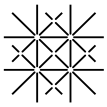

University of Basel

ASAP-UCT:TO DO

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

Calculations were performed at sciCORE (http://scicore.unibas.ch/) scientific computing core facility at University of Basel.

Abstract

Missing: ABstract schreiben

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. So it needs to learn which are good actions in its current state. Furthermore, our environment is dynamical 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 set of actions that leads to a desired goal given the possible actions and an initial state.

An example for a planning under uncertainty problem is the academic advising problem. Let us assume we have a student who wants to graduate. 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 simultaneously. Therefore, the student has several paths to reach his goal and the question now is, 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. If we calculate all the possible combination, we get $n! + 2$ states for $1 < \mathrm{n} \in \mathbb{N}$. Thus it is not advisable to use a brute force algorithm.

A traditional approach for an algorithm is to use abstraction. The theory behind abstraction is that we calculate equivalence classes, so that each class in an equivalence class has the same value and thus reduce the state space.
Another approach are Dynamic Programming algorithms [1], here we break our problem

in subproblems and save the intermediate result. Their main advantage is that they give an optimal solution, but because it needs to save the intermediate results, it needs a lot of memory.
An alternative approach is to use a Monte-Carlo-Tree Search(MCTS)[2] algorithm where we have only a part of the tree and try to find promising action and expand the tree in this direction. They have the advantage that they can be stopped anytime and can give a promising action back. A well-known algorithm for this is for example the UCT algorithm [3].

In this paper, we combine a UCT-algorithm with abstraction of states, but furthermore we also compute the equivalence classes of state-action pairs similar to the paper [4]. We firstly define the theoretical background needed like the Markov decision process, the search tree and UCT. We then implement ASAP-UCT (Abstraction-State-Action-Pair) as another approach for an algorithm. Afterwards we compare our result with a UCT-based MDP solver [5] and its later version, which used the idea introduced in [6]. 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). The useful property of a MDP is that a new state s' is only dependent of his 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*:* $S \times A \times S \to [0, 1]$ *is a probability distribution that gives the probability* $T(s, a, s')$ *that applying action* a *in* s *leads to* s*'*
- $R : S \times A \to \mathbb{R}$ *is the reward function*
- $s_0 \in S$ *is the initial state*
- $H \in \mathbb{N}$ *is the finite horizon*

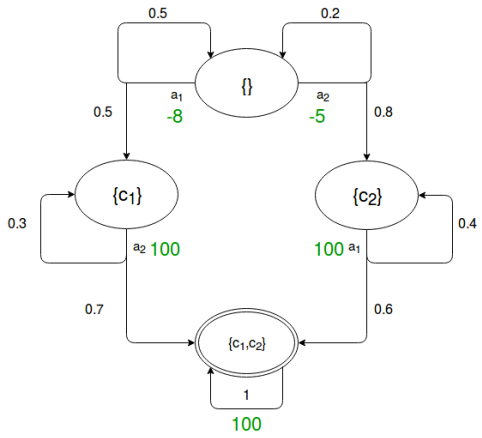

Figure 2.1: MDP for an academic problem with two courses. Black labels give transition probability, green are reward labels, the action a_i is to take course c_i

The constructed graph is a sufficient model to illustrate the problem and to find a solution, but because 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*
- $h \in [0, H]$ *is the steps-to-go to reach a leaf*

and 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 stepstogo 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*
- $L_d \in [0, 1]$ *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 has also some invariant properties.The root node n_0 is a decision node, also decision nodes and chance nodes alternate. Finally, all leaves of this tree are decision nodes.

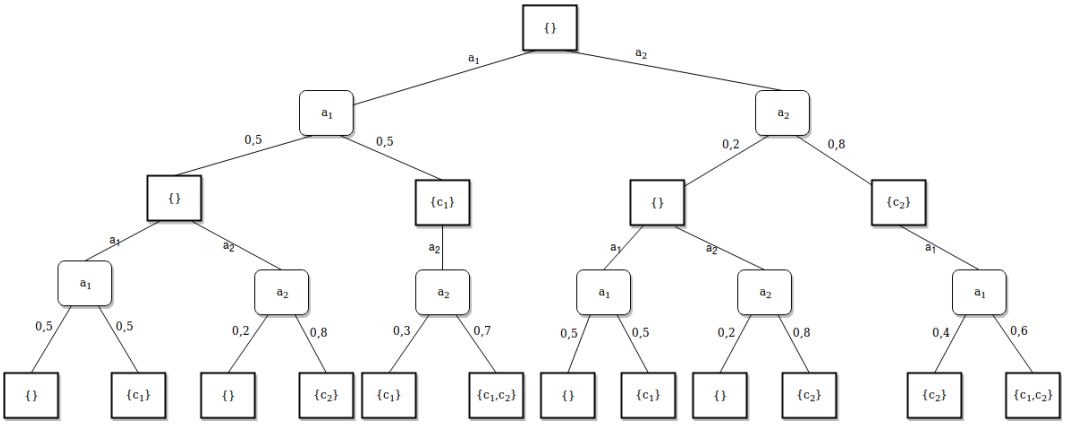

Figure 2.2: the corresponding tree to the MDP 2.1, *chance nodes* have rounded edges, a_i is the action to take course i, for easier understandability we remove the state notation in *chance nodes* and we omit redundant actions

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 build this tree iteratively as long as provided resources (time and memory) allow, to evaluate actions which are promising and expand the tree in these directions. For this, 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
- $N^t \in \mathbb{N}$ is the number of visit of this node after t trials

We further add a trial dependency to our tree, so the tree is now $T^t = \langle D^t, C^t, E_c^t, E_d^t, n_0, L_c^t, L_d^t\rangle$, we also change the property of our tree so that leaves are chance nodes and not decision nodes anymore. With these changes we can now use the UCT algorithm to construct our tree efficient.

Upper Confidence Bounds to Tree (UCT [3])

The idea behind a UCT algorithm is to iteratively add nodes to a tree. This step-wise expansion happens in trials t. Now the question is how to get 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

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

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. Through either its probability or a probability biased by solved siblings, we determine in which state we will end.
When the selection phase reaches a decision node, which was not previous been visited and therefore has no children, the expansion phase begins. We now expand all the children of the node and then initialize the reward with a heuristic function. After the expansion phase, we have two options. Either 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 now our T^{t+1}.
With this framework we can now 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 [7] 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. For this 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 also each equivalence class a Q-value-mean, this is for $\bar{d}$ the mean of all the state-value-estimates $\hat{Q}(\bar{d}) = \frac{1}{|\bar{d}|} \sum_{d \in \bar{d}} \hat{Q}(d)$
and for $\bar{c}$ the mean of all the action-value-estimates $\hat{V}(\bar{c}) = \frac{1}{|\bar{c}|} \sum_{c \in \bar{c}} \hat{V}(c)$

With the previous 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.

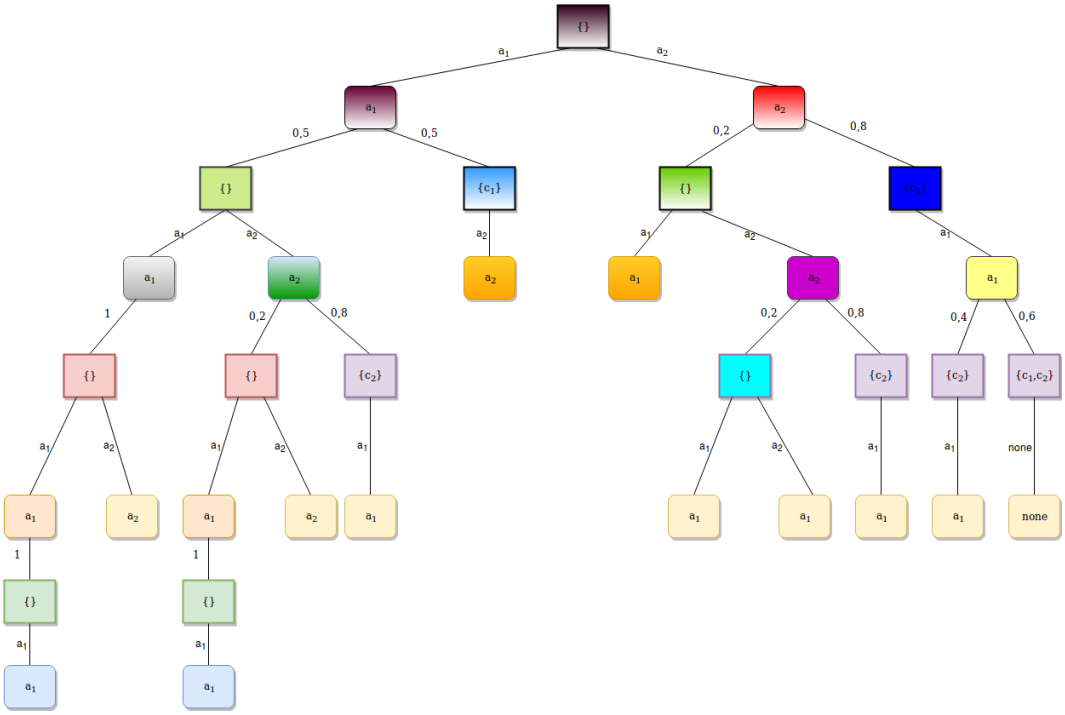

Figure 3.1: after calculating the equivalence classes, each color simulates a different equivalence class. Since we are in an incomplete tree, the sum of the probabilities do not need to be 1

In the algorithm 1 we can see how it could be implemented. Due to the structure of the search tree we alternate between abstraction of decision nodes (Algorithm 2) and abstraction of chance node (Algorithm 3).

Algorithm 1 ASAP-UCT

```
function GenerateEquivalenceClasses(SearchTree T^t )
steps= H
    while steps ≠ 0 do
        for all n ∈ D ∪ C where h = steps do
            if n ∈ D then SetEQofDecisionNodes
            else SetEQofChanceNodes
            end if
        end for
         CreateNewEquivalenceClass(d̄_1)
        i= i-1
    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 we reach the root node each node of the expanded tree has now an equivalence-class. Now the phase of generating the equivalence-classes is finished and the UCT algorithm goes again to the selection phase. The next time the UCT goes in the backup phase, the algorithm use, if the node is in an equivalence class, the Q-value mean of its equivalence class or the value-estimate respectively the action-value-estimate of the node, if the node is not yet in a equivalence class. If the node is in an equivalence class we need to recalculate the Q-value mean of the equivalence class, because the reward of the node and thus of the whole equivalence class has changed. The equivalence classes cannot be reused in general, therefore have to be recomputed every interval.

4

Experiment Evaluation

Environment

The ASAP was programmed in C++ and uses as a framework the UCT-based MDP solver PROST [5] with the improvements introduced in the THTS [6]. We use the computer cluster of the University Basel, which has Intel Xeon E5-2660 CPUs running at 2.2 GHz.

Configurations

We have made several different configurations to see difference and possible improvement. The setting for the solver includes a half a second time limit to choose an action. This is interrupted by the generation of the abstraction every τ time interval. 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 get a reward. For accuracy we do this exchange 30 times and then take the average. As τ we use the update frequency(uf), 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. Additionally we deactivate the reasonable action of the PROST planner to see if there are any differences. The reasonable action is an abstraction by itself, where actions are excluded if there is a obviously better solution. Finally we compare the difference between making the abstraction within PROST or within the UCT algorithm [3].
Our testing problem were the benchmarks of the International Planning Competition (IPC).

Results

The first results are with activated reasonable action and the trial length of one.

UCT -init [Single -h [RandomWalk]] -uf 0.11 -ndn H -t 0.5	0.74	0.53	0.65	0.02	0.45	0.47	0.32	0.61	0.82	0.42	0.84	0.17	0.5
UCT -init [Single -h [RandomWalk]] -uf 0.11 -t 0.5	0.31	0.5	0.02	0.26	0.29	0.08	0.31	0.32	0.11	0.29	0.54	0.22	0.27
UCT -init [Single -h [RandomWalk]] -uf 0.21 -ndn H -t 0.5	0.71	0.53	0.64	0.03	0.43	0.49	0.31	0.64	0.81	0.45	0.86	0.15	0.5
UCT -init [Single -h [RandomWalk]] -uf 0.21 -t 0.5	0.31	0.47	0.04	0.17	0.3	0.05	0.31	0.29	0.08	0.29	0.51	0.22	0.25
UCT -init [Single -h [RandomWalk]] -uf 0.41 -ndn H -t 0.5	0.71	0.51	0.64	0.02	0.42	0.46	0.33	0.65	0.82	0.42	0.86	0.18	0.5
UCT -init [Single -h [RandomWalk]] -uf 0.41 -t 0.5	0.28	0.47	0.04	0.37	0.3	0.05	0.33	0.28	0.13	0.27	0.51	0.18	0.27
UCT -init [Single -h [RandomWalk]] -uf 2.0 -ndn H -t 0.5	0.71	0.53	0.65	0.01	0.44	0.49	0.32	0.66	0.82	0.47	0.83	0.18	0.51
UCT -init [Single -h [RandomWalk]] -uf 2.0 -t 0.5	0.26	0.43	0.03	0.21	0.32	0.05	0.3	0.28	0.21	0.28	0.51	0.19	0.26

and the same for the UCT-based MDP solver PROST as a framework:

IPPC2011 -uf 0.11 -ndn H -t 0.5	0.86	0.5	0.6	0.92	0.76	0.71	0.95	0.88	0.9	0.81	0.93	0.47	0.77
IPPC2011 -uf 0.11 -t 0.5	0.92	0.49	0.6	0.86	0.83	0.71	**0.97**	0.87	0.88	0.79	0.94	0.53	0.78
IPPC2011 -uf 0.21 -ndn H -t 0.5	0.85	0.49	0.6	0.86	0.81	0.7	0.92	0.86	0.89	0.83	0.94	0.52	0.77
IPPC2011 -uf 0.21 -t 0.5	0.86	0.59	0.6	0.87	0.79	0.7	0.91	0.88	0.89	0.8	0.93	0.51	0.78
IPPC2011 -uf 0.41 -ndn H -t 0.5	0.86	0.49	0.6	0.92	0.81	0.77	**0.98**	0.87	0.91	0.83	0.92	0.49	0.79
IPPC2011 -uf 0.41 -t 0.5	0.87	0.51	0.6	0.88	0.8	0.71	**0.98**	0.87	0.9	0.82	**0.97**	0.54	0.79
IPPC2011 -uf 2.0 -ndn H -t 0.5	0.89	0.51	0.6	0.92	0.8	0.71	**0.99**	0.87	0.89	0.81	**0.96**	0.49	0.79

We can see that the PROST is in nearly all the domains better than the UCT algorithm. In both cases the abstraction of ASAP have no real impact, so we now increase the trial length, to try to increase the numbers of equivalence classes and to increase the impact of the abstraction.

UCT -init [Single -h [RandomWalk]] -uf 0.11 -ndn H -t 0.5	0.74	0.53	0.65	0.02	0.45	0.47	0.32	0.61	0.82	0.42	0.84	0.17	0.5
UCT -init [Single -h [RandomWalk]] -uf 0.11 -t 0.5	0.31	0.5	0.02	0.26	0.29	0.08	0.31	0.32	0.11	0.29	0.54	0.22	0.27
UCT -init [Single -h [RandomWalk]] -uf 0.21 -ndn H -t 0.5	0.71	0.53	0.64	0.03	0.43	0.49	0.31	0.64	0.81	0.45	0.86	0.15	0.5
UCT -init [Single -h [RandomWalk]] -uf 0.21 -t 0.5	0.31	0.47	0.04	0.17	0.3	0.05	0.31	0.29	0.08	0.29	0.51	0.22	0.25
UCT -init [Single -h [RandomWalk]] -uf 0.41 -ndn H -t 0.5	0.71	0.51	0.64	0.02	0.42	0.46	0.33	0.65	0.82	0.42	0.86	0.18	0.5
UCT -init [Single -h [RandomWalk]] -uf 0.41 -t 0.5	0.28	0.47	0.04	0.37	0.3	0.05	0.33	0.28	0.13	0.27	0.51	0.18	0.27
UCT -init [Single -h [RandomWalk]] -uf 2.0 -ndn H -t 0.5	0.71	0.53	0.65	0.01	0.44	0.49	0.32	0.66	0.82	0.47	0.83	0.18	0.51
UCT -init [Single -h [RandomWalk]] -uf 2.0 -t 0.5	0.26	0.43	0.03	0.21	0.32	0.05	0.3	0.28	0.21	0.28	0.51	0.19	0.26

and the same for the IPPC11 environment:

IPPC2011 -uf 0.11 -ndn H -t 0.5	0.86	0.5	0.6	0.92	0.76	0.71	0.95	0.88	0.9	0.81	0.93	0.47	0.77
IPPC2011 -uf 0.11 -t 0.5	0.92	0.49	0.6	0.86	0.83	0.71	**0.97**	0.87	0.88	0.79	0.94	0.53	0.78
IPPC2011 -uf 0.21 -ndn H -t 0.5	0.85	0.49	0.6	0.86	0.81	0.7	0.92	0.86	0.89	0.83	0.94	0.52	0.77
IPPC2011 -uf 0.21 -t 0.5	0.86	0.59	0.6	0.87	0.79	0.7	0.91	0.88	0.89	0.8	0.93	0.51	0.78
IPPC2011 -uf 0.41 -ndn H -t 0.5	0.86	0.49	0.6	0.92	0.81	0.77	**0.98**	0.87	0.91	0.83	0.92	0.49	0.79
IPPC2011 -uf 0.41 -t 0.5	0.87	0.51	0.6	0.88	0.8	0.71	**0.98**	0.87	0.9	0.82	**0.97**	0.54	0.79
IPPC2011 -uf 2.0 -ndn H -t 0.5	0.89	0.51	0.6	0.92	0.8	0.71	**0.99**	0.87	0.89	0.81	**0.96**	0.49	0.79

For the UCT we increase drastically the domain scores with the now higher trial length, in the PROST planner we do not see such improvement. Once again the abstraction has no real impact on the outcome.

Hence we deactivate the reasonable actions, to see if the additional abstraction have changed the results.

UCT -init [Single -h [RandomWalk]] -uf 0.11 -ndn H -t 0.5	0.65	0.17	0.45	0.02	0.53	0.47	0.15	0.65	0.87	0.38	0.86	0.11	0.44
UCT -init [Single -h [RandomWalk]] -uf 0.11 -t 0.5	0.14	0.22	0.09	0.07	0.33	0.08	0.0	0.3	0.21	0.19	0.64	0.12	0.2
UCT -init [Single -h [RandomWalk]] -uf 0.21 -ndn H -t 0.5	0.72	0.21	0.48	0.01	0.57	0.44	0.12	0.62	0.87	0.36	0.86	0.13	0.45
UCT -init [Single -h [RandomWalk]] -uf 0.21 -t 0.5	0.08	0.21	0.08	0.08	0.31	0.07	0.0	0.25	0.2	0.19	0.65	0.13	0.19
UCT -init [Single -h [RandomWalk]] -uf 0.41 -ndn H -t 0.5	0.68	0.17	0.64	0.02	0.58	0.48	0.1	0.65	0.89	0.4	0.86	0.12	0.47
UCT -init [Single -h [RandomWalk]] -uf 0.41 -t 0.5	0.11	0.2	0.09	0.16	0.31	0.05	0.0	0.31	0.46	0.2	0.66	0.09	0.22
UCT -init [Single -h [RandomWalk]] -uf 2.0 -ndn H -t 0.5	0.66	0.18	0.47	0.02	0.57	0.51	0.07	0.61	0.9	0.37	0.85	0.12	0.44
UCT -init [Single -h [RandomWalk]] -uf 2.0 -t 0.5	0.13	0.18	0.07	0.16	0.33	0.05	0.0	0.3	0.54	0.2	0.65	0.1	0.23

and again the same for the IPPC11 environment:

IPPC2011 -uf 0.11 -ndn H -t 0.5	0.86	0.3	0.59	0.79	0.9	0.7	**0.99**	0.86	0.81	0.76	0.92	0.46	0.75
IPPC2011 -uf 0.11 -t 0.5	0.84	0.31	0.6	0.8	0.91	0.71	**0.99**	0.86	0.85	0.77	0.93	0.49	0.75
IPPC2011 -uf 0.21 -ndn H -t 0.5	0.82	0.31	0.59	0.77	0.88	0.75	**0.99**	0.83	0.83	0.76	0.93	0.46	0.74
IPPC2011 -uf 0.21 -t 0.5	0.8	0.31	0.6	0.79	0.9	0.74	**0.98**	0.84	0.83	0.77	0.9	0.47	0.74
IPPC2011 -uf 0.41 -ndn H -t 0.5	0.8	0.31	0.6	0.8	0.94	0.71	**0.99**	0.83	0.85	0.8	0.91	0.49	0.75
IPPC2011 -uf 0.41 -t 0.5	0.87	0.31	0.6	0.82	0.89	0.74	**0.99**	0.84	0.84	0.74	0.93	0.51	0.76
IPPC2011 -uf 2.0 -ndn H -t 0.5	0.84	0.31	0.6	0.84	0.9	0.76	**0.99**	0.85	0.84	0.79	0.94	0.42	0.76
IPPC2011 -uf 2.0 -t 0.5	0.87	0.31	0.6	0.85	0.89	0.72	**0.99**	0.82	0.85	0.75	0.94	0.46	0.75

Without the reasonable action the UCT scores in the domains wildfire and triangle significant better, but in the traffic domain it falls of in quality. Our abstraction has here only minor effect and does not appear to be efficient.

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

UCT -init [Single -h [RandomWalk]] -uf 0.11 -ndn H -t 0.5	0.65	0.17	0.45	0.02	0.53	0.47	0.15	0.65	0.87	0.38	0.86	0.11	0.44
UCT -init [Single -h [RandomWalk]] -uf 0.11 -t 0.5	0.14	0.22	0.09	0.07	0.33	0.08	0.0	0.3	0.21	0.19	0.64	0.12	0.2
UCT -init [Single -h [RandomWalk]] -uf 0.21 -ndn H -t 0.5	0.72	0.21	0.48	0.01	0.57	0.44	0.12	0.62	0.87	0.36	0.86	0.13	0.45
UCT -init [Single -h [RandomWalk]] -uf 0.21 -t 0.5	0.08	0.21	0.08	0.08	0.31	0.07	0.0	0.25	0.2	0.19	0.65	0.13	0.19
UCT -init [Single -h [RandomWalk]] -uf 0.41 -ndn H -t 0.5	0.68	0.17	0.64	0.02	0.58	0.48	0.1	0.65	0.89	0.4	0.86	0.12	0.47
UCT -init [Single -h [RandomWalk]] -uf 0.41 -t 0.5	0.11	0.2	0.09	0.16	0.31	0.05	0.0	0.31	0.46	0.2	0.66	0.09	0.22
UCT -init [Single -h [RandomWalk]] -uf 2.0 -ndn H -t 0.5	0.66	0.18	0.47	0.02	0.57	0.51	0.07	0.61	0.9	0.37	0.85	0.12	0.44
UCT -init [Single -h [RandomWalk]] -uf 2.0 -t 0.5	0.13	0.18	0.07	0.16	0.33	0.05	0.0	0.3	0.54	0.2	0.65	0.1	0.23

and again the same for the IPPC11 environment:

IPPC2011 -uf 0.11 -ndn H -t 0.5	0.86	0.3	0.59	0.79	0.9	0.7	**0.99**	0.86	0.81	0.76	0.92	0.46	0.75
IPPC2011 -uf 0.11 -t 0.5	0.84	0.31	0.6	0.8	0.91	0.71	**0.99**	0.86	0.85	0.77	0.93	0.49	0.75
IPPC2011 -uf 0.21 -ndn H -t 0.5	0.82	0.31	0.59	0.77	0.88	0.75	**0.99**	0.83	0.83	0.76	0.93	0.46	0.74
IPPC2011 -uf 0.21 -t 0.5	0.8	0.31	0.6	0.79	0.9	0.74	**0.98**	0.84	0.83	0.77	0.9	0.47	0.74
IPPC2011 -uf 0.41 -ndn H -t 0.5	0.8	0.31	0.6	0.8	0.94	0.71	**0.99**	0.83	0.85	0.8	0.91	0.49	0.75
IPPC2011 -uf 0.41 -t 0.5	0.87	0.31	0.6	0.82	0.89	0.74	**0.99**	0.84	0.84	0.74	0.93	0.51	0.76
IPPC2011 -uf 2.0 -ndn H -t 0.5	0.84	0.31	0.6	0.84	0.9	0.76	**0.99**	0.85	0.84	0.79	0.94	0.42	0.76
IPPC2011 -uf 2.0 -t 0.5	0.87	0.31	0.6	0.85	0.89	0.72	**0.99**	0.82	0.85	0.75	0.94	0.46	0.75

As seen before we have the increase output in UCT and the stable output of PROST. The abstraction appears to be as effective as before, but we have a spike in the academic problem.

Overall the abstraction could not create a real improvement to our data, despite the disregard of the time efficient. A possible problem for the ASAP are the equivalence classes of leaves. Here we can easily dilute a promising candidate. Overall also the opposite can happen, so that we boost a unpromising one.

5
Conclusion

In this paper we implement the ASAP-UCT, an algorithm which combines abstraction and a UCT-framework. The idea of the ASAP is to use state and state-action abstraction to get the advantages of the UCT, like that they can be stopped anytime and the advantage of the abstraction to reduce the complexity.
In Chapter 4 we evaluate our results, we tested the ASAP within the UCT-based MDP solver PROST and with the UCT-algorithm. We used different trial length and also observe the difference between the usage of reasonable action or without. Hence the results the abstraction of state and state-action within a UCT is not a real improvement to a basic UCT.

Bibliography

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

[2] 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).

[3] Kocsis, L. and Szepesvári, C. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML*, pages 282–293 (2006).

[4] 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).

[5] 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).

[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).

[7] Jiang, N., Singh, S. P., and Lewis, R. L. Improving UCT planning via approximate homomorphisms. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pages 1289–1296 (2014).

A

Appendix

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
ASAP-UCT:TO DO

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

Signature — Unterschrift