University of Basel

Enhancing Prost with Abstraction of State-Action Pairs

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

Tim Steindel
tim.steindel@stud.unibas.ch
2014-050-389

02/11/2017

Acknowledgments

I would like to thank Dr. Thomas Keller for the supervision and all the very helpful support. I would also like to thank Prof. Dr. Helmert very much for giving me the opportunity to write this thesis in his research group. Special thanks goes to Christin Berger, Severin Wyss, Jonathan Aellen and Marvin Buff for proof reading and giving helpful advice. Calculations were performed at sciCORE (http://scicore.unibas.ch/) scientific computing core facility at University of Basel.

Abstract

A planner tries to produce a policy that leads to a desired goal given the available range of actions and an initial state. A traditional approach for an algorithm is to use abstraction. In this thesis we implement the algorithm described in the ASAP-UCT paper: Abstraction of State-Action Pairs in UCT by Ankit Anand, Aditya Grover, Mausam and Parag Singla [1].
The algorithm combines state and state-action abstraction with a UCT-algorithm. We come to the conclusion that the algorithm needs to be improved because the abstraction of action-state often cannot detect a similarity that a reasonable action abstraction could find.

Table of Contents

1 Introduction

The idea of robots in everyday life has been a dream for quite some time in our society. Be it autonomous cars or just the robotic vacuum cleaner. However, not only the hardware is problematic, but also the intelligence behind these robots. A robot vacuum cleaner, for example, which only cleans a room that is already clean, is just as effective as one which permanently switches rooms and never cleans a room completely. Thus it needs to learn which actions are good in its current state. Furthermore, our environment is dynamic and the robot has not the power to determine the outcomes without doubt or has not all the information. For this kind of problem planner exists. A planner tries to produce a policy that leads to a desired goal given the available range of actions and an initial state.

An example for a planning under uncertainty problem is the academic advising domain. Let us assume we have a student who wants to graduate. In order to accomplished this, he needs to pass several courses. These courses can have other courses as a precondition or overlap in content. Furthermore the student can only take one course at a given time. Therefore, the student has several policies to reach his goal and the question is now, how to determine an optimal solution. The problem for our student is to manage the cost respectively the reward and also the probability to pass the courses. It would be very profitable for the student to write his state examination in his first semester but his probability to pass would be nearly zero. Hence the student needs to accumulate first costs, before he can get to the point where he gets his reward (his graduation).

For a planner this task is difficult to simulate because there is no immediate reward to gain from taking a course. Therefore it is harder to know if the current path is promising. The number of solutions grow nonlinear with the number of courses, thus it is not advisable to use a brute force algorithm.

A traditional approach for an algorithm is to use abstraction [2]. The theory behind abstraction is that we calculate equivalence classes, so that each class in an equivalence class has the same value and thus reduces the state space.
Another approach are Dynamic Programming algorithms [3], where we break our problem

in subproblems and save the intermediate results. On one hand, those algorithms have the advantage of leading to optimal solution. On the other hand, they require a lot of computer memory.
An alternative approach is to use a Monte-Carlo-Tree Search(MCTS)[4] algorithm where we have only a part of the tree and try to find promising actions and expand the tree in these directions. They have the advantage that they can be stopped anytime and can give a promising action back. A well-known algorithm is for example the UCT algorithm [5].

In this paper, we combine a UCT algorithm with abstraction of states. Furthermore we also compute the equivalence classes of state-action pairs similar to the work of Anand et al.[1]. First of all we define the theoretical background needed like the Markov decision process [6], the search tree and UCT. We then implement ASAP-UCT (Abstraction-State-Action-Pair) within the UCT-based MDP solver PROST [7] as another approach for an algorithm. Afterwards we compare our result with the UCT algorithm [5] and with the original PROST. We conclude the paper with a short summary.

2 Background

A problem like the academic problem example can be modeled by a Markov decision process (MDP)[6]. The useful property of a MDP is that a new state s' is only dependent of its previous state s and an action a, it is independent of the predecessor of s. In addition we require that MDPs are dead-end free.

Definition 1. *A Markov decision process (MDP) is defined as a 6-tuple* $\langle S, A, T, R, s_0, H\rangle$ *where:*

- S *is a finite set of states,*
- A *is a finite set of actions,*
- T*:* $S \times A \times S \rightarrow [0,1]$ *is a probability distribution that gives the probability* $T(s,a,s')$ *that applying action a in s leads to s',*
- $R : S \times A \rightarrow \mathbb{R}$ *is the reward function,*
- $s_0 \in S$ *is the initial state and*
- $H \in \mathbb{N}$ *is the finite horizon.*

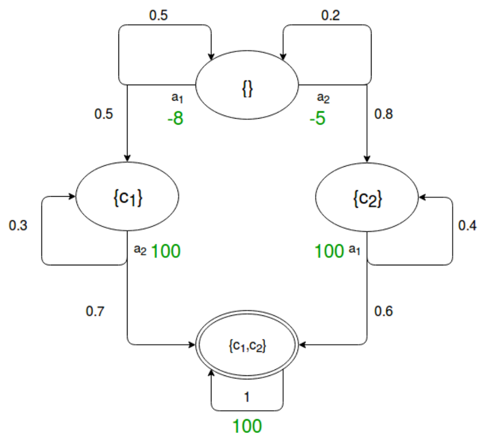

Figure 2.1: MDP for an academic problem with two courses. Black labels show the transition probability and the reward labels are indicated in green; a_i is the action to take course c_i.

The constructed graph is a sufficient model to illustrate the problem and to find a solution, but since most algorithms work better with a tree, we flatten the graph and break cycles if necessary.

Definition 2. *Given an MDP $M = \langle S, A, T, R, s_0, H \rangle$ a decision node is a 2-tuple $\langle s, h \rangle$ where:*

- *$s \in S$ is a state and*
- *$h \in [0, H]$ is the steps-to-go to reach a leaf.*

A chance node is a 3-Tuple $\langle s, a, h \rangle$ where

- *$s \in S$ is a state,*
- *$a \in A$ is an action,*
- *$h \in [1, H]$ is the steps-to-go to reach a leaf.*

The tree of M is a 7-tuple $T(M) = \langle D, C, E_c, E_d, n_0, L_c, L_d \rangle$ where:

- *D is a finite set of decision nodes,*
- *C is a finite set of chance nodes,*
- *$E_c \subseteq D \times C$ are the edges which connect a decision node with a chance node,*
- *$E_d \subseteq C \times D$ are the edges which connect a chance node with decision node,*
- *$n_0 \in D$ is the root node,*
- *$L_c \subseteq C \times A$ is the action of the following chance node and*
- *$L_d \in [0, 1]$ is the probability to get to the following decision node.*

$T(M) = \langle D, C, E_c, E_d, n_0, L_c, L_d \rangle$ is recursively defined by the following rules:

- the root node is $n_0 = \langle s_0, H \rangle$ and $n_0 \in D$
- For each decision node $d = \langle s, h \rangle \in$ D and each action a $\in$ A, there is a chance node c $\in$ C with c $= \langle s, a, h \rangle$, an edge $\langle d, c \rangle \in$ E and $L_c(\langle d, c \rangle) = a$
- For each $\langle s, a, h \rangle \in$ C and $s' \in S$ with T(s,a,s') $>$ 0 if h $>$ 0 then there is a $d = \langle s', h-1 \rangle \in D$

The tree also has some invariant properties. The root node $n_0 \in D$ is a decision node, also decision nodes and chance nodes alternate. Finally, all leaves of this tree are decision nodes.

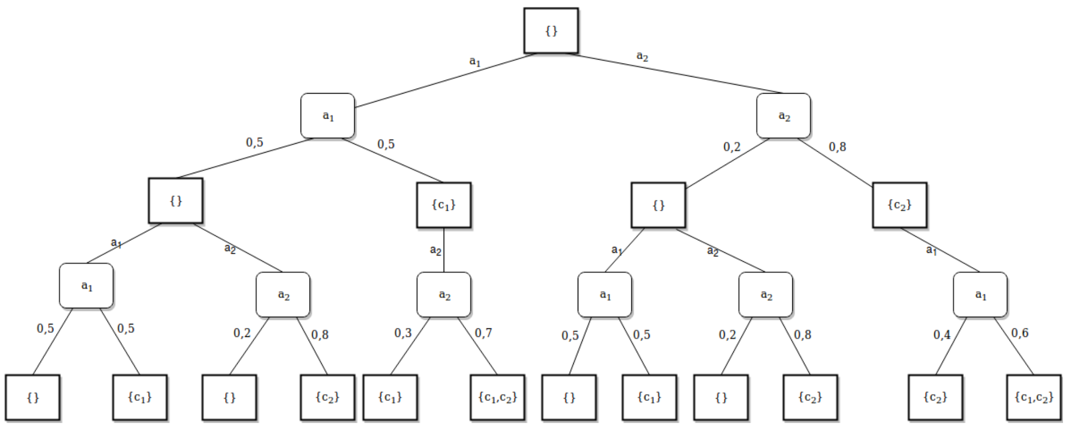

Figure 2.2: The corresponding tree to the MDP 2.1, *chance nodes* have rounded edges, a_i is the action to take course c_i. For easier understandability we remove the state notation in *chance nodes* and we omit redundant actions.

With these definitions we can now represent our planning problem, but it would be time consuming and resource intensive to generate and evaluate the whole tree for each problem. Therefore, we use a Monte-Carlo Tree Search algorithm like UCT, that constructs the tree in Figure 2.2 iteratively as long as provided resources (time and memory) allow. The algorithm evaluates actions which are promising and expand the tree in these directions. In order to do so, we need to update our definition of decision and chance nodes. A decision node is now a 4-tuple $\langle s, \hat{V}^t, N^t, h \rangle$ and a chance node a 5-tuple $\langle s, a, \hat{Q}^t, N^t, h \rangle$, where

- $\hat{V}^t \in \mathbb{R}$ is the state-value-estimate after t trials,
- $\hat{Q}^t \in \mathbb{R}$ is the action-value-estimate after t trials and
- $N^t \in \mathbb{N}$ is the number of visit of this node after t trials.

We further add a trial dependency to our tree, resulting in the tree $T^t = \langle D^t, C^t, E^t_c, E^t_d, n_0, L^t_c, L^t_d \rangle$, we also change the property of our tree so that leaves are chance nodes and not decision nodes anymore. With these changes we can use the UCT algorithm to construct our tree efficiently.

Upper Confidence Bounds to Tree (UCT)

The idea behind a UCT algorithm [5] is to iteratively add nodes to a tree. This step-wise expansion happens in multiple trials . Now the question is how to get in the time step t from our current tree T^t to the next one T^{t+1}. UCT achieves this transit in three phases: a *selection*, an *expansion* and a *backup* phase.
The selection phase consists of an action and an outcome selection. The action selection is used when we are at a decision node d, here we select the successor node

$$c = \underset{\langle d,c\rangle \in E_c}{argmax}\; Q^t(c) + V^t(d)\sqrt{\frac{N^t(d)}{N^t(c)}}, \tag{2.1}$$

where we have two main components in the formula. Either the action-value-estimate $\hat{Q}^t$ dominates the function then we exploit the promising candidate, or else we explore a less well examined candidate.
For a chance node we use the outcome selection. Either its probability or a probability biased by solved siblings determines in which state we will end.
When the selection phase reaches a decision node d which has not previous been visited and therefore has no children the expansion phase begins. We now expand all the children c of the node and then initialize the reward with a heuristic function. We now have $D^t \leftarrow D^t \cup d$ and $C^t \leftarrow C^t \cup c$. After the expansion phase, we have two options: we continue the selection phase or we initialize the backup phase.
In the backup phase the tree gets updated. Beginning in the previous expanded node we backtrack our path to the root node. On the way up we update the action-value-estimate $\hat{Q}^t$ for chance nodes and the state-value-estimate $\hat{V}^t$ for decision nodes. After we reach the root node we have T^{t+1}.
With this framework we can introduce ASAP (Abstraction of State-Action Pairs) in a UCT environment.

3 ASAP-UCT

The basic idea of ASAP is to use abstraction within the partially generated search tree of UCT. In a previous algorithm [8] only state pairs were compared for abstraction, here we go one step further and also compare state-action pairs. To save the abstraction we use equivalence classes. In order to do so we introduce two equivalence relations $\sim_d \subseteq D \times D$ and $\sim_c \subseteq C \times C$.

Definition 3. *For two decision nodes $d_1 = \langle s_1, \hat{V}_1^t, N_1^t, h_1 \rangle$ and $d_2 = \langle s_2, \hat{V}_2^t, N_2^t, h_2 \rangle$, we have $d_1 \sim_d d_2$ if and only if $h_1 = h_2$ and for all $\langle d_1, c_1 \rangle \in E_c^t$ there is a $\langle d_2, c_2 \rangle \in E_c^t$ with $c_1 \sim_c c_2$ and vice versa.*

Definition 4. *For two chance nodes $c_1 = \langle s_1, a_1, \hat{Q}_1^t, N_1^t, h_1 \rangle$ and $c_2 = \langle s_2, a_1, \hat{Q}_1^t, N_1^t, h_2 \rangle$ the relation $c_1 \sim_c c_2$ is true if and only if $h_1 = h_2$ and for all $\langle c_1, d_1 \rangle \in E_c^t$ there is a $\langle c_2, d_2 \rangle \in E_d^t$ with $d_1 \sim_d d_2$ where the transition probabilities are equal ($L_{d_1} = L_{d_2}$) and vice versa.*

We denote X as the set of equivalence classes under the relation $\sim_d$ and Y as the set of equivalence classes under relation $\sim_c$, we name $\bar{d} \in X$ the equivalence class for a decision node d and $\bar{c} \in Y$ respectively for a chance node c.
We assign each equivalence class a Q-value-mean M, this is for $\bar{d}$ the mean of all the state-value-estimates

$$M(\bar{d}) = \frac{1}{|\bar{d}|} \sum_{d \in \bar{d}} \hat{Q}(d) \tag{3.1}$$

and for $\bar{c}$ the mean of all the action-value-estimates

$$M(\bar{c}) = \frac{1}{|\bar{c}|} \sum_{c \in \bar{c}} \hat{V}(c) \tag{3.2}$$

In the implementation of Anand et al.[1] two trees are used, one abstract tree and the original flat tree. The flat tree is used in the expansion and in the simulation phase. We found this disadvantageous, since it means that we have an outdated tree. Hence, we use only one tree in which we use either the equivalence class of the nodes if one exists, or else the node itself. Thereby our tree can contain the information of the abstract and the original tree and we do not need to flatten the abstract tree. We do not overwrite the nodes with the equivalence class in the tree, because we cannot reuse the equivalence classes in general and need to recompute all equivalence classes after every interval.

With the previously declared UCT we expand our tree for a given time-interval τ. After τ we calculate the equivalence classes. We recursively go from the lowest steps-to-go to the root node.

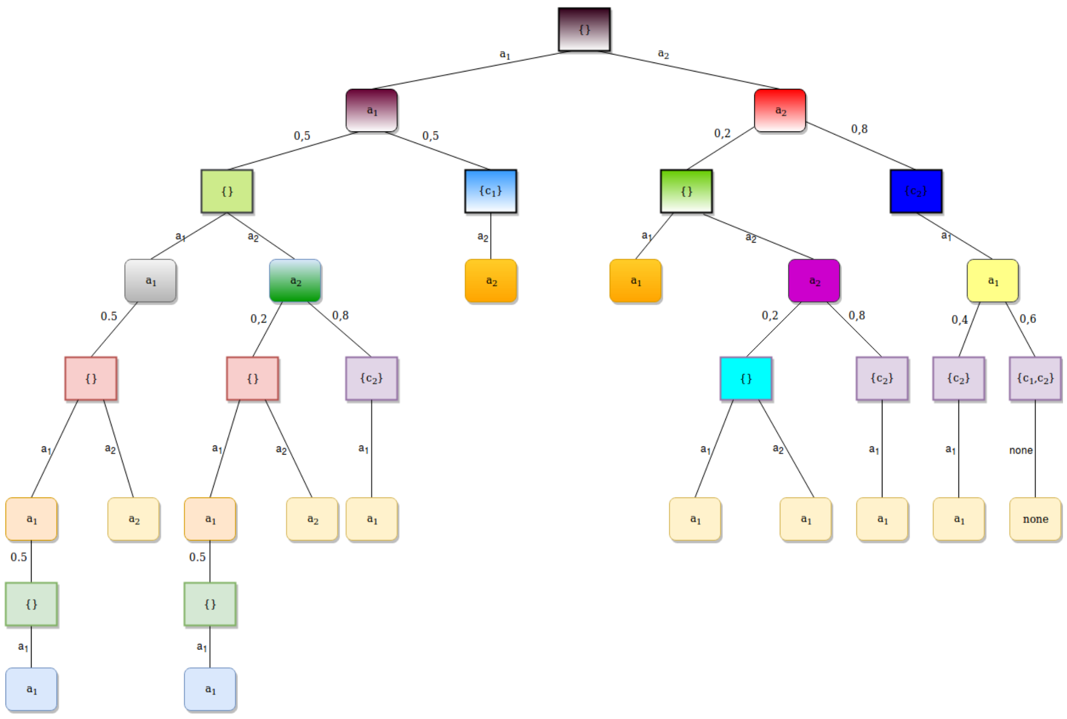

Figure 3.1: After calculating the equivalence classes, each color denotes a different equivalence class. Since we are in an incomplete tree, the sum of the probabilities does not need to be one.

In Algorithm 1 we can see how ASAP-UCT could be implemented. Because of the structure of the search tree we alternate between abstraction of decision nodes (Algorithm 2) and abstraction of chance nodes (Algorithm 3).

Algorithm 1 ASAP-UCT

```
  function GenerateEquivalenceClasses(SearchTree $T^t$ )
  steps= 0
    while steps $\neq$ H do
      for all n $\in D \cup C$ where $h = steps$ do
        if n $\in D$ then
          SetEQofDecisionNodes(n)
        else
          SetEQofChanceNodes(n)
        end if
      end for
      CreateNewEquivalenceClass($\bar{d_1}$)
      steps++
  end while
end function
```

Algorithm 2 Abstraction of decision nodes

```
  function SetEQofDecisionNodes( $d_1 = \langle s_1, \hat{V}_1^t, N_1^t, h_1 \rangle$)
    for all d = $\langle s_2, \hat{V}_2^t, N_2^t, h_2 \rangle \in$ X where $h_1 = h_2$ do
      if $d_1 \sim_d d$ then
        $\bar{d_1} = \bar{d}$ return
      end if
    end for
  CreateNewEquivalenceClass($\bar{d_1}$)
  end function
```

Algorithm 3 Abstraction of chance nodes

```
  function SetEQofChancenNodes( $c_1 = \langle s_1, a_1, \hat{Q}_1^t, N_1^t, h_1 \rangle$)
    for all c = $\langle s_2, a_2, \hat{V}_2^t, N_2^t, h_2 \rangle \in$ Y where $h_1 = h_2$ do
      if $c_1 \sim_c c$ then
        $\bar{c_1} = \bar{c}$ return
      end if
    end for
  CreateNewEquivalenceClass($\bar{c_1}$)
  end function
```

After reaching the root node, each node of the expanded tree has an equivalence class. This ends the phase of generating the equivalence-classes and the UCT algorithm proceeds again to the selection phase. If the node is in an equivalence class, the Q-value mean M is used in the action selection and more importantly in the backup phase.
The only change in the action selection is, that we use the Q-value-mean instead of the action-value-estimate and the Q-value-mean of the successor equivalence classes instead of the state-value-estimates for the children in equation 2.1. In the backup phase the Q-value-mean needs a recalculation as well.
When we are in the backup phase and without an equivalence class, we update only the value-estimate $\hat{V}^t$ or the action-value-estimate $\hat{Q}^t$ of the node. However, if the node is in an

equivalence class we need to recalculate the Q-value-mean with equation (3.1) or equation (3.2) respectively, because the reward of the node and thus of the whole equivalence class has changed.

4 Experiment Evaluation

Environment

The ASAP was programmed in C++ and uses as a framework the UCT-based MDP solver PROST [7] with the improvements introduced in the THTS [9] . Since the 2014 version makes use of the Partial Bellman Backups [9] as a backup function, it is not trivial to implement our abstractions with these kind of backups. Therefore we use the DP-UCT algorithm of the 2011 version of PROST which uses a Monte Carlo backup. To run the experiment we use the computer cluster of the University Basel, which contains Intel Xeon E5-2660 CPUs running at 2.2 GHz.

Setting

The setting for the solver includes a short time limit (0.5 seconds) to choose an action. This is interrupted by the generation of the abstraction every time interval τ. Based on the expenditure of time we excluded the time for the abstraction. After half a second the server receives the action, estimates the new state and sends the new state back to the solver. This exchange happens 40 times and then we receive a reward. For accuracy we do this exchange 30 times and then take the average.
Our testing domains were the benchmarks of the International Planning Competition (IPC) 2011[10] and 2014[11]. They include 12 domains in total, like for example the academic advising domain.

Configurations

We have run several configurations to see different and possible improvement. As τ we use an update count which represents how often we interrupt the solver for the abstraction. For additional configuration we also have the trial length as a parameter. This changes how many decision nodes are expanded in the expansion phase. Note that these decision nodes have not the same steps-to-go, they are successors from each other. The trial length does not change how many children in the expansion phase are generated. Additionally we deactivate the reasonable action of the PROST planner to see if there are any differences.

A reasonable action is an abstraction by itself, where actions are excluded if there is an obviously better solution[7]. In a last step we compare the differences between making the abstraction within PROST or with the UCT algorithm [5] in the PROST framework.

Parameter

For the representations of the parameter we use the format:

$$UCT_t^u - r(on/off),$$

where t is the trial length, u is how often we generate equivalence classes and $-r$ represents whether we have activated the reasonable action.
The difference between DP-UCT and UCT is how they expand the successor in the expansion phase and how they evaluate the state-value-estimate or the action-value-estimate respectively.
In the DP-UCT we expand H children in the expansion phase, in contrast to the UCT where we only expand one child. For the evaluation the UCT uses a random-walk and the DP-UCT utilizes a heuristic function.

Results

For reading purposes we split the result table shown in the Appendix A.1 and compare them among each other. The delimited pair is the reference data, where we use the UCT or DP-UCT without our abstraction.

	wildfire	triangle	recon	elevators	tamarisk	sysadmin	academic	game	traffic	crossing	skill	navigation	Total
DP-UCT$_1^0$ -r 1	0.9	0.94	0.98	0.89	0.91	0.9	0.5	0.95	0.98	0.81	0.91	0.31	0.83
UCT$_1^0$ -r 1	0.24	0.52	0.3	0.22	0.34	0.05	0.03	0.27	0.2	0.29	0.52	0.19	0.26
DP-UCT$_1^3$ -r 1	0.85	0.84	**0.96**	0.65	0.88	0.94	0.39	0.92	**0.97**	0.74	0.92	0.3	0.78
DP-UCT$_1^2$ -r 1	0.81	0.83	**0.98**	0.58	0.9	0.87	0.39	0.95	**0.98**	0.81	0.91	0.34	0.78
DP-UCT$_1^1$ -r 1	0.85	0.85	**0.98**	0.76	0.89	0.91	0.39	**0.96**	**0.97**	0.79	0.93	0.32	0.8
UCT$_1^4$ -r 1	0.28	0.59	0.3	0.27	0.3	0.08	0.02	0.31	0.1	0.29	0.55	0.23	0.28
UCT$_1^2$ -r 1	0.28	0.56	0.3	0.19	0.32	0.05	0.04	0.28	0.07	0.3	0.52	0.23	0.26
UCT$_1^1$ -r 1	0.25	0.57	0.32	0.41	0.32	0.05	0.04	0.27	0.12	0.28	0.52	0.18	0.28

Table 4.1: Results with activated reasonable action and the trial length of one.

We can see that the DP-UCT is significant better than the UCT algorithm in all the domains. In both cases the abstraction of ASAP has no real impact, therefore we increase the trial length, to try to increase the number of equivalence classes and to increase the impact of the abstraction.

	wildfire	triangle	recon	elevators	tamarisk	sysadmin	academic	game	traffic	crossing	skill	navigation	Total
DP-UCT^0_H -r 1	0.65	0.82	0.96	0.77	0.7	0.53	0.52	0.76	0.7	0.94	0.91	0.88	0.76
UCT^0_H -r 1	0.62	0.62	0.31	0.02	0.47	0.49	0.64	0.66	0.82	0.48	0.84	0.18	0.51
DP-UCT^3_H -r 1	0.77	0.86	0.94	0.72	0.68	0.57	0.49	0.73	0.74	0.92	0.92	0.88	0.77
DP-UCT^2_H -r 1	0.7	0.81	0.95	0.76	0.69	0.56	0.48	0.76	0.7	0.92	0.9	0.77	0.75
DP-UCT^1_H -r 1	0.66	0.83	**0.96**	0.78	0.69	0.57	0.5	0.78	0.71	0.91	0.91	0.8	0.76
UCT^4_H -r 1	0.64	0.63	0.32	0.02	0.49	0.47	0.65	0.6	0.81	0.43	0.85	0.17	0.51
UCT^2_H -r 1	0.62	0.62	0.31	0.03	0.45	0.49	0.64	0.63	0.8	0.46	0.87	0.15	0.51
UCT^1_H -r 1	0.62	0.61	0.32	0.02	0.45	0.46	0.64	0.64	0.81	0.43	0.87	0.18	0.5

Table 4.2: Results with activated reasonable action and the trial length of H.

For the UCT the domain scores increase with the higher trial length drastically. In the DP-UCT we do not see such an improvement. Once again the abstraction has no real impact on the outcome.

Hence we deactivate the reasonable actions to see if the additional abstractions have interfered with our abstraction and reset the trial length to one.

	wildfire	triangle	recon	elevators	tamarisk	sysadmin	academic	game	traffic	crossing	skill	navigation	Total
DP-UCT$^0_1$ -r 0	0.81	0.53	0.98	0.89	0.91	0.94	0.39	0.95	0.92	0.57	0.8	0.29	0.75
UCT$^0_1$ -r 0	0.13	0.28	0.0	0.17	0.36	0.05	0.07	0.3	0.53	0.2	0.64	0.1	0.23
DP-UCT$^3_1$ -r 0	0.81	0.52	0.94	0.49	0.91	0.91	0.39	0.92	0.9	0.46	0.76	0.18	0.68
$DP-UCT^2_1$ -r 0	0.81	0.47	**0.97**	0.72	0.91	0.88	0.39	0.93	0.91	0.6	0.85	0.18	0.72
DP-UCT$^1_1$ -r 0	0.81	0.51	**0.97**	0.8	0.88	0.9	0.49	**0.96**	0.91	0.59	0.74	0.26	0.73
UCT$^4_1$ -r 0	0.13	0.32	0.0	0.07	0.36	0.08	0.08	0.29	0.21	0.19	0.63	0.12	0.21
UCT$^2_1$ -r 0	0.08	0.31	0.0	0.09	0.34	0.07	0.08	0.25	0.2	0.19	0.64	0.12	0.2
UCT$^1_1$ -r 0	0.11	0.3	0.0	0.17	0.34	0.05	0.09	0.3	0.45	0.2	0.65	0.09	0.23

Table 4.3: Results with deactivated reasonable action and the trial length of one.

Without the reasonable actions the UCT scores worse, except for the traffic and the skill domain. Overall we can see that without the reasonable action the DP-UCT decreases its effectiveness. Our abstraction has here only minor effects and does not appear to be efficient.

Finally our last data without the reasonable action and with a increased trial length:

	wildfire	triangle	recon	elevators	tamarisk	sysadmin	academic	game	traffic	crossing	skill	navigation	Total
DP-UCT^0_H -r 0	0.62	0.6	0.91	0.75	0.77	0.53	0.58	0.69	0.71	0.86	0.88	0.72	0.72
UCT^0_H -r 0	0.59	0.28	0.07	0.02	0.63	0.51	0.38	0.6	0.89	0.39	0.84	0.12	0.44
DP-UCT^3_H -r 0	0.7	0.6	0.9	0.58	0.77	0.55	0.5	0.69	0.75	0.89	0.89	0.68	0.71
DP-UCT^2_H -r 0	0.64	0.6	0.91	0.61	0.77	0.51	0.47	0.68	0.73	0.89	0.91	0.66	0.7
DP-UCT^1_H -r 0	0.59	0.6	0.93	0.67	0.74	0.56	0.38	0.71	0.74	0.92	0.91	0.63	0.7
UCT^4_H -r 0	0.59	0.27	0.15	0.02	0.58	0.47	0.38	0.64	0.85	0.39	0.85	0.11	0.44
UCT^2_H -r 0	0.65	0.31	0.12	0.01	0.63	0.43	0.39	0.61	0.85	0.37	0.85	0.13	0.44
UCT^1_H -r 0	0.61	0.28	0.1	0.02	0.64	0.48	0.42	0.64	0.88	0.41	0.85	0.12	0.45

Table 4.4: Results with deactivated reasonable action and the trial length of H.

As seen before we have an increased output in UCT and a stable output in DP-UCT if we increase the trial length, thus the deactivation of the reasonable action has only a reduced impact. The abstraction appears to be as effective as before, but we have a small improvement in the recon domain.

We can see that with reasonable action and an increased trial length the UCT-algorithm is most efficient. The DP-UCT is in total constantly successful, but without the reasonable action the algorithm has a decreased effectiveness.

Overall the abstraction could not improve our results. A major problem in the theory of the ASAP is the abstraction of action-states. For this we look at the elevator domain, where we have multiple floors and an elevator. The elevator can be open, closed, moving up or moving down. On each floor there can be passengers which are waiting for the elevator.

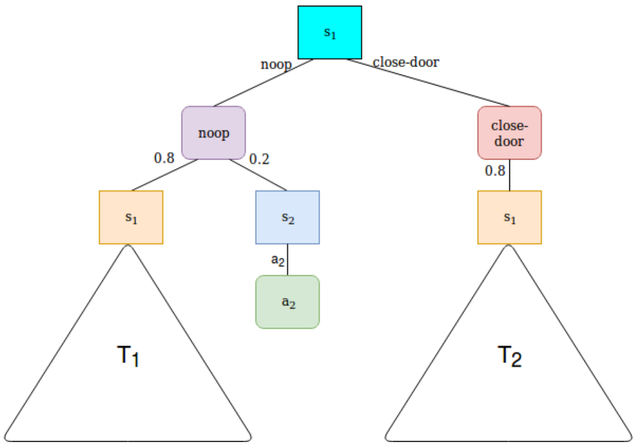

Figure 4.1: The elevator example; We create the special case where *close-door* is a redundant action, because in s_1 the door is already closed.

At some point we would like to have the chance node *noop* and the *close-door* in the same equivalence class which is only the case when the successors are also all in the same equivalence classes. If we have a child which has only a relatively small probability to be expanded like s_2, it is unlikely that the same node is expanded in the other state.
So in our example the subtree T_1 and T_2 are exactly the same, thus the decision nodes with the orange label are in the same equivalence class as they should because it is the exact same state. But through one expansion in T_1 which we do not perform in T_2 the subtrees become differently. If we then compute the new equivalence classes, we cannot put both decision nodes in the same equivalence class, since their successor equivalence classes are differently. This means we overlook two exact decision nodes. This inconsistency stretches through the tree, it is proportional to the size of the tree and will occur more often in the lower part of the tree and thus falsify the upper part.
For the first trials it is not a significant problem, since there are not many nodes to select from. Also if we have a complete tree the abstraction is correct and each same state is in a equivalence class.
An additional possible problem for the ASAP are the equivalence classes of leaves. Here we can easily suppress a promising candidate. The opposite can happen as well, meaning that we boost an unpromising one.

5 Conclusion

In this paper we implement the ASAP-UCT, an algorithm which combines abstraction and a UCT-framework. The ASAP algorithm uses state and state-action abstractions and combines them with the advantages of the UCT-algorithm, like that they can be stopped anytime and can give a promising action back.
The process of the algorithm consists of the three-phases of the UCT which are interrupted every interval to calculate the current equivalence classes. We constructs only one tree where we use either the equivalence class of the node or the node itself.

In Chapter 4 we evaluate our results, we tested the ASAP within the UCT-based MDP solver PROST and with the UCT-algorithm. We use different trial lengths and also observe the difference between the usage of reasonable action or without. Through our results we found a flaw in the abstraction approach.
In an incomplete tree which we create with the UCT, we have a high diversity in how subtrees can appear in the tree. For the abstraction this is a problem, because it leads to a near zero probability that two large subtrees are exactly the same. Therefore even if the root nodes of these subtrees represent the same state, they get different equivalence classes and thus they seem not the same for the algorithm.
The analysis of the various domains of the IPC domains leads to the result that the abstraction of state-action within a UCT is not a real improvement to a basic UCT.

Bibliography

[1] Anand, A., Grover, A., Mausam, and Singla, P. ASAP-UCT: Abstraction of State-Action Pairs in UCT. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1509–1515 (2015).

[2] Givan, R., Dean, T., and Greig, M. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1-2):163–223 (2003).

[3] Bellman, R. and Kalaba, R. On the role of dynamic programming in statistical communication theory. *IRE Trans. Information Theory*, 3(3):197–203 (1957).

[4] 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 Trans. Comput. Intellig. and AI in Games*, 4(1):1–43 (2012).

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

[6] Puterman, M. *Markov Decision Processes: Discrete Stochastic Programming*. Wiley (1994).

[7] 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).

[8] Jiang, N., Singh, S. P., and Lewis, R. L. Improving UCT planning via approximate homomorphisms. In *Proceedings of the International conference on Autonomous Agents and Multi-Agent Systems, AAMAS*, pages 1289–1296 (2014).

[9] 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*, pages 135–143 (2013).

[10] 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).

[11] 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).

A

Appendix

IPPC Scores: Total

	wildfire	triangle	recon	elevators	tamarisk	sysadmin	academic	game	traffic	crossing	skill	navigation	Total
$DP-UCT_1^0$ -r 0	0.81	0.53	0.98	0.89	0.91	0.94	0.39	0.95	0.92	0.57	0.8	0.29	0.75
$DP-UCT_1^0$ -r 1	0.9	0.94	0.98	0.89	0.91	0.9	0.5	0.95	0.98	0.81	0.91	0.31	0.83
$DP-UCT_H^0$ -r 0	0.62	0.6	0.91	0.75	0.77	0.53	0.58	0.69	0.71	0.86	0.88	0.72	0.72
$DP-UCT_H^0$ -r 1	0.65	0.82	0.96	0.77	0.7	0.53	0.52	0.76	0.7	0.94	0.91	0.88	0.76
UCT_1^0 -r 0	0.13	0.28	0.0	0.17	0.36	0.05	0.07	0.3	0.53	0.2	0.64	0.1	0.23
UCT_1^0 -r 1	0.24	0.52	0.3	0.22	0.34	0.05	0.03	0.27	0.2	0.29	0.52	0.19	0.26
UCT_H^0 -r 0	0.59	0.28	0.07	0.02	0.63	0.51	0.38	0.6	0.89	0.39	0.84	0.12	0.44
UCT_H^0 -r 1	0.62	0.62	0.31	0.02	0.47	0.49	0.64	0.66	0.82	0.48	0.84	0.18	0.51
min	0.0	0.2	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.04
$DP-UCT_1^3$ -r 0	0.81	0.52	0.94	0.49	0.91	0.91	0.39	0.92	0.9	0.46	0.76	0.18	0.68
$DP-UCT_1^3$ -r 1	0.85	0.84	**0.96**	0.65	0.88	0.94	0.39	0.92	**0.97**	0.74	0.92	0.3	0.78
$DP-UCT_1^2$ -r 0	0.81	0.47	**0.97**	0.72	0.91	0.88	0.39	0.93	0.91	0.6	0.85	0.18	0.72
$DP-UCT_1^2$ -r 1	0.81	0.83	**0.98**	0.58	0.9	0.87	0.39	0.95	**0.98**	0.81	0.91	0.34	0.78
$DP-UCT_1^1$ -r 0	0.81	0.51	**0.97**	0.8	0.88	0.9	0.49	**0.96**	0.91	0.59	0.74	0.26	0.73
$DP-UCT_1^1$ -r 1	0.85	0.85	**0.98**	0.76	0.89	0.91	0.39	**0.96**	**0.97**	0.79	0.93	0.32	0.8
$DP-UCT_H^3$ -r 1	0.77	0.86	0.94	0.72	0.68	0.57	0.49	0.73	0.74	0.92	0.92	0.88	0.77
$DP-UCT_H^3$ -r 0	0.7	0.6	0.9	0.58	0.77	0.55	0.5	0.69	0.75	0.89	0.89	0.68	0.71
$DP-UCT_H^2$ -r 0	0.64	0.6	0.91	0.61	0.77	0.51	0.47	0.68	0.73	0.89	0.91	0.66	0.7
$DP-UCT_H^2$ -r 1	0.7	0.81	0.95	0.76	0.69	0.56	0.48	0.76	0.7	0.92	0.9	0.77	0.75
$DP-UCT_H^1$ -r 0	0.59	0.6	0.93	0.67	0.74	0.56	0.38	0.71	0.74	0.92	0.91	0.63	0.7
$DP-UCT_H^1$ -r 1	0.66	0.83	**0.96**	0.78	0.69	0.57	0.5	0.78	0.71	0.91	0.91	0.8	0.76
UCT_1^4 -r 0	0.13	0.32	0.0	0.07	0.36	0.08	0.08	0.29	0.21	0.19	0.63	0.12	0.21
UCT_1^4 -r 1	0.28	0.59	0.3	0.27	0.3	0.08	0.02	0.31	0.1	0.29	0.55	0.23	0.28
UCT_1^2 -r 0	0.08	0.31	0.0	0.09	0.34	0.07	0.08	0.25	0.2	0.19	0.64	0.12	0.2
UCT_1^2 -r 1	0.28	0.56	0.3	0.19	0.32	0.05	0.04	0.28	0.07	0.3	0.52	0.23	0.26
UCT_1^1 -r 0	0.11	0.3	0.0	0.17	0.34	0.05	0.09	0.3	0.45	0.2	0.65	0.09	0.23
UCT_1^1 -r 1	0.25	0.57	0.32	0.41	0.32	0.05	0.04	0.27	0.12	0.28	0.52	0.18	0.28
UCT_H^4 -r 0	0.59	0.27	0.15	0.02	0.58	0.47	0.38	0.64	0.85	0.39	0.85	0.11	0.44
UCT_H^4 -r 1	0.64	0.63	0.32	0.02	0.49	0.47	0.65	0.6	0.81	0.43	0.85	0.17	0.51
UCT_H^2 -r 0	0.65	0.31	0.12	0.01	0.63	0.43	0.39	0.61	0.85	0.37	0.85	0.13	0.44
UCT_H^2 -r 1	0.62	0.62	0.31	0.03	0.45	0.49	0.64	0.63	0.8	0.46	0.87	0.15	0.51
UCT_H^1 -r 0	0.61	0.28	0.1	0.02	0.64	0.48	0.42	0.64	0.88	0.41	0.85	0.12	0.45
UCT_H^1 -r 1	0.62	0.61	0.32	0.02	0.45	0.46	0.64	0.64	0.81	0.43	0.87	0.18	0.5

Table A.1: All the results generated in our experiment. UCT_t^u -r 1, where t is the trial length, u is how often we generate the equivalence classes and -r represents if we have activated the reasonable action.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

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

Author — Autor
Tim Steindel

Matriculation number — Matrikelnummer
2014-050-389

Title of work — Titel der Arbeit
Enhancing Prost with Abstraction of State-Action Pairs

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, 02/11/2017

[illegible]

Signature — Unterschrift