Time Constrained MDPs
16.420-Project Report

Alborz Geramifard-921700902
agf@mit.edu

1 Introduction

In some planning problems, there are time dependent constraints over the trajectory of states. For example consider a scenario where an unmanned underwater vehicle (AUV) is exploring deep-sea areas. As Figure 1 highlights, the AUV is dropped at a certain area and is going to be fished in another location within a timeline. While mapping the area along the way is a static desired behavior, in certain periods of time, interesting phenomena going to happen, and the research team requires the AUV to be present and captures such incidents. While in some cases such information are not available beforehand, through this work we assume that the agent has prior access to such timelines. In particular, we are interested in the problem of finding the trajectory which satisfies such constraints while maximizing the reward along the way. We begin by providing the formal problem formulation in the next section. Then, we introduce Soft Dynamic Programing (**Soft-DP**) and **DP**^{2} as two algorithms to deal with such problems and discuss their space and time complexity. Further we extend these methods to tackle the problem of planning under uncertain transition model. We empirically examine the quality and speed of the proposed algorithms. Finally we conclude.

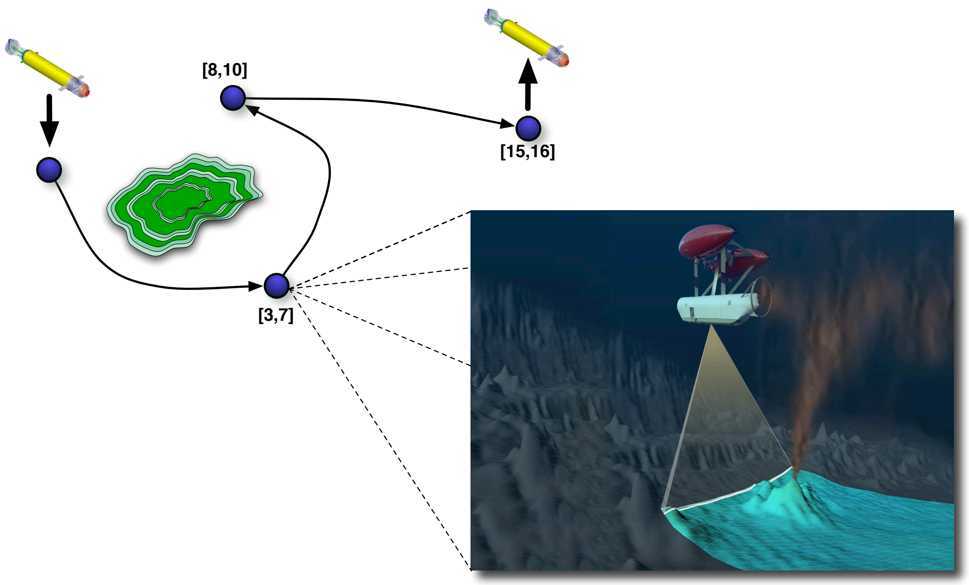

Figure 1: An example exploring scenario using a AUV: (a) timing constraints through the trajectory (b) A simulation shot showing a submarine capturing data from an erupting underwater volcano [MBA, 2008]

2 Problem Formulation

Markov Decision Processes (MDPs) is a rich framework for modeling sequential decision making problems (*e.g.* see [Sutton and Barto, 1998]). We formulate our problem as an extended version of MDPs called Time Constrained MDPs (TCMDPs). A TCMDP is defined as a tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma, T, D^a, \mathcal{H})$ in which $\mathcal{S}$ is the set of States, $\mathcal{A}$ is the set of actions, $\mathcal{P}^a_{ss'}$ is the probability of getting to state s' when executing action a from state s, $\mathcal{R}^a_{ss'}$ is the corresponding reward, $\gamma \in [0,1]$ is the discount factor, $T : \bar{\mathcal{S}} \subset \mathcal{S} \rightarrow (t_{min}, t_{max}), t_{min} \leq t_{max}$ is the time constrain which maps state s to a bound, meaning that state s must be visited within $[t_{min}, t_{max}]$ at least once, D^a is the duration for action a, and $\mathcal{H}$ is the planning horizon.[1] Solving a TCMDP in the general case is very challenging. Therefore we will have the following assumptions, though we will extend the results to the stochastic case later.

- $\forall a \in \mathcal{A}, D^a = 1$
- $\mathcal{P}^a_{ss'}$ is deterministic.
- $\forall s, s' \in \mathcal{S}, s \neq s', T(s) \cap T(s') = \emptyset$
- $\forall s \in \mathcal{S}, T(s) = (t_{min}, t_{max}) \in \mathbb{N} \times \mathbb{N}$
- $\mathcal{H} = \max_{t_{max}} \{T(s) = (t^s_{min}, t^s_{max}) | s \in \bar{\mathcal{S}} \subset \mathcal{S}\}$

Since we assume that there is no time overlap between time constraint intervals, we define s^t as the state corresponding to the constraint at time t if such constraint exist. For example $s^{\mathcal{H}}$ is the state corresponding to the last time constraint at the end of horizon. Further, we address the lower and upper bound of an existing constraint at time t with t^s_{min} and t^s_{min}. We are interested in policies generating trajectories of length $\mathcal{H}$ starting from any state, satisfying all time constraints T, while maximizing the sum of discounted rewards along the way.

$$V^*(s) \quad = \quad E\left[\sum_{t=1}^{\mathcal{H}} \gamma^{t-1} r_t \middle| s_0 = s, \pi^*\right].$$

3 Proposed Algorithms

3.1 Soft Dynamic-Programing

Algorithm 1 : **Soft-DP** ($penalty$)

```
For all $s \in \mathcal{S}$
    $V(s, \mathcal{H}, False) = penalty$
    $V(s, \mathcal{H}, True) = 0$
$V(s^{\mathcal{H}}, \mathcal{H}, Flase) = 0$

For t = $\mathcal{H} - 1$ to 0
    For $f \in \{True, False\}$
        For $s \in \mathcal{S}$
            $V(s,t,f) = \max_a E\left[r^a_{(s,t,f)(s',t+1,f')} + \gamma V(s', t+1, f')\right]$
            $\pi(s,t,f) = argmax_a E\left[r^a_{(s,t,f)(s',t+1,f')} + \gamma V(s', t+1, f')\right]$
```

One approach to deal with TCMDPs, is to turn all hard constraints into soft constraints by penalizing the agent for not meeting any of the timing constraints along the way (extra negative reward = $penalty$). Additionally to translate the TCMDP into conventional MDPs, we have to change the state definition in order to be independent of the past history. To do so, we include the time step together with a flag stating whether the possible time constraint related to that time has been met so far or not. Notice that one flag is enough since according to our assumption there is no overlap

[1]Notice that we can set $\gamma = 1$, as the planning horizon is limited.

between time constraints. In general, the number of flags equals to the maximum overlapping time constraints. In particular,

$$\mathcal{S}' \quad = \quad \{(s,t,f)|s \in \mathcal{S}, t \in \{0,...,\mathcal{H}\}, f \in \{True, False\}\}\,.$$

Respectively the new transition function is defined over $\mathcal{S}' \times \mathcal{A} \times \mathcal{S}'$ in which s evolves according to $\mathcal{P}^a_{ss'}$, t increments after each move by one, and f is set to $True$, only if it was $True$ and the current and next time constraint both are the same, or if the new state meets the corresponding time constraint. The new MDP can be solved efficiently with one backward sweep of dynamic programing. Algorithm 1 shows **Soft-DP** in detail. At the beginning the algorithm initialize the states at the end of horizon ($t = \mathcal{H}$). Notice that only states $s \neq s^{\mathcal{H}}$ for which flag is $False$ are panelized. **Soft-DP** performs backward DP from time $\mathcal{H}-1$ to 0, over all possible states. we already talked about how to find f'. $r^a_{(s,t,f)(s',t+1,f')}$ is calculated by the sum of TCMDP reward, $R^a_{ss'}$, and the *penalty* only if the agent passes a deadline while still have not met the constraint. (*i.e.* $t = t^s_{max}$ and $f = False$).

The space complexity of **Soft-DP** is $O(\mathcal{H}|\mathcal{S}|)$, while its running time is $O(\mathcal{H}|\mathcal{S}||\mathcal{A}|)$ in the deterministic case and $O(\mathcal{H}|\mathcal{S}|^2|\mathcal{A}|)$ in the stochastic case. An interesting fact about the resulting V and π is that it captures two policies for each state-time pair. The conservative policy ($f = False$) to satisfy the constraint while maximizing the reward along the way, and the aggressive policy ($f = True$), where the agent is no longer worried about the constraint and only focuses on maximizing the reward along the way. As time evolves, the agent switches back and forth between such policies. We will talk further about how the agent uses its policy to act in the world in the following sections.

3.2 Deterministic DP^2

Another approach for solving TCMDPs with the assumptions made earlier is to run backward-DP with the goal of only maximizing the reward while logging time consistencies (C). This will build the aggressive policy. Whenever, a lower bound of a time constraint is found, we run another DP to find the conservative policy within the time constraint interval as well and switch back to find the rest of the aggressive policy based on the last step of conservative data in order to assure consistency along the plan.

Algorithm 2 : **Deterministic DP^2**

```
For all $s \in \mathcal{S}$
    $V(s, \mathcal{H}, False) = 0$
    $C(s, \mathcal{H}, False) = False$
$C(s^{\mathcal{H}}, \mathcal{H}, False) = True$

For t = $\mathcal{H}$ to 0
    $f = False$
    if $t == t^s_{min}$
        Deterministic DP-Interval($V, \pi, C, t^s_{min}, t^s_{max}$)
        $f = True$
    For $s \in \mathcal{S}$
        Deterministic Update($V, \pi, C, s, t, f, False$)
```

Algorithm 6 shows the detail of this new approach named DP^2. C keeps track of the consistency of all states. A state is consistent if it is guaranteed to meet all future constraints following the resulting policy. The algorithm starts by initialization all states at the end of the horizon. All values at the end of horizon are zero and only state $s^{\mathcal{H}}$ is consistent. On each timestep, **DP^2** checks to see if it hits the lower bound of any constraints. If that happens, it calls **Deterministic DP-interval** on the constraint's interval, and set f to $True$, meaning that the next aggressive DP must use the conservative results as opposed to use aggressive results ($f = False$)

The **Deterministic DP-interval** function, shown in Algorithm 3, copies the values (V) and consistencies (C) of aggressive policy ($f = False$) to the conservative part ($f = True$). Afterwards it sets all values of conservative section for which the constraint did not meet to $-\infty$. This will

Algorithm 3 : **Deterministic DP-Interval**(V, π, C, t_1, t_2)

```
For $t = t_1$ to $t_2$
    For $s \in \mathcal{S}$
        $V(s, t, True) = V(s, t, False)$
        $C(s, t, True) = C(s, t, False)$
        if not $C(s, t, True)$
            $V(s, t, True) = -\infty$
For $t = t_2 - 1$ to $t_1$
    For $s \in \mathcal{S}$
        if $C(s, t, True) == False$
            Deterministic Update($V, \pi, C, s, t, True, True$)
```

discourage any state to backup from such states. Finally it calculates the conservative policy for all inconsistent states by calling the **Deterministic Update** sub-function.

Algorithm 4 : **Deterministic Update**($V, \pi, C, s, t, source, target$)

```
$\pi(s, t, target) = argmax_a\, [r^a_{ss'} + \gamma V(s', t+1, source)]$
$V(s, t, target) = \max_a\, [r^a_{ss'} + \gamma V(s', t+1, source)]$
if $s \neq s^t$ and $t == t^s_{max}$
    $C(s, t, target) = False$
else
    $C(s, t, target) = C(s', t+1, source)$
```

The main update part is being done by the **Deterministic Update** sub-function as shown in Algorithm 4. Notice that $source$ is used as the base of backup and $target$ specifies where information should be backed up to. These two variables can be either $True$ meaning conservative or $False$ meaning aggressive. After performing the famous Bellman backup on the value and policy, the algorithm updates the consistency as follows: if the current time is equal to the upper bound of a constraint and the state could not satisfy it then it is not consistent, otherwise it is backed up from the next state according to the selected action.

I suspect that one can prove that **DP**^{2} calculates the optimal policy in the shortest possible computation. My intuition comes from the fact that **DP**^{2} calculates conservative data only if it is needed along the trajectory. In general the space complexity of the algorithm is $O(\mathcal{H}|S|)$, and the time complexity in the worst case is $O(\mathcal{H}|S||A|)$ which is the same for **Soft-DP** .

3.3 Following the policy

For both methods the trajectory can be derived given π from any starting position. The agent should always follow the aggressive policy ($f = False$) unless it is within an unsatisfied constraint interval where it should switch to the conservative policy ($f = True$) till it satisfies the constraint.

4 Stochastic Case

In this section, we relax the assumption of having deterministic transition models and elaborate on adopting previous algorithms to the stochastic case. The **Soft-DP** algorithm already turned the hard constraints into soft constraints, therefore the same algorithm can be used for the stochastic case as well. Extending **DP**^{2} requires more work though. The notion of time consistency (C) is no longer a binary variable, but a probability which can be between $[0, 1]$. In most stochastic domains it is highly unlikely to generate a trajectory which satisfies all time constraints with probability 1, hence we add the acceptable risk as ϵ. The new objective is to find the trajectory which has at least $1-\epsilon$ probability of success (meeting all constraints along the way), while maximizing the sum of discounted rewards. The higher the acceptable risk is, the more chance for the agent to take aggressive moves along the way and gather rewards. The **DP**^{2} algorithm should adopt to include the new risk parameter (ϵ), and extend to the stochastic case. Algorithm 5 shows the **DP**^{2} in the general form.

Algorithm 5 : **DP**$^2(\epsilon)$

```
For all $s \in \mathcal{S}$
    $V(s, \mathcal{H}, False) = 0$
    $C(s, \mathcal{H}, False) = 0$
$C(s^{\mathcal{H}}, \mathcal{H}, False) = 1$

For t = $\mathcal{H}$ to 0
    $f = False$
    if $t == t^s_{min}$
        DP-Interval($V, \pi, C, t^s_{min}, t^s_{max}, \epsilon$)
        $f = True$
    For $s \in \mathcal{S}$
        Update($V, \pi, C, s, t, f, False, \epsilon$)
```

Notice the change to the main body of **DP**^{2} : switching C values from boolean to numbers and passing $epsilon$ to both sub-functions.

Algorithm 6 : **DP-Interval**($V, \pi, C, t_1, t_2, \epsilon$)

```
For $t = t_1$ to $t_2$
    For $s \in \mathcal{S}$
        $V(s, t, True) = V(s, t, False)$
        $C(s, t, True) = C(s, t, False)$
        if $C(s, t, True) < 1 - \epsilon$
            $V(s, t, True) = -\infty$
For $t = t_2 - 1$ to $t_1$
    For $s \in \mathcal{S}$
        if $C(s, t, True) < 1 - \epsilon$
            Update($V, \pi, C, s, t, True, True, \epsilon$)
```

The **DP-Interval** sub-function only changes in places where C values are considered in **if** statements. With a probabilistic approach, we define the consistency for a state as the probability of a success when starting from that state and follow the resulting policy. We can say a state is inconsistent if its consistency is less than $1 - \epsilon$.

Algorithm 7 : **Update**($V, \pi, C, s, t, source, target, \epsilon$)

```
$\pi(s, t, target) = argmax_a E_{s'} [r^a_{ss'} + \gamma V(s', t + 1, source)]$

if not satisfyConstraint and $t == t^s_{max}$
    $C(s, t, target) = 0$
else
    $C(s, t, target) = E_{s'|\pi}[C(s', t + 1, source)]$

if $C(s, t, target) < 1 - \epsilon$
    $V(s, t, target) = -\infty$
else
    $V(s, t, target) = E_{s'|\pi} [r^a_{ss'} + \gamma V(s', t + 1, source)(1 - \delta(V(s', t + 1, source), -\infty)]$
```

The most substantial changes happen to the **Update** sub-function. First the policy is calculated according to the Bellman backup rule. Notice that now we included the expectations to count for the uncertainty of the transition model. The C value are now calculated probabilistically. If the current time and position means failing a constraint, then C is set to zero, otherwise it would be the expected success rate. Updating the values is a bit tricky. First, if the current state does not have the least success rate ($1 - \epsilon$), we set its value to $-\infty$ to prevent other states backing up from it. If the state is consistent, we take the expectations ignoring states with $-\infty$ values. The δ function is:

$$\delta(i,j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

5 Empirical Results

5.1 Deterministic Case

In order to empirically demonstrate the algorithms proposed so far, we ran **Soft-DP** and **DP**^{2} algorithms in grid-world domains. The agent has 3 actions: Left, Right (moves the agent one step in each direction), and Wait. For the first test case, we picked a toy deterministic world shown in Figure 2. There are 3 states, and two time constraints in this environment located on the both sides. The $+10$ reward is given to all transitions ending up in state 2, for all other transitions the reward is 0. We used $\gamma = 1$. The solutions of **Soft-DP** and **DP**^{2} are shown in Figures 3 and 4. Time evolves along Y-axis. V is shown on green and red scale corresponding to positive and negative values, while C is shown in green and white corresponding to $True$ and $False$. Notice that both policies are identical. Although **DP**^{2} did not consider a conservative policy for the times outside of time constraints intervals. Having all states at time zero consistent (C is totally green) means that regardless of the starting point the toy TCMDP has a solution. For example the optimal trajectory from state 1 is: right, left, right, wait, right which can be derived from both solutions. The running time of both methods is very close, with **DP**^{2} being on the lead as it requires less computation on average. In the best case **DP**^{2} can be two times faster that **Soft-DP** . We think ideas about saving time can carry over from **DP**^{2} to **Soft-DP** algorithm as well, which will be interesting for the future work.

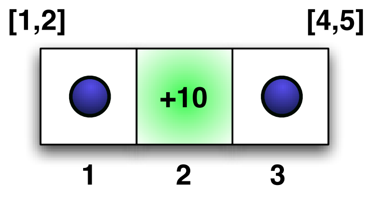

Figure 2: A toy deterministic TCMDP

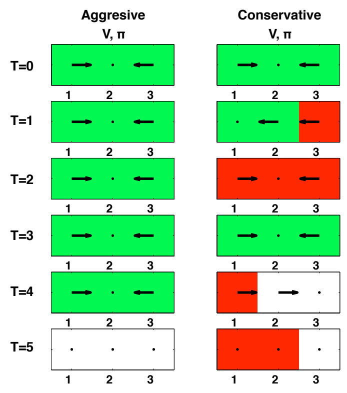

Figure 3: **Soft-DP** Solution of the toy TCMDP

5.2 Stochastic Case

We used the same toy domain explained in the previous section although we added 5% failure probability to all movements, meaning that there is 5% chance that the agent stays in the same

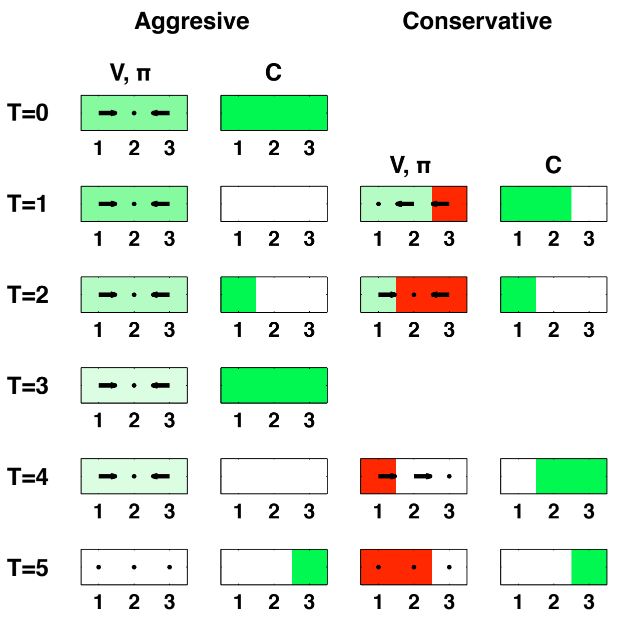

Figure 4: **DP**^{2} Solution of the toy TCMDP

position taking a move action from. Notice that wait action wont suffer from this change. We also set $\epsilon = 0.1$ for **DP**^{2} . The solution of **Soft-DP** and **DP**^{2} are shown in Figures 5 and6. For **DP**^{2} C values are now scaled between [0,1]. Notice the change between the resulting policy between **Soft-DP** and **DP**^{2} algorithms. **Soft-DP** 's policy is always aligned in the direction of satisfying constraints because of the uncertainty. For example the aggressive policy for state 2 at time 3, is to move right. Even though staying at the same position will result into $+10$ reward, **Soft-DP** 's policy prefers to move towards the next state with the time constraint, state 3. On the other hand **DP**^{2} inspected the probabilistic consistencies at each time step and could trade off the risk for higher amount of reward along the way. The resulting policy is not different from the deterministic case. Looking at the C values at time 0, we can see that only state 3 has inconsistent success rate ($C < 0.9$). Now The question is now how well such policies perform. We ran 10,000 trajectories from state 1 using both policies and calculated the average success rate and return. If a trajectory failed, we set its return to 0.

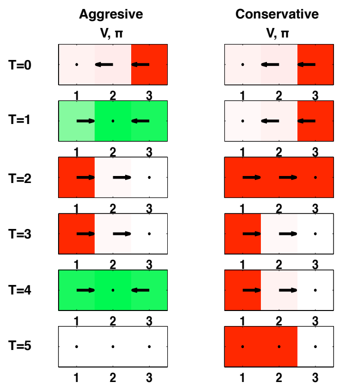

Figure 5: **Soft-DP** Solution of the stochastic toy TCMDP

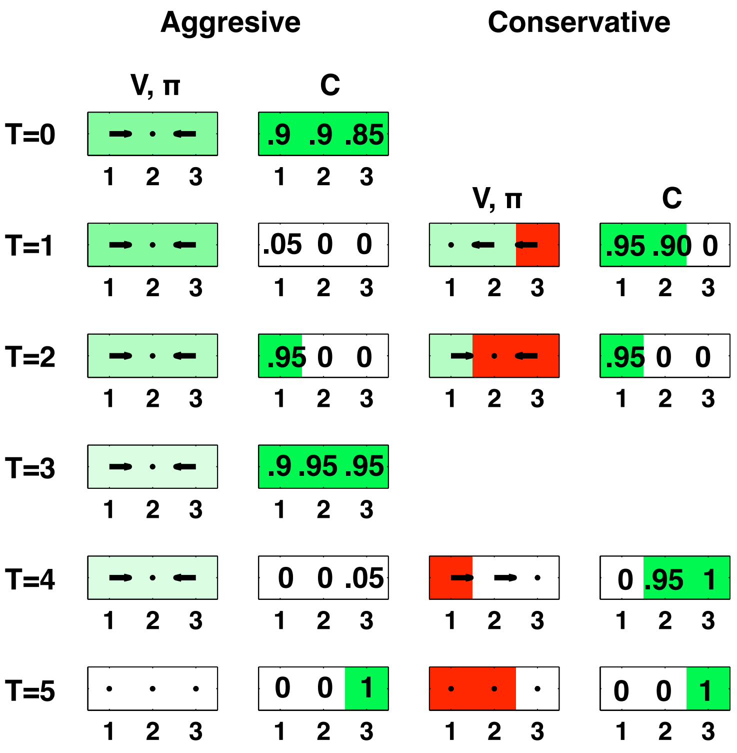

Figure 6: **DP**^{2} Solution of the stochastic toy TCMDP

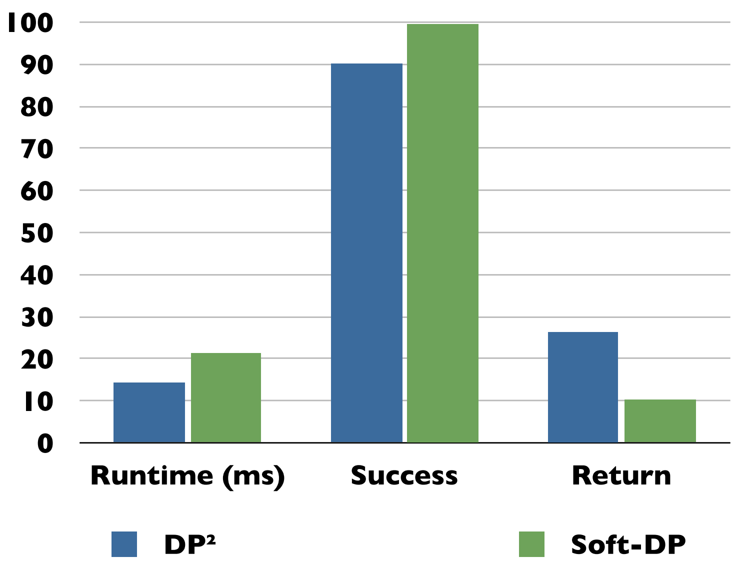

Figure 7: Averaged results of **DP**^{2} and **Soft-DP** for the stochastic toy TCMDP

Figure 7 shows the running time in milliseconds to compute the policies together with the averaged success rate and the cumulative reward (Return) for **DP**^{2} and **Soft-DP** methods. As expected **DP**^{2} algorithm has lower running time. An stark observation can be observed by comparing 2nd and 3rd bars. **Soft-DP** had a higher success rate compared to **DP**^{2} . Although this translated into less return value. On the other hand, **DP**^{2} traded off the success rate by taking advantage of the risk parameter ($\epsilon = .1$) and gathered more reward on average.

6 Conclusion

Through this work, we introduced TCMDP framework which can be used to represent many research problems. we also introduce **Soft-DP** and **DP**^{2} as two algorithms for solving such problems under certain assumptions. **DP**^{2} seems to be a promising algorithm which can trade off risk for higher rewards in stochastic problems. Perhaps one can map such a risk parameter to the $penalty$ in the case of **Soft-DP** , although such a mapping is not immediately obvious to us. Future work includes, relaxing more assumptions such as having actions with arbitrary execution time and continuous timesteps. Perhaps such ideas can carry over to the realm of POMDPs as well. Finally it would be interesting to run these algorithms on large scale problems as their running time suggests nice scalability.

7 Acknowledgement

we would like to give credit to Tom, Emma, Nick and Prof. Bertsekas, who kindly helped me mature some ideas through this work.

References

[MBA, 2008] Monterey bay aquarium research institute. `http://www.mbari.org/`, 2008.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.