Institut für Parallele und Verteilte Systeme

Abteilung Maschinelles Lernen und Robotik

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Bachelorarbeit Nr. 231

Monte Carlo Tree Search for Concurrent Actions

Li Yang Wu

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Marc Toussaint
Betreuer:	Ph.D. Vien Ngo
begonnen am:	06.05.2015
beendet am:	05.11.2015
CR-Klassifikation:	F.2.0, I.2.8

CONTENTS

ABSTRACT

Many decision problems within robotics concern the control of potentially concurrent actions in continuous time, each with stochastic durations. There exist various formalizations of such decision processes. This thesis aims to investigate in a reduction that is suitable for Monte-Carlo Tree Search. In particular, the approach should be compared to existing reductions as Semi-Markov Decision Processes w.r.t. the generality of the formalisms as well as the notions of optimality guaranteed. Do the optimality proofs of Upper Confidence Bounds applied to Trees directly transfer to the concurrent action case? The handling of general stochasticity, also of the action durations, should be investigated in detail. Further, it is of interest to theoretically investigate and compare existing implementations w.r.t. the consistency and performance.

1. INTRODUCTION

When it comes to concurrent action planning with stochastic components, Markov decision processes (MDP) for discrete time steps and semi Markov decision processes (sMDP) for continues time has always been a good model do describe these problems. Our Formalism is an instance of sMDP. The initial motivation for the creation of this formalism was that we wanted to have a simple and well performing implementation for first order logic environments. It turned out to perform well, so we were very interested in the theoretical aspect of our formalism. The best way to do so is to compare it to an existing similar formalism, which is the Concurrent Action Model. We start to describe this model in short. Then we define our formalism in two parts, first the syntax then the semantics. Shortly after we show that it converges to the optimum within its own policy space by applying the bellman optimality equation to it and show the convergence of the value iteration. This result is important for further investigation. And now, both of the most important topics of this thesis are discussed:

- proof of generality and optimality of our formalism
- runtime comparison between the concurrent action model and our formalism

The basic idea in showing generality and optimality is to do a reduction in both directions, defining a simulation and show that they converge to the same optimum. For the runtime comparison we use the Upper Confidence Tree Algorithm, which is an instance of Monte Carlo Tree Search algorithms. We give a pseudocode and analyse it first independent from the choice of model. Then we apply these two models to the runtime entities and show the difference. At the end, we discuss the results.

2. Notations

In this thesis, standard mathematic notations are used. We use abbreviation only seldom; those we use are well known. All new terms and concepts are defined before used. But since some mathematic expressions are always used differently, we will give a list of terms that might be confusing:

- $\mathbb{R}$... the set of real numbers
- $\mathbb{N}$... the set of natural numbers
- $\mathfrak{P}(M)$... power set of set M
- $\mathcal{O}(g(x))$... big O notation according to Landau, if g is a function and x its parameters
- expected utility $EU(a|s) = \sum_{t(s,a)} \sum_{\tau(a)} P(t(s,a),\tau(a)|s,a)R(s,d,t(s,a),\tau(a))$
  - a... action
  - $t(s,a)$... next state
  - $\tau(a)$... duration of action a

  We use this term more in prose than in equations to explain something. For the parameter s we assume the current state. The action is given in context.
- $\square$... completion of a proof which is part of another proof
- $\blacksquare$... completion of a proof
- $\lightning$... contradiction

3. Formalisms

3.1. Concurrent Action Model (CAM)

Rohanimanesh and Mahadevan [2] presented a formalism for concurrent actions with stochastic duration described as a sMDP, which is formally the 4-tuple (S, A, T, R). Its components are:

- S – A set of states
- A – A set of primary actions
- T – A transition probability distribution $S \times \mathfrak{P}(A) \times S \times \mathbb{N} \to [0,1]$
  - $\mathfrak{P}(A)$ – The power-set of A. Each element of it is also called *multi-action*
- R – The reward function mapping $S \to \mathbb{R}$

Since different action may have different duration, the termination of a multi-action must be defined. They presented three termination schemes, all of them coverging to the opti-

mum. One of them, which is T_{any}, matches our formalism best, thus we will use this termination scheme to compare with. In this scheme a Markov step is terminated when the action within the multi-action with the least duration is terminated. All remaining actions are interrupted.

3.2. Sequential Initiation Model (SIM)

In SIM we assume a first order logic world. Decisions are made according to a knowledge base. The knowledge base is modified by actions. Multiple Actions can be performed concurrently and have stochastic duration and state transition. Basically the formalism allows the agent to make two different kinds of decisions. The agent can decide to *Wait* or to *Initiate(a)*. Where *a* is a single action. The *Wait* decision ensures that no other action is initiated, until the next potentially change in the knowledge base. First i want to present the syntactic elements of SIM. Afterwards, the semantics are defined.

3.2.1. Syntax

- *RelationalState(i)* – A set of FOL konjunction literals
- *go(a, X)* – A real valued pedicate
- *Activity* – A action symbol with two Operators: Initiation, Termination
- *ActivitySpace* – A finite set of Activities
- *InitiationOperator* – A FOL sentence named $a_I(X)$
- *TerminationOperator* – The FOL sentence named $a_T(X)$
- *Initiate* – A Markov decision
- *Wait* – A Markov decision
- *DecisionSet* – A finite set of Markov decisions D
- *Reward* – A mapping $R : S \times D \times S \times \mathbb{R} \to \mathbb{R}$
- *ProbabilityDistribution* – $P : S \times D \times S \to \mathbb{R}$

3.2.2. Semantic

- *RelationalState(i)* – The *RelationalState(i)* are the facts from the relational knowledge base after the i-th modification; modifications are caused by the *InitiationOparator* and *TerminationOperator*; the initial relational state is s_0.
- *go(a, X)* – The go predicate indicates whether the activity *a* with arguments X is currently *active* or *not active*. Its value is the total amount of time needed to perform the activity. The random distribution of this value is problem specific.

- *Activity* – An Activity represents an action. If the go predicate for the activity is greater zero the action is *active*, otherwise not; we say that an *Acitivity a* is *hot* iff the precondition in the *InitiationOperator* is true for a.
- *ActivitySpace* – The agent can only initiate activities contained in this set.
- *InitiationOperator* – $a_I(X) : precond(X) \rightarrow go(a,X) = d_a, other\_effects$, where a is a activity and X are the arguments. The *other_effects* are problem specific and can change the world according so some domain rules.
- *TerminationOperator* – $a_T(X) : \ \rightarrow \neg go(a,X), other\_effects$, where a is a activity and X are the arguments.
- *Initiate(a)* – If the precondition of Activity a is fulfilled, trigger the *InitiationOperator* of a.
- *Wait* – The agent does not *Initiate* anything until the active action with the lowest *go* value is terminated. We often call this time $\tau(Wait) = \min_a\{go(a,X) > 0\}$.
- *DecisionSet* – $D = \{Wait\} \cup \{Initiate(a) | a \in ActivitySpace \wedge a \text{ is hot}\}$.
- *Reward* – This is the reward function $R(s,d,s',\tau(d)) = r$.
  - $\tau(d)$ – This is called *Real Time*. This is a effect caused by each decision. $\tau(Wait) \geq 0$ and $\tau(d) = 0$ for all other decisions.
- *ProbabilityDistribution* – For *Wait*, the state transition can be modelled by any kind of distribution, however the Poisson distribution is typically used due to simplicity. For any other decisions, particularly all initiations, the state transition is deterministic. After an initiation of an activity a, there is a variable in the *RelationalState* which marks a as active.

In some cases we want to express a whole sequence of initiation for a sequence of activities $a_1,..,a_n$. We do that by denoting $Initiate(a_1,..,a_n)$. We call it an *initiation batch*.

This list of model components looks very large. While each of these components are important for the understanding of how the formalism works and how to implement it, of course not all of them are important for theoretical investigation. Later, when we map a general CAM problem to a SIM problem, we pass on defining every components shown above, but concentrate of the important ones.

POLICY The policy is defined as the mapping $\pi : S \rightarrow D$. So, we assume a simple reactive agent.

THE MEANING OF WAIT At the beginning we said that SIM is an instance of sMDP. This is important because the Upper Confidence Tree algorithm works only for problems described as MDP or sMDP. One thing might look very different from other formalisms. Initiations do not progress real time, thus the agent is doing nothing. But in SIM we can imagine a *super*

Markov step (sMs). An sMs is nothing else than a initiation batch followed by a *Wait* decision. So, we can say that the end of one sMs and the start of the next is when the *Wait* decision has finished. It assures that no other decisions can be made before it finished. After that, new activities can be initiated, until the next *Wait*. This concept of an initiation batch followed by a *Wait* shows that SIM is highly compatible with the understanding of sMDPs and therefore suitable for Monte Carlo Tree Search. In fact, we make heavy use of this concept to show the equivalence of CAM and SIM

3.2.3. Bellman Optimality Equation

Applying the Bellman Optimality Equation [3] on SIM, we have:

$$V^*(s) = \max_{d \in D} \sum_{s'} \sum_{\tau(d)} P(s', \tau(d)|s, d) \left[R(s, d, s', \tau(d)) + \gamma^{\tau(d)} V^*(s') \right]$$

There are two special things in this equation. First, it also sums over $\tau(d)$ which is the duration of an activity. This is because the duration is stochastic.

Second, we can see that the value iteration for this equation converges only if $\gamma^{\tau(d)} < 1$. We choose $\gamma < 1$. Since only the *Wait* decision progresses real time, the expression becomes 1 for any other decisions.

Intuitively, the value iteration converges if we look at following consideration: Assume that it does not converge. Then the sequence of the full value iteration would have only a finite amount of *Wait* decisions and an infinite amount of other decisions. So, there would exist an instant of time from which no *Wait* decisions are made any more. But it is not possible to make infinite consecutive Initiations because the *ActivitySpace* is finite. By no later than all activities has been initiates since the last *Wait* decisions, only the *Wait* decisions itself can be made.

It follows directly that there must be an infinite amount of *Wait* decisions. As a consequence, the number of times in with $\gamma^{\tau(d)} < 1$ is infinite, thus the mean value of $\gamma^{\tau(d)}$ is smaller than 1. So, the equation converges.

We use the Q-Function to show the convergence of our Bellman Equation:

$$Q(s,d) = \sum_{s'} \sum_{\tau(d)} P(s', \tau(d)|s, d)[R(s, d, s', \tau(d)) + \gamma^{\tau(d)} \max_{d' \in D} Q(s', d')]$$

Define: $\Delta(s,d) := Q*(s,d) - Q(s,d)$

$$\Rightarrow \forall s, d : |\Delta(s,d)| < \epsilon$$

$$\Rightarrow Q(s,d) \leq \sum_{s'} \sum_{\tau(d)} P(s', \tau(d)|s, d)[R(s, d, s', \tau(d)) + \gamma^{\tau(d)} \max_{d' \in D} Q^*(s', d') + \epsilon]$$

$$\begin{aligned}
&= R(s) + \sum_{s'} \sum_{\tau(d)} P(s', \tau(d)|s, d) \gamma^{\tau(d)} \max_{d' \in D} Q^*(s', d') + \epsilon] \\
&= R(s) + \gamma^{\tau} \epsilon + \sum_{s'} \sum_{\tau(d)} P(s', \tau(d)|s, d) \gamma^{\tau(d)} \max_{d' \in D} Q^*(s', d')] \\
&= \gamma^{\tau} \epsilon + Q^*(s, d)
\end{aligned}$$

$\Rightarrow Q(s,d) \geq Q^*(s,d) - \gamma^\tau \epsilon \Rightarrow Q(s,d) \in [Q^*(s,d-\gamma^\tau\epsilon, Q^*(s,d)+\gamma^\tau\epsilon)]$ for $\gamma < 1 \Rightarrow Q$ converges.

According to the discussion above, the mean value of $\gamma^{\tau(d)}$ is smaller than 1, therefore it converges.

■

In the next chapters sometimes we treat CAM and SIM as actors who does something. What is meant is an implementation of a solution using this model. And they do something.

4. Equivalence of CAM and SIM

4.1. SIM as general and optimal as CAM

We first want to show that SIM can cover the same policy space and within that it converges to the same optimum. The key idea here is that we map a general CAM problem to a specific SIM problem. In addition to that we also want to map the optimal policy for a CAM to a optimal policy that can deal with the mapped version in SIM. Remember that we do not want to show all components of SIM here, since some of them are only relevant for the understanding and implementation.

Mapping CAM to SIM Given a CAM $M = (S, A, T, R)$ we define a mapping $f : CAM \rightarrow SIM$ as follows:

- $f(S, A, T, R) = (S', A', T', R')$
- $S' = \{(s, M) | s \in S \wedge M \in \mathfrak{P}(A)\}$
- $A' = \{I_a | a \in A\} \cup \{Wait\}$
- $T' : S' \times A' \times S' \times N \rightarrow [0, 1]$
- $R' : S' \times A' \times S' \times \mathbb{R} \rightarrow \mathbb{R}$

Since the state of SIM in very different from that of CAM, we need to describe the state transition t' using the state transition t from CAM and the reward function more clearly. We start with the state transition. As we can see above, a state in SIM is a tuple which consists of the state in CAM and a set of actions. The latter can be interpreted as an information saved in the state. This information tells which actions has already be initiated since the last *Wait* action. The transition at current Markov step x looks as follows:

- $t'((s, M), a') = \begin{cases} (s, M \cup \{a'\}) & \text{if } a' \neq Wait \\ (t(s, M), h_{x+1}) & \text{if } a' = Wait. \end{cases}$

The reward function can be defined very straight forward:

- $R'((s,M),a',t(s,M),\tau(a')) = \begin{cases} 0 & \text{if } a' \neq Wait \\ R(t(s,M)) & \text{if } a' = Wait. \end{cases}$

This mapping of the reward function makes both models easily comparable. Remember when we said that the T_{any} termination scheme in CAM is equivalent to the formalism of a *Wait* decision. Getting reward only after a *Wait* decision matches exactly the functioning of CAM when it gets only reward after termination of a single Markov step.

Until now everything seems to match well, but let us take a look at the following problem: In CAM the order of primary actions within a multi-action is not relevant. In our FOL domain in SIM, each *Initiation* can only be performed if its precondition is *true*, thus the order within an initiation batch is relevant. And it is not clear whether each mutual exclusion condition in the CAM model can be semantic equivalently mapped to such preconditions. This is the first thing we want to show.

MAPPING MUTUAL EXCLUSION Now we present a mapping for any mutual exclusion E in CAM to a set of preconditions $f(E)$ for all initiation operators in SIM, in the next paragraph we show that this mapping is semantically equivalent.

Each mutual exclusion rule can be expressed by a sentence of boolean logic. For each multi-action $X \in \mathfrak{P}(A)$ we can determine weather they can be run in parallel (legal) or not (illegal) by checking all mutual exclusion rules. We can define the set of multi-action $Y = \{X \in \mathfrak{P}(A) | X \text{ is legal}\}$. Then we can define following mapping:

$g : A \to \mathfrak{P}(A)$,

$g(a) = \{\bigcup_{x \in A}\{x\} | \forall X \in Y : \neg(a \in X \wedge x \in X)\} \setminus \{a\}$.

Informally, g maps a action a to all other actions with which a cannot run in parallel. Then each *InitiationOperator* $a_{I_i}(X)$ has following precondition: $\bigwedge_{a \in g(a_i)} go(a,X) = 0$.

LEMMA $\forall X \in \mathfrak{P}(A) : X \in Y \Leftrightarrow$ *Initiate*(X) does not hurt any precondition.

PROOF $X \in Y \Leftrightarrow \forall x \in X : X \setminus \{x\} \cap g(x) = \emptyset \Leftrightarrow$ *Initiate*(X) does not hurt any precondition.

■

Note that a nice side effect of this mapping is that the order of initiations does not matter for this mapped SIM. Now we can show that for all problems which can be expressed by a CAM can be expressed by SIM with same policy space. In order to that, we need to define a policy which simulates CAM in SIM.

- $\pi_{SIM} : \pi_{CAM} \to A'^{n+1}$, $n = |A|$, A is the multi-set chosen by π_{CAM}.
- $\pi_{SIM}(\{a_1,..,a_n\}) = (I_{a_1},..,I_{a_n},Wait)$.

This simulation takes a multi-action and transforms it straight forward to a initiation batch followed by a *Wait* decision. The last thing we need is a helping theorem to make things easier.

LEMMA For any problem p in CAM and any policy π_{CAM}, we can compute $f(p)$ and π_{SIM} such that

$$V^{\pi_{CAM}}(s) \leq V^{\pi_{SIM}}(f(s)),\ \forall s \in S.$$

PROOF Proof with Induction:
Induction start:
Assume π_{CAM} selects a multi-action $\{a_1, .., a_n\}$ for the initial state s_0. Then π_{SIM} can select for $f(s_0)$ the sequence $(I_{a_1}, .., I_{a_n}, Wait)$. In addition, there might be a better sequence we do not know about.
$\Rightarrow V^{\pi_{CAM}}(s_0) \leq V^{\pi_{SIM}}(f(s_0))$
Induction step: Analogue to induction start.

□

The last step is compare the value function for the optimal policy.

THEOREM 1 For any problem p in CAM and the optimal policy π^*_{CAM}, we can compute $f(p)$ and π^*_{SIM} so that

$$V(s) = V(f(s)),\ \forall s \in S$$

PROOF From *Theorem 2* we have $V(s) \leq V(f(s)),\ \forall s \in S$.
Since π^*_{CAM} is the optimal policy, $V(s)$ cannot be smaller than $V(f(s))$.
$\Rightarrow V(s) = V(f(s)),\ \forall s \in S$

■

4.2. CAM AS GENERAL AND OPTIMAL AS SIM

Now we want to show that CAM and SIM are actually equal w.r.t. the policy space and their optimality. To do that, we do the same proofs like in the previous section, but the other way around.

MAPPING SIM TO CAM Given an arbitrary SIM X according to the definition in Section 1.2, we define the mapping $f : SIM \rightarrow CAM$ as follows:

- $f(X) = (S, A, T, R, I)$
- $S = \{RelationalState(i) | i \in \mathbb{N}\}$
- $A = [\{a \in \mathfrak{P}(ActivitySpace) | g(a)\} \times \{I.clear()\}] \cup \{I.add(d)\}$
  - $g(a) = \begin{cases} true & \text{if } \exists (a_1, .., a_n) : a_1, .., a_n \in a \wedge n = |a| \wedge (Initiate(a_1, .., a_n), Wait) \\ & \text{can be executed in } X \\ false & \text{else .} \end{cases}$
  - $d \in ActivitySpace$
  - $I.add(d)$ will add d to I
  - $I.clear()$ will set $I = \emptyset$
- $T : S \times A \times S \times N \rightarrow [0, 1]$
- $R : S \rightarrow \mathbb{R}$
- I... a set with *ActivitySpace* for element type, in which the initiations between two *Wait* decisions are saved.

Although S contains all possible configuration of the relational state, we do not need all of them. In SIM, the relational state is changed, as soon as a activity is initiated. But in CAM with a T_{any} option, we only care about the result after the *Wait* decision is performed. Therefore CAM naturally only cares about the state after a *Wait* decision. And the state transition look rather simple. For the definition we use the state transition t_X from X:

- $t(S, A) = t_X(S, (Initiate(A), Wait))$

For the reward function we need to make assumptions on the SIM we want to map. While the SIM works with continuous time, the CAM uses discrete micro time steps for the duration of actions. The assumption we make is that

$$\forall a \in ActivitySpace : go(a, X) \in \mathbb{N}$$

and therefore

$$\tau(d) \in \mathbb{N}.$$

These are only two different styles and the choice between them will not affect the generality of the model. In actual computations, a real valued number must be represented by float number anyway, which can be bijectively mapped to integers.

- $R(s, a) = \sum_{s'} \sum_{\tau(d)} P(s', \tau(d) | s, d) R(s, d, s', \tau(d))$

MUTUAL EXCLUSION The set of mutual exclusion rules is empty since there is no need for that. $A \in X$ is not the full power set of the action space. The restriction $g(a)$ (see above) does a implicit mapping of the preconditions.

MAPPING THE POLICY Using I as a helping variable, the mapping is simple:

- $\pi_{CAM} : \pi_{SIM} \rightarrow A$
- $\pi_{CAM}(\pi_{SIM}(s)) = \begin{cases} I.add(\pi_{SIM}(s)) & \text{if } \pi_{SIM}(s) \neq Wait \\ (I, I.clear()) & \text{else .} \end{cases}$

Whenever a initiation is made, no action is executed. Only the helping variable I grows. Once a *Wait* decision is made, all actions in I are executed as a multi-action and then I is reset to the empty set, because we use the T_{any} termination scheme.

THEOREM 2 Given any problem X in SIM with discrete time steps $\tau(d)$ for decisions d and an optimal policy π^*_{SIM}, we can compute the mapping $f(X)$ and $\pi^*_{CAM} = f(\pi^*_{SIM})$ such that

$$V^{\pi^*_{SIM}}(s) = V^{\pi^*_{CAM}}(f(s)),\ \forall s \in S$$

PROOF Given such a X and a optimal policy π^*_{SIM}
Define:

- $d := \pi^*_{SIM}$,
- $d(s) := \pi^*_{SIM}(s),\ s \in RelationalState$,
- $a := \pi_{CAM}(\pi^*_{SIM})$,
- $a(s) := \pi_{CAM}(\pi^*_{SIM}(s)),\ s \in RelationalState$

It is obvious that

$$\forall s \in S : \left[d(s) = Wait \wedge M = (a_1, .., a_n) \text{ are active}\right] \Leftrightarrow \left[a(f(s)) = (I, I.clear()) \wedge I = M\right]$$

$\Rightarrow \forall t \in \mathbb{N}, s_t \in S : P(s_t, d(s_t), s_{t+1}) = T(f(s_t), a(f(s_t)), f(s_{t+1}), \tau(d(f(s_t))))$
$\Rightarrow \forall t \in \mathbb{N}, s_t \in S : V^{\pi^*_{SIM}}([s_t, s_{t+1}, ...]) = V^{\pi^*_{CAM}}([f(s_t), f(s_{t+1}), ...])$
$\Rightarrow \forall s \in S : V^{\pi^*_{SIM}}(s) = V^{\pi^*_{CAM}}(f(s))$

■

4.3. RESULTS

COROLLARY CAM and SIM have the same policy space and within that, they converge to the same optimum.

PROOF This follows directly from *Theorem 1* and *Theorem 2*.

■

5. RUNTIME ANALYSIS

This section aims to show the difference between SIM and CAM w.r.t. the worst case runtime when the Upper Confidence Tree [1] algorithm (UCT) is used, which is state-of-the-art for concurrent action planning. The convergence speed in general is hard to determine theoretically. It heavily depends on the environment and the reliability of actions. But we can see this as a black box in order to be able to focus on the actual differences between these models. We first present the UCT algorithm our analysis is based on.

5.1. UPPER CONFIDENCE TREE

The basic idea of this algorithm is that multiple simulations of the system determines how good an action is. These simulations are done according to a *Rollout Policy*. In many cases, this is just a simple Monte Carlo procedure, in which the actions are chosen u.a.r. in each simulation step. Of course, more sophisticated algorithms can be used, especially if useful domain information are given. But we assume the simple Monte Carlo simulation, since it works for all domains. A rollout must start at some node. We want to chose the node which is most likely to return the best result. This choice is based on the *Upper Confidence Bound* (UCB) value.

Given is a node v, then $r(v)$ is the sum of reward v has received so far, $n(v)$ is the number of times a rollout has performed on v and n^* is the total amount of rollouts that has ever been performed, then the UCB value is:

$$UCB := \frac{r(v)}{n(v)} + \beta\sqrt{\frac{2ln(n^*)}{n(v)}}$$

Note: β is a tunable parameter, typically set to 1. This tuning is not relevant for us.

The *Tree Policy* is the function returning a node for the rollout. It always starts at the root node and iteratively chooses child nodes with the highest UCB value until a leave node is reached. Then we expand it. That means, we create a node for each action in the legal action space and return one of them, u.a.r. chosen. Now we can do a rollout with this node which returns a reward. *Update* takes this reward and adds it to the old reward value of all nodes between the rollouted node and the root, both of them included. Also, their numbers of rollouts are incremented.

These steps are repeated until a given computation resource is used up. This procedure called *Exploration*. The Exploration converges to the optimum. After that, a real action is now performed, and we explore again, etc. There are domains in which it is not guaranteed, that

a final state can be found. But we know that in each step, the value function gets closer to the optimum. So in order to guarantee the termination, we can give a small number $\epsilon \in \mathbb{R}$, such that the algorithm terminates if the difference between the expected utility of the current state and the expected utility of the previous state is smaller than ϵ. The pseudocode is shown below.

```
while (V_{n-1}-V_n)>ε:
        while in computation resource:
                x = tree_policy
                r = rollout(x)
                update(x,r)
        play best child
```

5.2. Runtime Difference between SIM and CAM under UCT

We assume that the size of the problem n is the cardinality of the action space $|A|$. This is reasonable because the Exploration traverses the action space in breadth and a sequence of actions in depth of the tree. Furthermore we want to make statements independent from the domain.

First, we want to show an important correlation between SIM and CAM w.r.t. runtime.

Lemma Given a concurrent action problem P with action space A, $n = |A|$, then

$$\forall t \in \mathbb{N} : \exists k \leq n : V^{\pi^*_{CAM}}([s_t, s_{t+1}) = V^{\pi^*_{SIM}}([s_t, s_{t+1}..., s_{t+k+1}])$$

This means that SIM needs to perform $\mathcal{O}(n)$ more actions than CAM to achieve the same expected utility.

Proof

1. Assume that a solution with SIM can do better than $\mathcal{O}(n)$ times more steps. Then there would exist a simulation according to Section 4.2 such that each initiation batch $(Initiate(I_{a_1}, .., I_{a_k}), Wait)$ can be consolidated to a single multi-action with same expected utility. Since $k \leq n$, CAM needs $\mathcal{O}(n)$ less steps than SIM. $\lightning$
2. Assume that a solution with CAM can do better than $\frac{1}{\mathcal{O}(n)}$. Then there would exist a simlation according to Section 4.1 such that each multi-action $(a_1, .., a_k)$ can be performed in SIM with $(Initiate(I_{a_1}, .., I_{a_k}), Wait)$ with same expected utility. Since $k \leq n$, SIM only needs $\mathcal{O}(n)$ more steps than CAM. $\lightning$

■

We can now use this lemma to compare runtime of SIM and CAM under UCT, considering the following conclusion. Whenever there is a black box we do not want to investigate further and substitute it with variable λ or if one value on one side is already given, then $\lambda_{CAM} = \mathcal{O}(n)\lambda_{SIM}$.

For each line in the pseudocode above we can assign a runtime expression to it, given d as the depth and b as the breadth of the search tree:

- The outer loop is the actual black box i mentioned at the beginning of this chapter. We simply set it to X.
- The runtime of the inner loop is a parameter which can be tuned. For real computation, there is a trade off between cost of time and precision of approximation. For simplicity, we assume that this value Y is chosen from $Y_{CAM} \in \mathcal{O}(n)$ for CAM. According to the Lemma above we need to choose $Y_{SIM} \in \mathcal{O}(n^2)$ for SIM to make it comparable.
- The Tree Policy iteratively chooses children until a leaf child is found. In each iteration, every child has to be looked at to determine their UCB value. So the runtime is $\mathcal{O}(d \cdot b)$.
- For the Rollout Policy we must give a maximum number of iteration steps because of termination problems we discussed above. Just like the runtime for the inner loop, this can treated like a tunable parameter with the same trade off. Therefore we also assume its runtime value Z to be chosen from $Z_{CAM} \in \mathcal{O}(n), Z_{SIM} \in \mathcal{O}(n^2)$.
- Update makes modifications beginning from a leaf up to the root. So its runtime is $\mathcal{O}(d)$.

Summarizing it up, we have:

```
while (V_{n-1}-V_n)>ε: -- X
        while in computation resource: -- Y
                x = tree_policy -- O(d·b)
                r = rollout(x) -- Z
                update(x,r) -- O(d)
        play best child -- O(1)
```

The last thing we need to know is, what the depth and breadth for CAM and SIM are. Trivially, $b_{CAM} \leq 2^n \in \mathcal{O}(2^n)$ and $b_{SIM} \leq n+1 \in \mathcal{O}(n)$. For the depth, we need to count how many times is the tree expanded and how many times is it cut. Whenever Tree Policy is performed, the tree grows by one node. In worst case the depth also grows by one. Whenever the best child is played, the depth of the tree reduces by one. So, in worst case $d = X \cdot Y - X \Rightarrow d_{CAM} \in \mathcal{O}(X \cdot n), d_{SIM} \in \mathcal{O}(X \cdot n^2)$

And we can now conclude:

- mean valued runtime for CAM $R_{CAM} = \mathcal{O}(X^2 \cdot 2^n)$
- mean valued runtime for SIM $R_{SIM} = \mathcal{O}(X^2 \cdot n^5)$

This means, with problem size size $n = |A|$ UCT performed in CAM has exponential time complexity and in SIM polynomial time complexity, thus SIM performs better.

6. Discussion

The result of the proof of equivalence was as we have expected. Both CAM and SIM are sMDPs. The way how the decision *Wait* in SIM works matches exactly the T_{any} termination scheme. Additionally, a multi-action heavily correspond to a batch of initiations followed by a *Wait* decision. But the difference is that SIM needs more Markovian steps for that. And this has an influence on the runtime. This might be counter intuitive because of the following reason. Each node of the search tree for CAM has potentially 2^n children, but it is also true that for SIM at depth n it has potentially 2^n leafs. If in CAM we have to traverse through all children, why do we need not to traverse through all the leafs in SIM? What is the difference? The answer is given, if we simply look at how *Tree Policy* works. At any node v, only the children of v are looked at. Other nodes at the same tree level as the children of v do not matter. And this of course also holds at depth n. It is also easy to see that CAM does not get a 'better' node than SIM. CAM chooses the best child, but at the same time SIM also chooses the best leaf at depth n, because this is a property of *Tree Policy* which always holds.

Now we indeed know that SIM has better runtime than CAM under UCT but we still do not exactly know by how much, since we still have the black box X. If X is polynomial or better, then SIM is really much better than CAM, otherwise the difference is not significant.

Further investigation could address exactly this unknown X. Since this is the outer loop, it describes the convergence speed of UCT used with CAM and SIM. Before diving into theoretical research again, one could do some experiments first, measuring the runtime of CAM and SIM under UCT for various problems. Another interesting topic is the average case which is typically more sophisticated but also harder to determine. Related to that we could also search for a specific classification of concurrent action problems. If there exist one class of problems for which we only need a few multi-actions, then the solution should definitely be described in CAM. One could even try to modify UCT itself in a way that SIM works better with.

Appendices

A. Einleitung

Wenn es um das nebenläufige Planen geht, eigenen sich Markoveinscheidungsprozesse (MEP) für diskrete Zeitschritte und Semi-Markoveinscheidungsprozesse (sMEP) für kontinuierliche Zeit immer sehr gut, um solche Probleme zu beschreiben. Unser Formalismus ist eine Instanz der sMEP. Die eigentliche Motivation dahinter, diesem Formalismus zu schaffen war, dass wir eine einfache und performante Implementierung für Umgebungen in Prädikatenlogik erster Ordnung haben wollten. Da sie gute Leistung gezeigt hatte, waren wir interessiert an den theoretischen Eigenschaften unseres Formalismuses. Der beste Weg das zu tun, ist es, ihn mit einem ähnlichen bereits exitstierenden Formalismus zu vergleichen. Dafür zogen wir den Concurrent Action Model heran. Wir beginnen mit einer Beschreiben eben dieses Modells und fahren fort mit der Definition unseres Modells, in dem wir zunächst die Syntax, dann die Semantik defineren. Direkt im Anschluss Zeigen wir seine Optimalität in der eigenen Taktikhülle, in dem wir die Bellmanoptimalitätsgleichung darauf anwenden und zeign, dass dessen Werteiteration konvergiert. Diese Tatsache ist für weitere Untersuchungen notwenig. Und nun werden die beiden wichtigsten Themen dieser Arbeit diskutiert:

- Beweis der Allgemeinheit und Optimalität unsers Formalismuses
- Laufzeitvergleich der beiden Formalismen

Die grundlegende Idee, um die Allgemeinheit und Optimalität zu zeigen, ist, eine Reduktion in beiden Richtungen zu führen, eine Simulation zu definieren und zu zeigen, dass sie zum selben Optimum konvergieren. Für den Laufzeitvergleich verwenden wir den Upper Confidence Tree-Algorithmus, welcher eine Instanz von Monte Carlo-Algorithmen ist. Wir geben einen Pseudocode an, den wir zunächst unabhängig des Modellwahls auf Laufzeit analysieren. Dann wenden wir diesen Algorthmus auf beide Modelle an und ermitteln den Unterschied. Am Ende wird über die Ergebnisse diskutiert.

REFERENCES

[1] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin Heidelberg, 2006.

[2] Khashayar Rohanimanesh and Sridhar Mahadevan. Coarticulation: An approach for generating concurrent plans in markov decision processes. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 720–727, New York, NY, USA, 2005. ACM.

[3] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift: 

Stuttgart, 05.11.2015

Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Signature: 

Stuttgart, 5th of November, 2015