on the field of *All Moves as First*(or shorter AMAF) enhancements, that have proven to be effective in computer go. These modifications are applied to a probabilistic planning scenario and are implemented in the probabilistic planning framework PROST[?].

The family of *All Moves as First* enhancements is based on the idea of treating each action as if it was the first action selected. This technique was first proposed in 1993[?] and has been very successfully applied to game playing in the MCTS context. In this thesis, α-AMAF, Cutoff-AMAF and Rapid Action Value Estimation (RAVE) are implemented and evaluated.

All of the performance enhancements discussed before have in common that they are part of the node selection process. Therfore they only modify the *tree policy*. In Chapter 4 modifications to the backpropagation and simulation steps are introduced. In order to optimize results, modifing the *default policy* generally requires domain specific knowledge. To maintain the domain independence of PROST, sampling techniques in backpropagation enhancements that don't need domain specific knowledge are implemented and benchmarked. The *Move-Average Sampling Technique*(or shorter MAST), an action's value is stored independent of the state it is selected in. The rationale behind this is that actions that have a higher average value might be a good choice. While MAST favors actions that are good on average, *Predicate-Average Sampling Technique* (or shorter PAST) uses the predicates valid in each state to favor actions that performed well in certain settings. MAST is then implemented in PROST and benchmarked against other default policies.

Chapter 5 presents a performance evaluation of the methods introduced in the chapters before.

2 Background

In order to find a solution to a given planning task one has to find a correct set of actions which, after beeing applied to an initial state, yield the desired return. One way to model such a process is using a *Markov Decision process*(or shorter MDP) which describes a decision making process in a setting where the outcome is stochastic. More formally

Definition 1. *A* ***Markov Decision Process(MDP)*** *is a stochastic model for decision problems where the probability of selecting a new state only depends on the current state and chosen action, not on historic decisions.*
An MDP is a 5-tupel $M = \langle V, s_0, A, T, R \rangle$ *with*

- *A set of binary variables* V *inducing the set of states as* $S = 2^V$
- *The initial state* $s_0 \in S$
- *A set of applicable actions* A
- *A transition model* $T : S \times A \times S \to [0; 1]$ *that models the probability of reaching a state* s' *from* s *when action* a *was chosen.*
- *A reward function* $R(s, a)$

A solution to an MDP is a *policy.*

Definition 2. *A* **policy** *maps a state to an action*

$$\pi : S \to A \tag{2.1}$$

One way of solving such a Markov decision process is the Monte Carlo Tree Search Framework that has its roots in the application of Monte Carlo Methods and game theory to board games. Here, the planning and execution are interleaved resulting in an iterative method so that only the next step has to be simulated.
Monte Carlo methods emerged from applied physics and mathematics where they were used to calculate integrals. The idea is to approximate the solution of a problem by randomly sampling it a large number of times. Besides their initial use case these methods are used for a large number of problems from risk management to weather forecasts.

In a *Monte Carlo Tree Search* a decision tree made of nodes is built in memory. In the context of this thesis a node in this tree can be defined as follows

Definition 3. *A MCTS tree for a Markov Decision Problem M is made of **nodes**. In the $i-$ th iteration of an MCTS algorithm a node V is a 5-tupel $V = \langle s, N^{(i)}(s), N^{(i)}(s,a), Q^{(i)}(s,a), p^{(i)} \rangle$ with*

- *A state $s \in S$*
- *A counter $N^{(i)}(s)$ for the number of times the node has been visited in the first i iterations*
- *A counter $N^{(i)}(s,a) \forall a \in A$ for the number of times action a has been selected in state s in the first i iterations*
- *A reward estimate $Q^{(i)}(s,a) \forall a \in A$ for action a in state s after the $i-th$ iteration*
- *A parent action $p^{(i)}$ that was selected in the $(i-1)-th$ node.*

In order to solve the decision problem, in each node, the best action is searched. To do so, an action's *reward* must be estimated. The basic idea of MCTS is that the number of times an action has been sampled in Monte Carlo approach can be used to approximate this move's[**?**] reward or value.
A **Monte Carlo Tree Search** is made of a (asymetrically growing) search tree with four general steps that are recursively applied until a certain limit, in general a time or step constraint, is met at which point the decision tree can be evaluated.

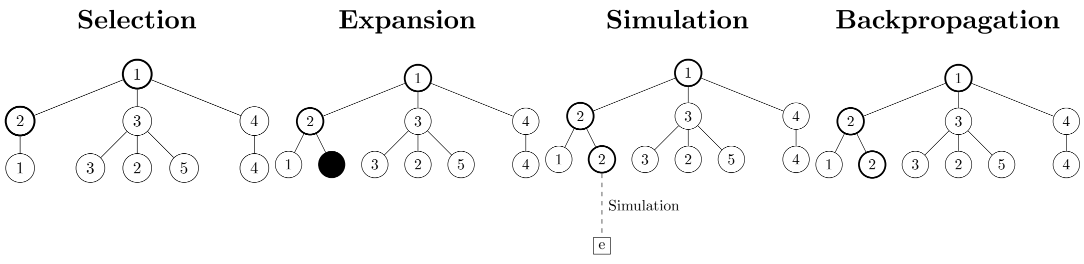


Figure 2.1: The four general steps of a MCTS algorithm

In the **selection** phase, the algorithm iterates through the tree and searches for the next most important action to select.
This node v_e is then expanded in the **expansion** phase and added to the tree based on the actions available in v_e. Starting from node v_e, the **simulation** phase is initialized in which a simulation is used to travers the following nodes until a time or step constraint is met. This estimates a value for this traversal.
In the **backpropagation** step, these values are propagated back into the tree and the estimated value of all nodes on the path to the selected node are updated.
The four phases of a MCTS algorithm are governed by two policies.

Definition 4. *For an MCTS algorithm that solves an MDP M and is represented in the form of nodes V the **tree policy** π^{tree} is used to select (or create) a node from the already* *known part of the search tree. It governs the selection and expansion phase [**?**]. The **tree policy** is stationary and maps the state of a node v to an action a.*

$$\pi^{tree} : V \to a \tag{2.2}$$

It estimates a reward value $Q^{(i)}_{\pi^{tree}}(s, a)$ for a state s and action a under policy π_{tree}.

Definition 5. *A **default policy** $\pi^{default}$ is used to simulate a process upon reaching an unknown part in the tree in order to estimate a value for this path[**?**]. It governs the simulation phase and is non-stationary.*

Algorithm 1 shows the general MCTS algorithm[**?**].

Algorithm 1 General MCTS algorithm

```
function MctsSearch($s_0$)
        create root node $v_0$ with state $s_0$
    while within computational budget do
        $v_l \leftarrow$ TreePolicy($v_0$)
        $\Delta \leftarrow$ DefaultPolicy($s(v_l)$)
        Backup($v_l, \Delta$)
    end while
    return a(BestChild($v_0$))
end function
```

Originally used to approximate integrals in statistical physics, Abramson[**?**] suggested a method of approximating an action's estimated reward, its Q-value using this method.

Definition 6. *The **Q-value** describes an action's reward based on the state it is executed on. An **optimal policy** always maps the action with the highest Q-value. Another policy π estimates this reward value in a node v with state s. It's value in the $i-th$ iteration of the algorithm can be approximated using*

$$Q^{(i)}_{\pi}(s, a) = \frac{1}{N^{(i)}(s, a)} \sum_{i=1}^{N^{(i)}(s)} \mathbb{I}^{(i)}(s, a) z^{(i)}(s) \tag{2.3}$$

where $z^{(i)}(s)$ is the result of the i-th simulation played out from s and $\mathbb{I}^{(i)}(s, a)$ is defined as

$$\mathbb{I}^{(i)}(s, a) = \begin{cases} 1, & \textit{if } \pi^{tree}(v) = a \\ 0, & \textit{otherwise} \end{cases} \tag{2.4}$$

Choosing an action randomly as a default policy this lazy Monte Carlo Tree search can already be applied to real life examples such as Bridge or Scrabble, however there are examples where influencing the action that will be selected based on past experience is helpful. This leads to more advanced selection techniques such as UCT that will be described in the following sections.
If it is known that the tree policy converges against the optimal policy, an additional benefit of this method is that MCTS algorithms are "statistical anytime algorithms for which more computing power generally leads to better performance"[**?**]. This means that results can be increased by simply allowing for more computing time.

In itself, Monte Carlo Tree Search is just a framework for many different kinds of selection algorithms that base around two concepts. First, an actions value might be approximated using random simulation, and second these Q-values might be used to guide a policy towards a good strategy. This means that a policy learns from previous results and improves over time. Therefore MCTS does not build full game trees such as a simple BFS approach but rather a partial tree whoms exploration is guided using previous results as an estimate of an action's value. The estimates of these "best moves" become more accurate as the tree grows.
In the following sections some basic algorithms shall be defined and explained.

2.1 Upper Confidence Bound for Trees(UCT)

One possible algorithm for Monte Carlo Tree Search algorithms is the UCT algorithm[**?**] first proposed in 2006.
The major problem for a tree policy is to balance the exploitation of already known good actions with the exploration of actions that might not be the best choice in this node but may yield a higher reward in the future.
To tackle this exploitation-exploration dilemma UCT introduces the concept of *Upper Confidence Bound*(UCB) methods into the MCTS algorithm.

2.1.1 Bandit-Based Methods and Upper Confidence Bounds(UCB)

In a bandit problem, one has to choose from K actions in every subsequent state. This scenario is comparable to choosing an arm among the K arms of a multi-armed bandit, hence the name. The goal is to maximise the cumulative reward from the chosen arm over all states. In theory, this is done by always choosing the action that yields the biggest reward. In practice however, the reward's distribution for each action is unknown.
An estimate which arm $X_{j,n}$ among the $X_{i,n} 1 \leq i \leq K, n \geq 1$ possible arms should be chosen might be drawn from past experience upon an arm's reward. However, there might be other arms that offer an even higher reward to the player than the one currently believed to be the best. This is also known as the *exploitation-exploration dilemma*[**?**] where one has to balance between exploiting the currently believed to be optimal action with exploring other actions that might turn out to be more rewarding in subsequent playouts.
In UCB an arms reward is estimated by

$$UCB1(a_j) = \underbrace{\overline{X}_j}_{\text{exploitation}} + \underbrace{\sqrt{\frac{2\ \ln n}{n_j}}}_{\text{exploration}} \tag{2.5}$$

and the policy chooses the arm X_j, and therfor action a_j, where

$$UCB1 = \max\{UCB1(a_j)\} \qquad \forall j \tag{2.6}$$

Here, $\overline{X}_j$ is the average reward from arm j, n_j is the number of times arm j was played and n is the overall number of playouts.

As indicated in the equation, the two parts balance exploration and exploitation. Choosing the action that has the maximal average return emphasises the past knowledge while the exploration term emphasises arms that haven't been selected previously.

2.1.2 UCT algorithm

The UCB implementation has the desirable effect of balancing exploitation and exploration. Especially its property of being in range of the best possible bound on the growth of regret makes UCB a fitting algorithm for MCTS problems[**?**]. *Upper Confidence Bounds*(UCT) takes this approach into search trees by treating the act of choosing a child node as a multi-armed bandit problem where A is the set of K actions available. The expected reward $\overline{X}_j$ from the UCB calculation is taken from the reward approximated by Monte-Carlo simulations and thus is a random variable with unknown distribution as stated in a multi-armed bandit problem.
In the $i-th$ iteration of the algorithm, estimate the UCT score of a child node v_j with the parent node v_l as

$$UCT(v_l, v_j) = Q^{(i)}(s_l, a_j) + 2\,C_p\sqrt{\frac{2\,\ln N^{(i)}(s_l)}{N^{(i+1)}(s_j)}} \tag{2.7}$$

Where a_j is the action leading from node v_l to node v_j and $C_p > 0$ a constant factor. This factor can be used to control the impact of the exploration on node selection. A standard value of $C_p = \frac{1}{\sqrt{2}}$ can be found in literature.
From parent node v_l select the child node v^* that maximizes

$$v^* = \max_{v_j}\{UCT(n_l, n_j)\} \tag{2.8}$$

Since UCT is the most-used selection method in the context of MCTS, it will be used as the baseline for evaluation of enhancements.

2.2 Default Policy

In order to estimate the outcome of a run beyond the known state of the tree a default policy is used to simulate the run upon expansion.

2.2.1 Random Walk

The basic UCT implementation proposed a random walk. Here a random action is selected. While this method is easy to implement it does not guide the default policy towards more promising actions.

2.2.2 IDS

A more sophisticated approach is *Iterative Deepening Search*(or shorter IDS). Here, a limited depth first search is conducted and the search depth is incremented in every step.

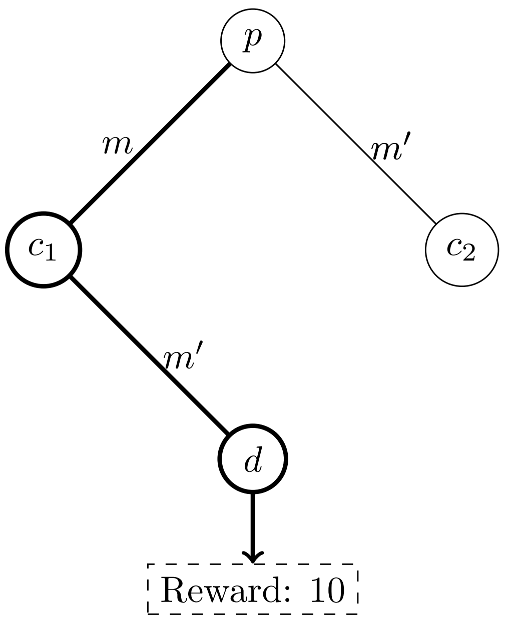
3 All Moves as First (AMAF) Enhancements

The idea of *All Moves as First* heuristics have their roots in MCTS algorithms for computer Go. They originated from history heuristics that try to leverage history information to enhance action selection on the *Tree-level* as well as on the *tree-playout level* that tries to improve the simulation using historic information. The general idea of history heuristics is to approximate an initial action value from history.

For All Moves as First enhancements, the statistics of all actions selected during the simulation process are treated as if they were the first action applied. This means that all actions selected during selection and simulation are treated equally as if they were previously played out.

A reward estimate generated from the AMAF heuristic is generally referred to as the *AMAF score A*.

To explain how this AMAF score A is calculated consider the following tree:



UCT

Before this run	
$Q^{(i)}(s_p, m)$	5
$N^{(i)}(s_p)$	2
$N^{(i)}(s_p, m)$	1
After this run	
$Q^{(i+1)}(s_p, m)$	15
$N^{(i+1)}(s_p)$	3
$N^{(i+1)}(s_p, m)$	2

In this example, the available actions are (m, m') and the different states are (p, c_1, c_2, d).

After some trials the tree policy chooses the bold printed playout path $p \rightarrow c_1 \rightarrow d$ by selecting actions m, m'. For normal UCT backup in Node p with state s_p, the update would increment the number of visits by 1 and add the reward from this trial as shown in the table to the right. AMAF keeps an additional table and would additionally update the $Q^{(i+1)}_{AMAF}(s_p, m')$ pair with the reward from this trial, even tough it was not the action directly selected in p but it was selected in a subsequent move. Also the $N_{AMAF}(s_p, m')$ counter that counts the number of times the AMAF score for action m' in state s_p has been updated is incremented.
All Moves as First enhancements have proven to be very efficient in the context of computer Go. The following sections outline some AMAF enhancements that will be benchmarked against pure UCT.

3.1 α-AMAF

When selecting actions, one could solely rely on the *AMAF score* for selecting the next action. However the UCT score still carries value so α-AMAF seeks to combine the two approaches.
The total score for an action is a linear combination of *AMAF score* and UCT score with

$$SCT = \alpha A + (1 - \alpha)U \tag{3.1}$$

Here, U is the UCT score and A the *AMAF-score*. α is a weighting factor that determines which part should influence the score more.
In the example above, the UCT score for action m' in the node p with state s_p would be:

$$U = Q^{(i)}(s_p, m) + 2\,C_p \sqrt{\frac{\ln(N^{(i)}(s_p))}{N^{(i+1)}(s_{c1})}} \tag{3.2}$$

In α-AMAF this value would then be biased with

$$A = \frac{1}{N_{AMAF}(s_p, m')}\, Q_{AMAF}(s_p, m') \tag{3.3}$$

using the linear weight function introduced above.

3.2 Cutoff-AMAF

Cutoff-AMAF is closely related to α-AMAF. The idea is to initialize the tree with some AMAF data but rely on the more accurate UCT data later in the search process. Therefore, the score α-AMAF score is used for the first k simulations after which only the UCT score is used. This gives

$$SCT = \begin{cases} \alpha A + (1 - \alpha)U & \text{, for } i \leq k \\ U & \text{else} \end{cases} \tag{3.4}$$

where i is the current episode obtained from the search node.

3.3 Rapid Action Value Estimation(RAVE)

Rapid Action Value Estimation, or shorter RAVE, was first introduced in a 2007 paper [?]. The aim of RAVE is to exploit the already exisiting structure of the search tree to get a rapid evaluation of an actio's cost and reward.
A general bottleneck in UCT based MCTS algorithms is, that an action a from a state s has to be sampled multiple times in order to get reliable values. With large action spaces this can lead to a slow down.
As opposed to UCT, where an action is valued based on the number of times it was selected, RAVE averages the return of all episodes where a was selected at any subsequent time[?].
RAVE seeks to combine this approach with the idea of α-AMAF. While the parameter α in $\alpha-$AMAF is static, RAVE uses a calculated value for α that depends on the number of times a node is visited. This is a softer approach to cutoff AMAF. Instead of only using the UCT score after a certain amount of trials, RAVE favors the UCT score for nodes that have been visited multiple times while using the quicker updated but more variant AMAF score A for less visited nodes.
First introduce a new paramter $V > 0$. Then calculate the value of α according to

$$\alpha = \max\left\{0, \frac{V - v(n)}{V}\right\} \tag{3.5}$$

The parameter V yields the number of visits to a node befor the RAVE values are not considered anymore.

4 Backpropagation Enhancements

In standard MCTS, the default policy randomly selects an action a among all available options. The advantage of using such an approach is the domain independence of the technique. There is no domain knowledge required and the randomness ensures that all states are covered, given enough trials. However, this is not realistic as a human player in, for example, a board game would not randomly select an action but rather use knowledge to determine a promising action. Incorporating offline knowledge, action selections based on databases or expert games, is beyond the scope of this thesis, but rather this will focus on using online knowledge gained from self-play and learning.

4.1 Move-Average Sampling Technique (MAST)

Move-Average Sampling Technique (or shorter MAST) has a similar approach to history heuristics. The idea is, that moves or actions that were good previously might be good in another state. MAST uses this idea and keeps a table in which the average reward $Q(a)$ independent of state is stored.

In subsequent moves, the default policy uses a Gibbs distribution to bias selection using these MAST values. It selects an action according to the probability

$$P(a) = \frac{e^{\frac{Q(a)}{\tau}}}{\sum\limits_{b \in \mathcal{A}}^{n} e^{\frac{Q(b)}{\tau}}} \tag{4.1}$$

Here, $P(a)$ is the probability of action a being selected and $\mathcal{A}$ is the set of actions available. $Q(a)$ is an action's average return value over previous playouts and τ is a stretching parameter of the distribution and controls how much influence the $Q(a)$ value should have on the action selection. $\tau \to 0$ stretches the distribution while higher values of τ make it converge towards a uniform action selection. This selection enhancement was first introduced for CadiaPlayer[**?**] and biases the selection towards moves that were good *on average*.

As an example consider a simple board game where a player has 5 moves on the board to come as close to a goal field.The player can move in one of the four directions *up*,*down*,*left* and *right*. Consider the game situation in Figure 4.1.

As a reward the number of steps needed to reach the goal is used. A situation that needs

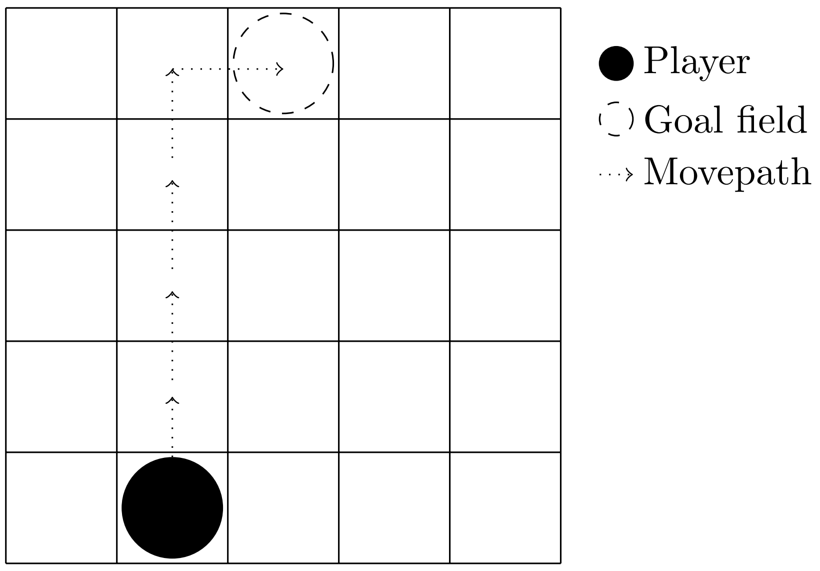
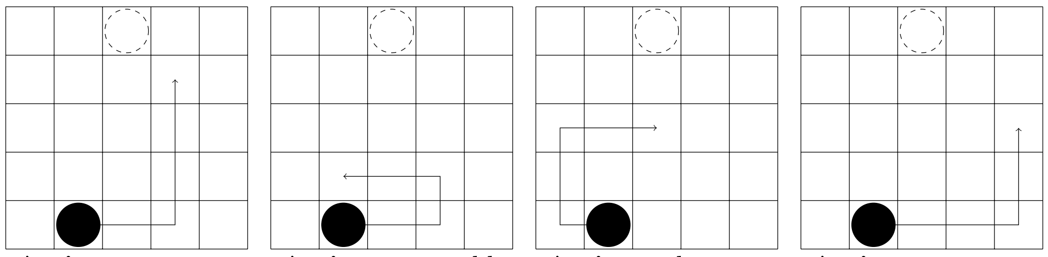


Figure 4.1: Sample game situation for the example board game

one more move to get to the finish would have a reward of $5 - 1 = 4$ and would therefor be considered a good move. It's easily visible that either moving *up* four times and then moving *right* or moving *right* and then moving *up* four times is the best way.
A random walk would choose this ideal path with a probability of

$$p_{\text{ideal}} = (p(up)^4 \cdot p(right)) + (p(right) \cdot p(up)^4) \tag{4.2}$$
$$= 2 \cdot \left(\frac{1}{4}\right)^5 \tag{4.3}$$

Figure 4.1 shows four simulated games.



Actions: *r,r,u,u,u*	**Actions:** *r,r,u,l,l*	**Actions:** *l,u,u,r,r*	**Actions:** *r,r,r,u,u*
$Q(r) = 1; N(r) = 2$	$Q(r) = 2; N(r) = 4$	$Q(r) = 7; N(r) = 6$	$Q(r) = 7; N(r) = 9$
$Q(u) = 6; N(u) = 3$	$Q(u) = 7; N(u) = 4$	$Q(u) = 8; N(u) = 6$	$Q(u) = 9; N(u) = 8$
	$Q(l) = 3; N(l) = 2$	$Q(l) = 2; N(l) = 3$	$Q(l) = 2; N(l) = 3$

Figure 4.2: Four sample move paths with their calculated MAST values $Q(a)$ and $N(a)$ the number of times an action a has been applied

After the four simulations shown in figure 4.1 action u would have a MAST value of $\frac{9}{8}$ while action r would have a value of $\frac{7}{9}$ and l a value of $\frac{2}{3}$. For further simulations the selection would be biased towards selecting the upward move since this move has previously led to a better reward resulting in a more goal focused default policy.

4.2 Predicate-Average Sampling Technique (PAST)

MAST has the general shortcoming of treating an action's value without the context it was applied upon. Consider the previously introduced walking game. Given the scenario in figure 4.3. After four moves the player is in the gray position.

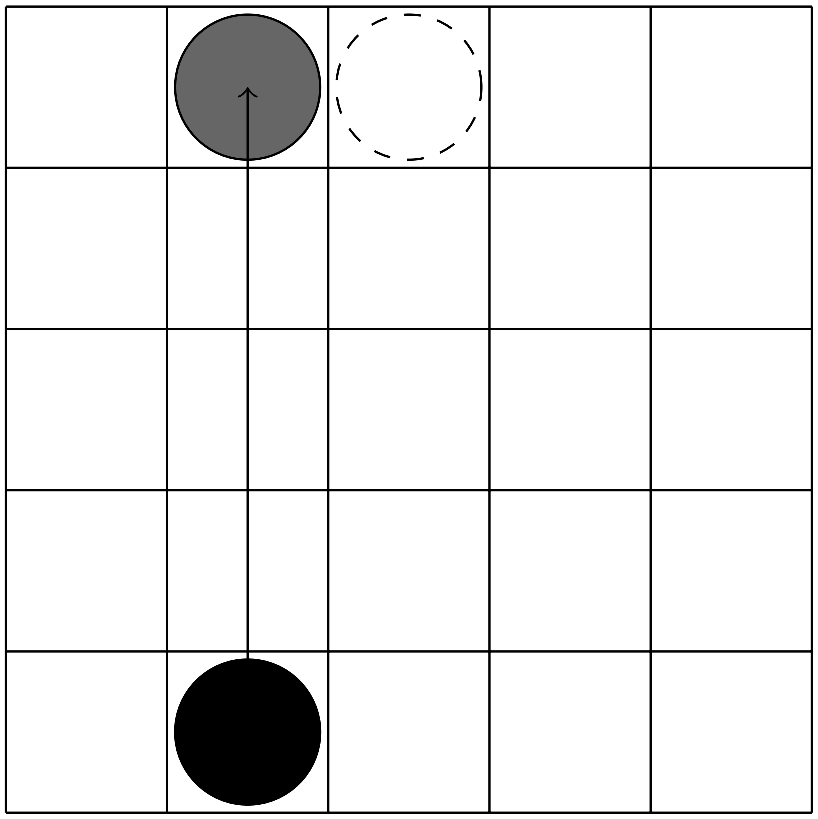


Figure 4.3: Sample situation after four moves with one move left

Since all actions are available even if they don't make sense, moving upward will still be considered a very good action and a default policy that is influenced by MAST will therefore be biased towards selecting the upward action. However, when reaching the upper bound, the action simply results in the player staying at his current position and therefore wasting a move.

Predicate-Average Sampling Technique (or shorter PAST) is based on MAST but uses a list of predicates that hold true in a certain state to describe it. It maintains $Q_p(p, a)$ that holds the average returns for an action given a certain predicate. These values are updated on backpropagation as well and the default policy biases actions using a Gibbs distribution.

A threshold parameter T indicates how many predicates in a state have to match in order for them to be considered similar enough. Since the predicates that hold true in a certain state offer some context about the state itself PAST biases the selection towards actions that perform better in a certain context.

In the above example consider a simple predicate set of four binary variables (m_u, m_d, m_l, m_r) that describe if the player can move in the direction specified in the index. With the m_u

flag being set to false in the scenario given above, PAST would not bias the action selection towards selecting the u action, biasing it towards using the option with the second best estimated return value the r action which actually results in reaching the goal state.
A general problem with this technique is that the one important predicate(The n predicate in the example) in two states is different, but the predicates are still considered similar and therefore matched.

5 Evaluation

As a basis for the evaluation the problems from the IPPC were used. IPPC measures the performance of a probabilistic planner with its own IPPC score. The score describes the performance of a probablistic planning algorithm on a scale from 0 to 1. For benchmarking each of the 12 IPPC problems were simulated 100 times and the result is then averaged into the IPPC score for the specific parameters.

5.1 AMAF Evaluation

In α-AMAF the score is combined with UCT score using a weighting factor $\alpha \in [0; 1]$. Figure 5.1 shows different values for α against the IPPC score over all domains. The data was captured using the strong IDS heuristic as default policy.

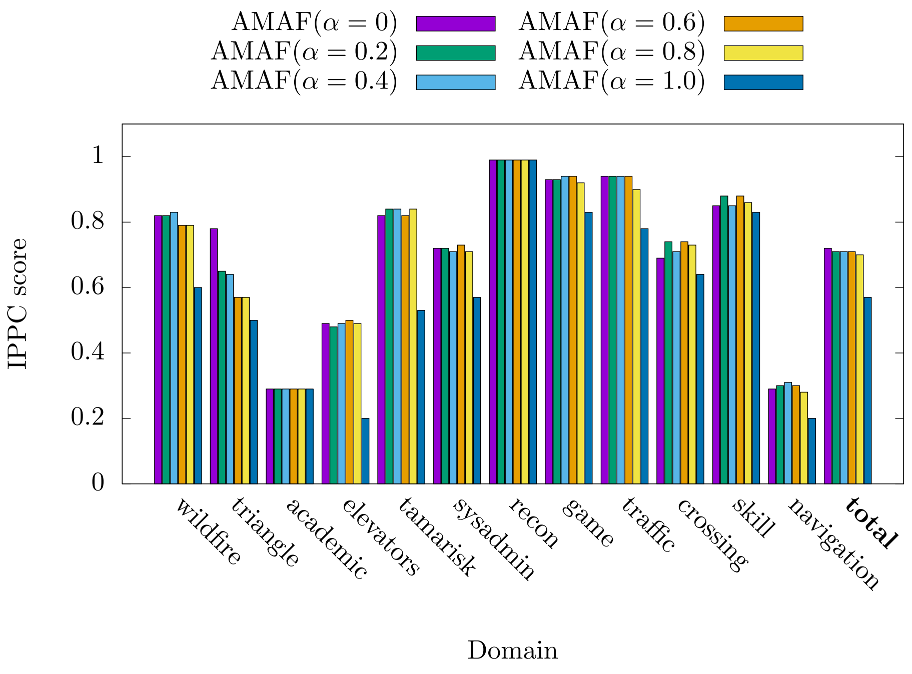


Figure 5.1: IPPC scores for different α values

Here, an α value of 0 describes plain UCT action selection while an α value of 1 is using only the AMAF score for action selection. The closer the α values comes to 1 the more aggressive action selection is biased towards selecting an action that performed well in the context of the AMAF score. With an IPPC score of 0.57 using just the AMAF score for action selection performs significantly worse than incorporating the UCT score. Given that the UCT score becomes more and more reliable the more trials have passed, this is no surprise. However, incorporating the online knowledge into the planning also does not show an improvement over using just plain UCT. This is can be seen by the fact that the IPPC score stays the same for all the different α weights used. An explanation for this is presented in chapter 5.3.

Since the AMAF score is not as reliable as the UCT score, Cutoff-AMAF introduces a cutoff parameter of trials after which only the UCT score is used. Figure 5.2 shows the IPPC score of Cutoff-AMAF with an α value of 0.6 and different cutoff parameters K in comparison with plain α-AMAF with the same α and an unmodified UCT with IDS as a default policy.

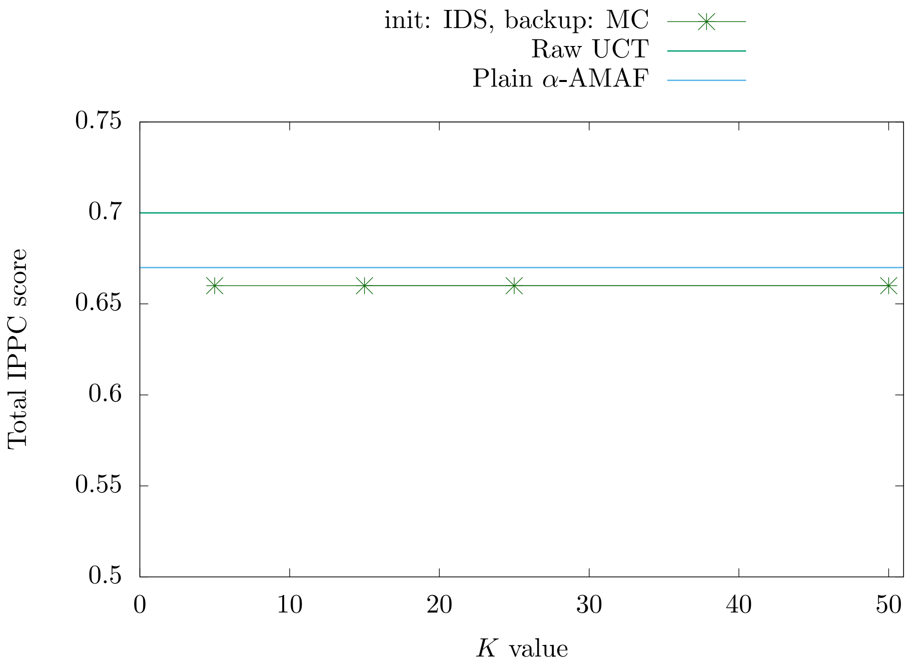


Figure 5.2: Different Cutoff parameters

Cutoff-AMAF is performing worse than raw UCT in every domain and the Cutoff parameter seems to have no influence on the performance of AMAF biasing. The experiment shows a similar performance for a plain α-AMAF with no cutoff parameter. The advantage over Cutoff-AMAF is not significant and α-AMAF is still performing worse than UCT with no modifications.

5.2 RAVE Evaluation

For a more dynamic approach to Cutoff AMAF, RAVE was introduced.

Figure 5.3 shows the IPPC score for all domains for several Cutoff scores V in the range of 5 to 50. All experiments were made with the informed IDS heuristic as a default policy.

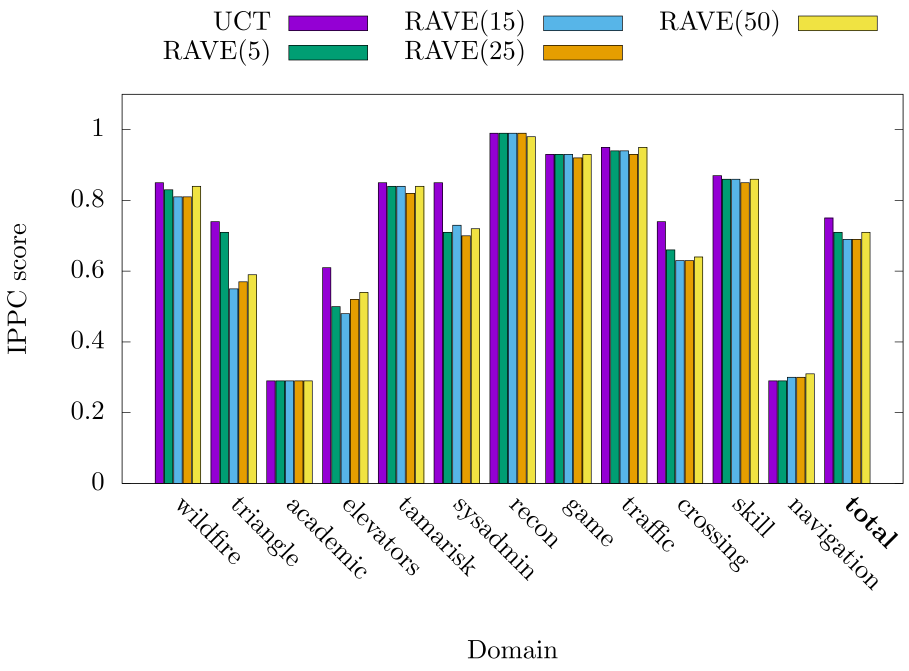


Figure 5.3: RAVE performance for different V values

The figure shows that RAVE is either on par or worse than plain UCT. Some domains like elevators even show huge discrepancies of 0.61 for UCT and 0.54 for the best performing RAVE implementation. A reasoning for this is explained in section 5.3. One exception is the navigation domain where all RAVE implementations are outperforming unmodified UCT.

5.2.1 Influence of Default Policy

From a conceptual point one might expect that RAVE becomes more relevant in scenarios where the UCT value is not as reliable. Previous experiments where carried out with the rather well-informed IDS heuristic. For RAVE the experiments were run with $V = 20$ as a cutoff parameter in RAVE and $\alpha = 0.4$ as the weighting factor for AMAF.

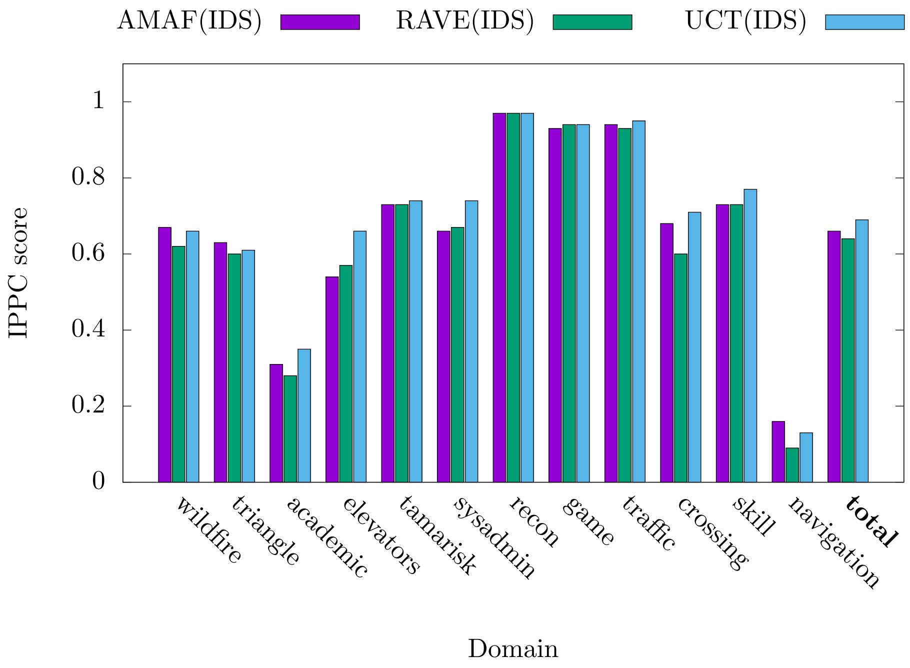


Figure 5.4: Performance with IDS as a default policy

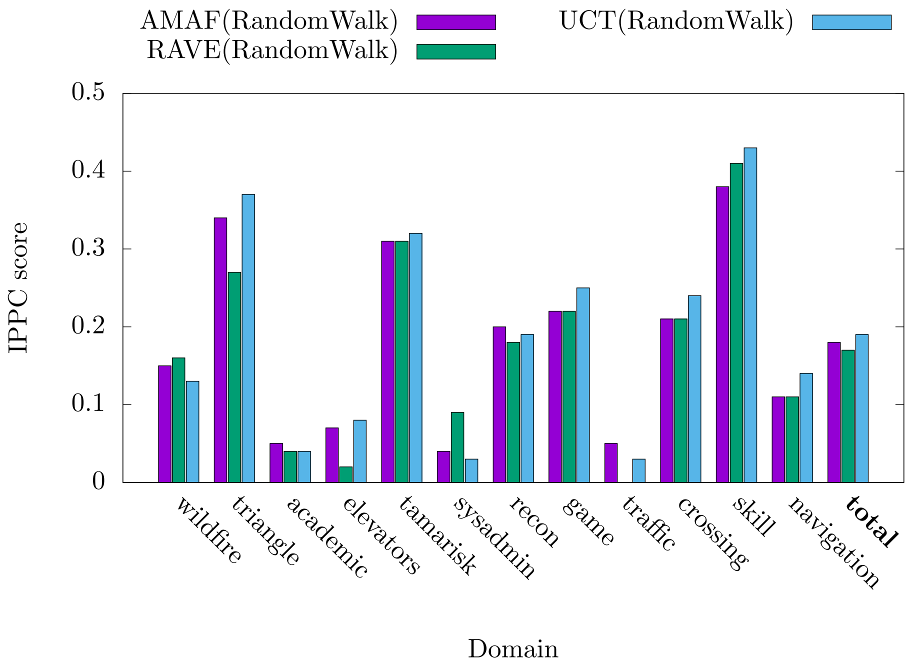


Figure 5.5: Performance with RandomWalk as a default policy

Figure 5.2.1 shows the IPPC score over all domains for IDS as a heuristic and figure 5.2.1 shows the same domains for a Random walk as a default policy.
While the informed IDS heuristic outperforms the uninformed random walk there is no

significant score increase for using RAVE and AMAF. Also, AMAF and RAVE perform on par. This means that deriving online knowledge is not performing significantly better with an uninformed default policy compared to an informed one.

5.3 Analysis of performance

The experiments presented in the previous sections show that AMAF enhanced action selections perform either on par or slightly worse than their non-biased counterpart UCT. This is in contrast to literature(**?**) where these heuristics were applied with great effect to game playing software, especially in the field of computer go.
A reason for this might be the fact that the rddl problem description *PROST* uses does not incorporate any preconditions and conditional effects. As a simple example consider a robot that can move in all four cardinal directions. While in game playing states and their actions might yield information about inapplicable actions such as an invalid westward move if the robot is already at the western bound, *PROST* still considers moving westward as a valid move that just yields a nop. This means that actions that might be good in a state and thus have been updated in the AMAF score won't necessarily fit the exact same state.

5.4 Predicate Rapid Action Value Estimation

As a workaround we introduce *Predicate Rapid Action Value Estimation*(or shorter PRAVE). In PRAVE, instead of updating the Q_{RAVE} value for a state-action pair, PRAVE uses predicates and actions, thus combining the idea of *Predicate-Average Sampling* with RAVE. In action selection instead of biasing with the RAVE value for this state-action pair use

$$Q_{PRAVE}(s,a) = \frac{1}{N}\sum_{q\in P} Q_{RAVE}(q,a) \tag{5.1}$$

here, P is the set of all states where the predicates match the predicates of s up to a certain threshold constant T and N is the number of states in P. So instead of using only the RAVE values where both state and action match, *PRAVE* uses all states where the evaluated action has been applied and the predicates are similar enough suggesting a similar situation.
To implement this a lookup table holds all predicate-state relations to quickly find all matching states and then lookup their *RAVE* value.

5.4.1 Evaluation of PRAVE

Figure 5.4.1 shows the results for PRAVE with a cutoff parameter $V = 25$ and a threshold value of 0.8. This means that 80 percent of a nodes predicates had to match for PRAVE to consider them similar. For RAVE a cutoff parameter of $V = 25$ was set as well.
Figure 5.4.1 shows that PRAVE is slightly outperforming pure RAVE, thus indicating that preconditions as well as conditional effects play a role in the poor performance of *All moves as first* enhancements. However, the standard UCT action selection is still performing better.

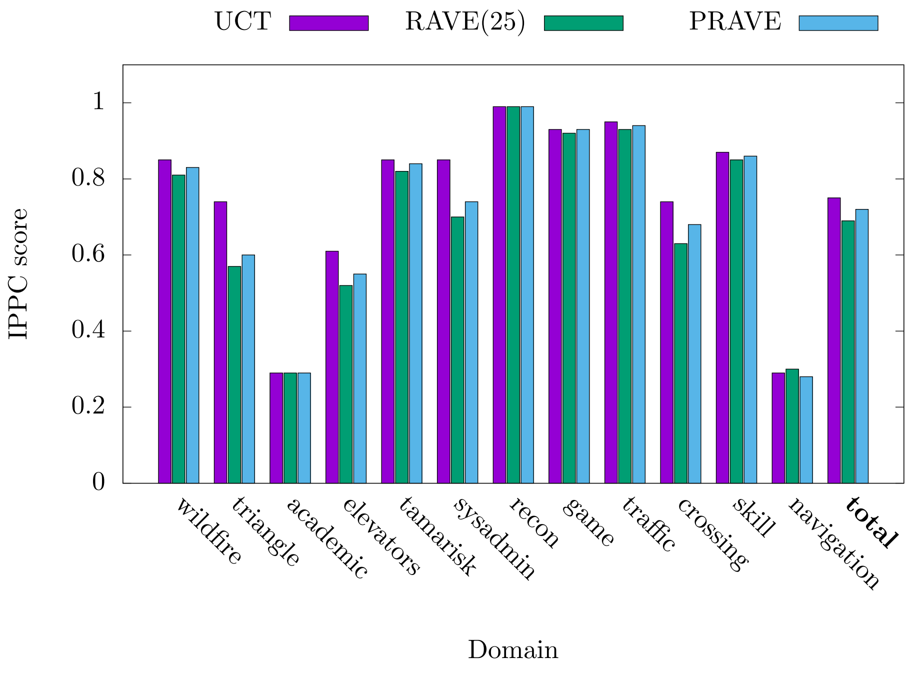


Figure 5.6: PRAVE in comparison to RAVE and UCT

5.4.2 Summary of Tree Policy Enhancement Performance

Figure 5.4.2 shows the best performing *All moves as first* enhancements that were presented in this thesis.

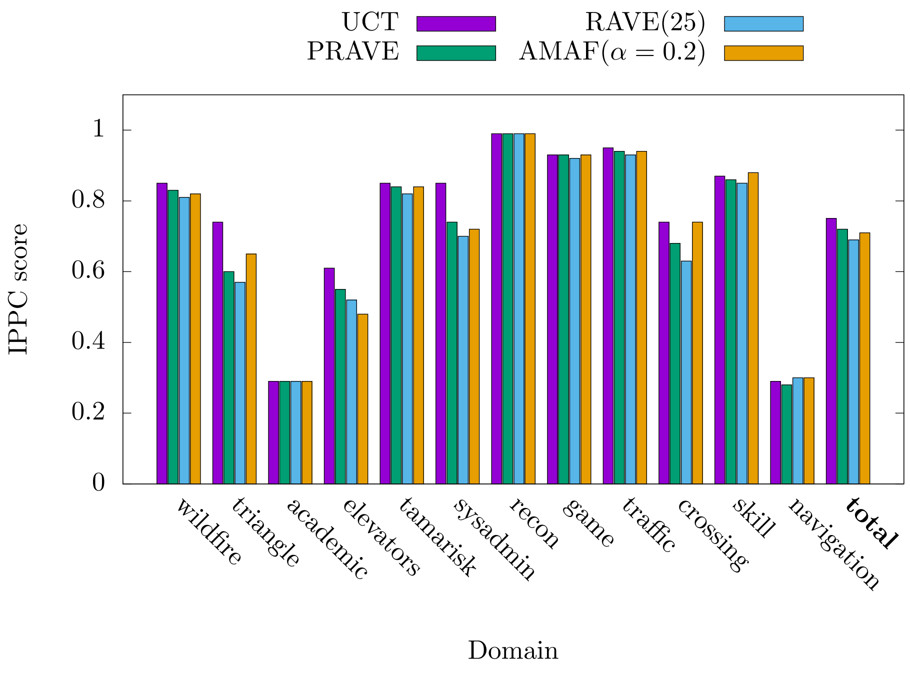


Figure 5.7: The best performing AMAF enhancements and UCT

The plot shows that there are two domains, *navigation* and *skill* where α-AMAF outperforms

UCT. For all other domains, as well as on average, UCT performes better.
The best-working α parameter for α-AMAF is the lowest tested parameter with $\alpha = 0.2$. This shows that less influence from α-AMAF results in a better performance.
PRAVE is capable of outperforming standard RAVE in all but the *skill* domain. This can be explained with the fact that PRAVE is less prone to preconditional effects than standard RAVE.

5.5 Move-Average Sampling Technique

To enhance execution speed it is also useful to incorporate online knowledge into the simulation part of MCTS.
Figure 5.5 shows the performance of MAST against the uninformed random walk.

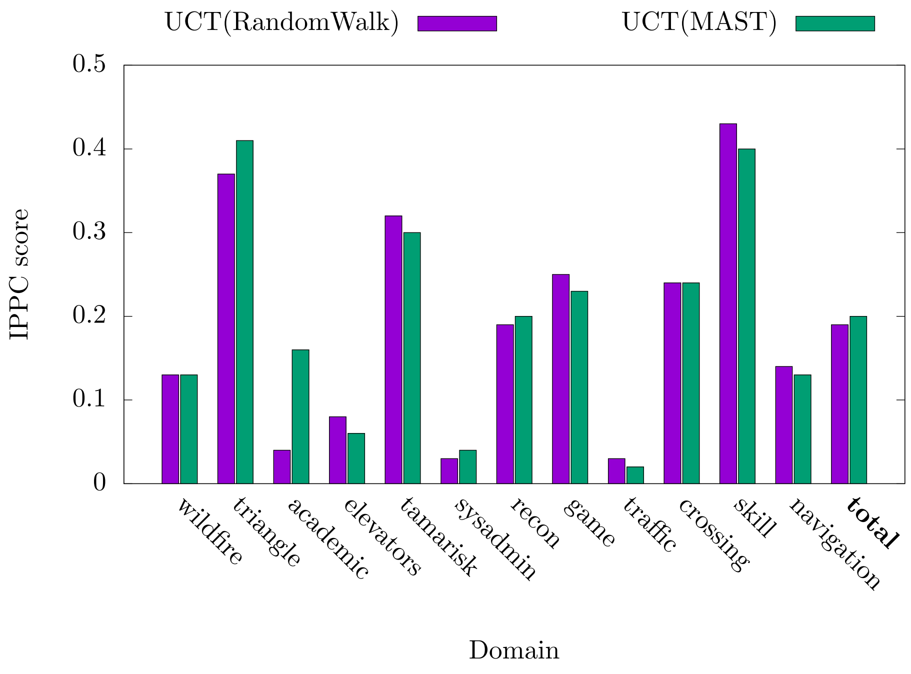


Figure 5.8: MAST performance against random walk using UCT as the action selection

MAST can outperform random walk as a simulation policy for domains like *triangle* or *academic*. However the overall benefit of MAST over a random walk is not as significant as one might expect and MAST is outperformed by the more sophisticated Iterative Deepening Search(IDS) default policy that was used for most of the previous experiments.
A reason for MASTs performance migth again be the more general description of moves in the problems PROST is acting upon.

6 Conclusion

6.1 Summary

In this thesis different enhancements for the tree and default policy that performed well in game playing were discussed and benchmarked in the context of probabilistic planning.
For tree policy or action selection enhancements the three *All moves as First* heuristics α-AMAF, Cutoff AMAF and Rapid Action Value Estimation were benchmarked against UCT. The idea of AMAF heuristics is to update not only the nodes on the path that were chosen but also for the actions leading to the sibling nodes of the actual node played out. This gives an estimate of a state-action pair's value for early action selection where the UCT score is not yet reliable. The results show that all enhancements do not yield an improvement compared to standard UCT. An explanation for this can be found in the way the domains PROST is finding a plan for, are described. These do not include preconditions and conditional effects leading to a more general state description where the AMAF score is not accurate. To overcome this limitation RAVE was combined with the Predicate-Average Sampling Technique to form the new *Predicate Rapid Action Value Estimation.* While PRAVE performed slightly better than RAVE or α-AMAF it was still performing worse than standard UCT.
For default policy or initialization enhancements Move- and Predicate-Average Sampling Techniques were introduced and MAST was benchmarked against random walk and IDS. In Move- or Predicate-Average Sampling Techniques an action's value is kept book of independant of state. MAST uses only the action while PAST also incorporates a states predicates to offer some contextual dependence. This gained knowledge about an action can then be used to bias the default policy towards selecting more promising actions.
MAST performed slightly better than a pure random walk but was not able to compete against IDS as a default policy.

6.2 Future Work

Besides the backpropagation enhancements presented in this thesis, CadiaPlayer also employs the *Feature-to-Action Sampling Technique*(or shorter FAST)[**?**]. CadiaPlayer is a general game playing AI that employs MCTS techniques for various board games. FAST parses

the game description and uses template matching to extract features such as piece types and cell locations for board games. After learning the importance of features, a value function $V(s)$ is then estimated.
FAST offers an interesting way to enhance the value estimation beyond pure state-based approaches such as MAST or simple pattern approaches such as PAST. Additionally, CadiaPlayer combined MAST and FAST together[**?**] relying on feature-detected outcome biasing only when features were active in the state. For CadiaPlayer the usage of RAVE and a combination of MAST and FAST resulted in a 1.84 % improvement in game winning ratio against an implementation using just RAVE and MAST[**?**].
This might be an interesting improvement for probablistic planning considering the acting patterns available in some domains and the missing preconditions and conditional effects.

Bibliography

Abramson, B. (1990). Expected-outcome: a general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):182–193.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1 – 43.

Bruegmann, B. (1993). Monte carlo go.

Finnsson, H. and Björnsson, Y. (2011). Cadiaplayer: Search-control techniques. *KI - Künstliche Intelligenz*, 25(1):9–16.

Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in uct. In *Proceedings of the 24th International Conference on Machine Learning*, pages 273–280, New York, NY, USA. ACM.

Keller, T. and Eyerich, P. (2012). Prost: Probabilistic planning based on uct. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling.*

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg. Springer-Verlag.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

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

Author — Autor
Marcel Neidinger

Matriculation number — Matrikelnummer
2014-050-280

Title of work — Titel der Arbeit
Online Knowledge Enhancements for Monte Carlo Tree Search in Probabilistic Planning

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, 09/02/2017

[illegible]

Signature — Unterschrift