Risk-Aware and Multi-Objective Reinforcement Learning with Distributional Monte Carlo Tree Search

Conor F. Hayes*[,1], Mathieu Reymond[2], Diederik M. Roijers[2,3], Enda Howley[1] and Patrick Mannion[1]

[1]School of Computer Science, National University of Ireland Galway, Ireland
[2]AI Lab, Vrije Universiteit Brussel, Belgium
[3]HU University of Applied Sciences Utrecht, the Netherlands
*email: c.hayes13@nuigalway.ie

Introduction

- In certain scenarios, the utility of a user is derived from a single execution of a policy
  - Making decisions based on the expected value is not sufficient
  - *Need* a distribution over the returns
- Distributional Monte Carlo Tree Search learns a distribution over the returns of different policy executions
- Distributional Monte Carlo Tree Search learns good policies under risk-aware and SER problem domains
- Distributional Monte Carlo Tree Search achieves state-of-the-art performance in ESR settings

Background

- Multi-Objective Reinforcement Learning
- SER and ESR
- Expected Utility Policy Gradient
- Bootstrap Thompson Sampling
- Monte Carlo Tree Search

Multi-Objective Reinforcement Learning

- Decision problems with multiple objectives
- Multi-Objective Markov Decision Process
  - M = (S, A, T, γ $\mathcal{R}$)
  - $\mathcal{R}$ is an n-dimensional vector, where n is the number of objectives
- MORL methods can be single policy or multi-policy
- Various MORL scenarios exist depending on the availability of the user's utility function
- Utility function represents the user's preferences over objectives
- Two optimality criteria exist: SER and ESR

SER and ESR

- Scalarised Expected Returns (SER) is the most commonly used optimality criterion in MORL
- Utility of a user is derived from multiple executions of a policy
- First the expectation is computed, then utility function is applied

$$V_u^{\pi} = u\left(\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \mid \pi, \mu_0\right]\right)$$

SER and ESR

- Expected Scalarised Returns (ESR) has largely been ignored by the MORL community
- Utility of a user is derived from the single execution of a policy
- Applies the utility function to the return vector first, and then the expected utility is calculated

$$V_u^\pi = \mathbb{E}\left[u\left(\sum_{t=0}^{\infty}\gamma^t \mathbf{r}_t\right) \mid \pi, \mu_0\right]$$

SER and ESR

- For linear utility functions the policies learned under the ESR and the SER criterion are the same
  - Single-objective methods can be used
- For non-linear utility functions the policies learned under the ESR and the SER criterion are different
  - Strictly multi-objective methods *must* be used
- *In the real world utility functions can be non-linear!*

SER and ESR

- Non-linear utility functions do not distribute across the sums of the immediate and future returns

$$\max_{\pi} \mathbb{E}\left[u\left(\mathbf{R}_t^- + \sum_{i=t}^{\infty} \gamma^i \mathbf{r}_i\right) \middle| \pi, s_t\right] \neq u(\mathbf{R}_t^-) + \max_{\pi} \mathbb{E}\left[u\left(\sum_{i=t}^{\infty} \gamma^i \mathbf{r}_i\right) \middle| \pi, s_t\right]$$

- Non-linear utility functions invalidate the Bellman equation
- New methods need to be formulated to handle the ESR criterion and non-linear utility functions

Expected Utility Policy Gradient

- Roijers et al. [4] proposed an Expected Utility Policy Gradient (EUPG) to handle the ESR criterion.
- Roijers et al. showed that to learn optimal policies under the ESR criterion the accrued and future returns must be taken into consideration.
- EUPG calculates the accrued returns:

$$\mathbf{R}_t^- = \sum_0^{t-1} \mathbf{r}_t$$

Expected Utility Policy Gradient

- EUPG calculates the future returns:

$$\mathsf{R}_t^+ = \sum_t^{t_n} \mathsf{r}_t$$

- EUPG applies the utility function to the sum of the accrued and future returns:

$$\mathsf{R}_t = \mathsf{R}_t^- + \mathsf{R}_t^+$$

Bootstrap Thompson Sampling

- We aim to learn a posterior, however, this is not always possible!
- In this case we can approximate a posterior with Bootstrap Thompson Sampling (BTS) [2]
- BTS was proposed in a multi-armed bandit setting
- Each arm, i, has a bootstrap distribution that contains a number of bootstrap replicates, $j \in 1, ..., J$
  - J is a hyper-parameter that needs to be tuned
  - A small J makes BTS greedy while a larger J increases exploration
- Each bootstrap replicate contains two parameters $(\alpha_{ij}, \beta_{ij})$ where $\frac{\alpha_{ij}}{\beta_{ij}}$ is an observation
- To determine an optimal action, for each arm the bootstrap distribution is sampled and the arm with maximum returned observation is executed

Bootstrap Thompson Sampling

- Once an arm has been pulled, the arm, *i*, must be updated with the return, *r*, recieved
- BTS uses random re-weighting
  - For arm, *i*, a Bournoulli bandit is sampled for each bootstrap replicate j
  - If the Bournoulli bandit returns 1 the bootstrap replicate is updated
  - The α and β parameters for arm, *i*, for the bootstrap replicate, *j*, are then updated as follows:

$$\alpha_{ij} = \alpha_{ij} + r$$

$$\beta ij = \beta_{ij} + 1$$

Monte Carlo Tree Search

- Monte Carlo Tree Search (MCTS) uses heuristic tree search to solve decision problems
- Solved problems like Go [6]
- MCTS builds a search tree of nodes, where each node has a child per action
- MCTS has two phases: learning phase and execution phase

Monte Carlo Tree Search

- During the learning phase the agent builds a search tree using the following four steps:
  - Selection
  - Expansion
  - Simulation
  - Backpropagation
- The execution phase selects the child node which the agent traverses to next

Distributional Monte Carlo Tree Search

- Majority of MORL and RL research focuses on learning a policy that maximises the expected returns
- The expected returns (or value) cannot account for the range of positive or adverse outcomes a decision may have
- Making decisions based on the expected returns (or value) the agent does not have sufficient critical information
- This is crucial for MORL under the ESR criterion and for risk-aware reinforcement learning
- *Scenarios where the policy may only be executed once!*

Distributional Monte Carlo Tree Search

- We propose Distributional Monte Carlo Tree Search (DMCTS) that learns a posterior distribution over the returns
- DMCTS can learn optimal policies for non-linear utility functions for risk-aware RL and the ESR criterion
- DMCTS is a single policy algorithm and relies on the utility function of a user to be known

Distributional Monte Carlo Tree Search

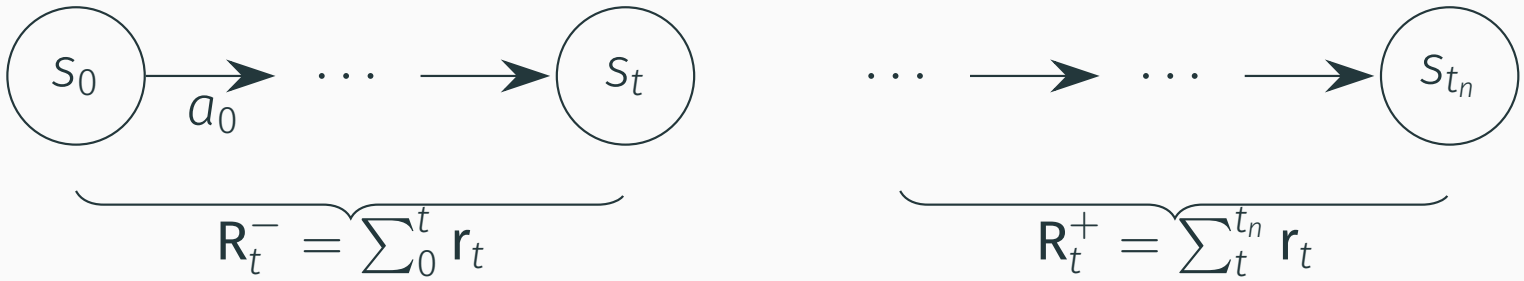
- Like MCTS, DMCTS has a learning phase and an execution phase
  - DMCTS uses selection, expansion, simulation and backpropagation during the learning phase
- DMCTS builds an expectimax tree [9]
  - An expectimax tree uses both chance and decision nodes
  - Each decision node represents a state, action and reward of an MOMDP and has a child chance node per action
  - At each chance node the environment is sampled
  - If an unseen reward is generated, a new child decision node is created

Distributional Monte Carlo Tree Search

- DMCTS maintains a distribution over the utility of the returns at each node in place of the expected value traditionally used by other MORL and RL methods
- Although DMCTS is defined as an algorithm for ESR, DMCTS can also optimise for the SER criterion with a minor change to the algorithm
- DMCTS maintains a posterior distribution over the utility of the returns

Distributional Monte Carlo Tree Search

- To compute the distribution, we first calculate the accrued returns
  - The accrued returns, $\mathbf{R}_t^-$, is the sum of the rewards received when traversing the search tree during the learning phase
- The future returns, $\mathbf{R}_t^+$, are then calculated
  - The future returns are the sum of the rewards received from traversing the search tree during Monte Carlo simulations

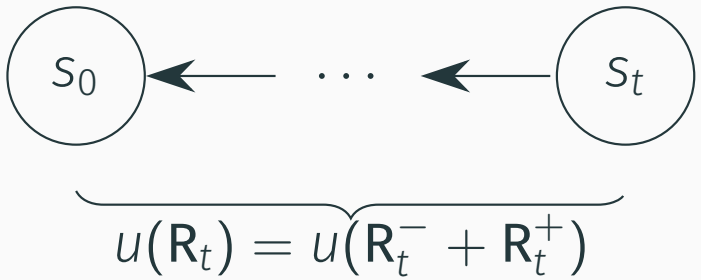


Distributional Monte Carlo Tree Search

- Once the future returns have been calculated and the simulation phase is complete, the cumulative returns, $\mathbf{R}_t$, are calculated

$$\mathbf{R}_t = \mathbf{R}_t^- + \mathbf{R}_t^+$$

- The utility function is then applied to the cumulative returns, $u(\mathbf{R}_t)$
- The utility of the cumulative returns is then backpropagated to each node visited during the learning phase



Distributional Monte Carlo Tree Search

- During the backpropagation phase the distribution at each node is updated
- We aim to maintain a posterior distribution, however this is not always possible
- Instead a bootstrap distribution is maintained at each decision node
- Algorithm 1 shows how the bootstrap distribution at each node is updated for ESR during the backpropagation phase
- On initialisation of a new node, α and β are set to 1.
- *J* is hyperparameter to be tuned, which controls exploration/exploitation
  - For a low *J* value BTS becomes greedy
  - For a high *J* value BTS increases exploration, at a computational cost

DMCTS ESR : Algorithm 1

Input: $i \leftarrow$ *Node in the tree*
Input: $\mathbf{R}_t \leftarrow$ *Cumulative Reward*
$J \leftarrow$ *node.bootstrapDistribution*
for *j, ..., J bootstrap replicates* **do**
  Sample d_j *from Bernoulli(1/2)*
  if $d_j = 1$ **then**
    $\alpha_{ij} = \alpha_{ij} + u(\mathbf{R}_t)$
    $\beta_{ij} = \beta_{ij} + 1$
  end
end

DMCTS ESR : Algorithm 2

- To traverse the search tree during learning, the agent samples the bootstrap distribution
- Algorithm 2 outlines how the agent selects which action to execute during the learning phase

Input: $n \leftarrow$ *Node in the tree*
Require: α, β *prior parameters*
α_{ij} *:=* α, β_{ij} *:=* β *{For each n child, i, and each bootstrap replicate, j }*
for *i, ..., n children* **do**
  Sample j from uniform 1, ..., J bootstrap replicates
  Retrieve α_{ij}, β_{ij}
end
maxChild = $\arg\max_i \frac{\alpha_{ij}}{\beta_{ij}}$
return *maxChild or maxChild.action*

Distributional Monte Carlo Tree Search

- Using Algorithm 1 and Algorithm 2 ensures that we optimise under the ESR criterion. Since the following is true:

$$\frac{\alpha_{ij}}{\beta_{ij}} \equiv \mathbb{E}[u(\mathbf{R}_t^- + \mathbf{R}_t^+)]$$

- Since the following is also true:

$$ESR = \mathbb{E}[u(\mathbf{R}_t^- + \mathbf{R}_t^+)]$$

- At execution time we simply select the overall maximising action by averaging over all the acquired data

Distributional Monte Carlo Tree Search

- To use DMCTS to optimise for the SER criterion minor changes need to be made to the algorithm
- Firstly, we initialise the following:

$$\boldsymbol{\alpha} = [1, ..., 1_o]$$
$$\beta = 1$$

- For each node, i, and each bootrap replicate, j, $\boldsymbol{\alpha}_{ij}$ and β_{ij} are updated as follows for the SER criterion:

$$\boldsymbol{\alpha}_{ij} = \boldsymbol{\alpha}_{ij} + \mathsf{R}_t$$
$$\beta_{ij} = \beta_{ij} + 1$$

- Algorithm 3 shows how to update the bootstrap distribution for the SER criterion

DMCTS SER : Algorithm 3

Input: $i \leftarrow$ *Node in the tree*
Input: $\mathbf{R}_t \leftarrow$ *Cumulative Reward*
$J \leftarrow$ *node.bootstrapDistribution*
for *j, ..., J bootstrap replicates* **do**
  Sample d_j *from Bernoulli(1/2)*
  if $d_j = 1$ **then**
    $\boldsymbol{\alpha}_{ij} = \boldsymbol{\alpha}_{ij} + \mathbf{R}_t$
    $\beta_{ij} = \beta_{ij} + 1$
  end
end

DMCTS SER: Algorithm 4

- At decision time we apply the utility function after the expectation has been computed:
- Algorithm 4 shows how an action is selected at decision time during the learning phase under the SER criterion

Input: $n \leftarrow$ *Node in the tree*
Require: α, β *prior parameters*
α_{ij} *:=* α, β_{ij} *:=* β *{For each n child,* i*, each bts replicate,* j *}*
for $i, \ldots, n$ *children* **do**
  Sample j *from uniform 1, ..,* J *bootstrap replicates*
  Retrieve α_{ij}, β_{ij}
end
maxChild = $\arg\max_i u(\frac{\alpha_{ij}}{\beta_{ij}})$
return *maxChild or maxChild.action*

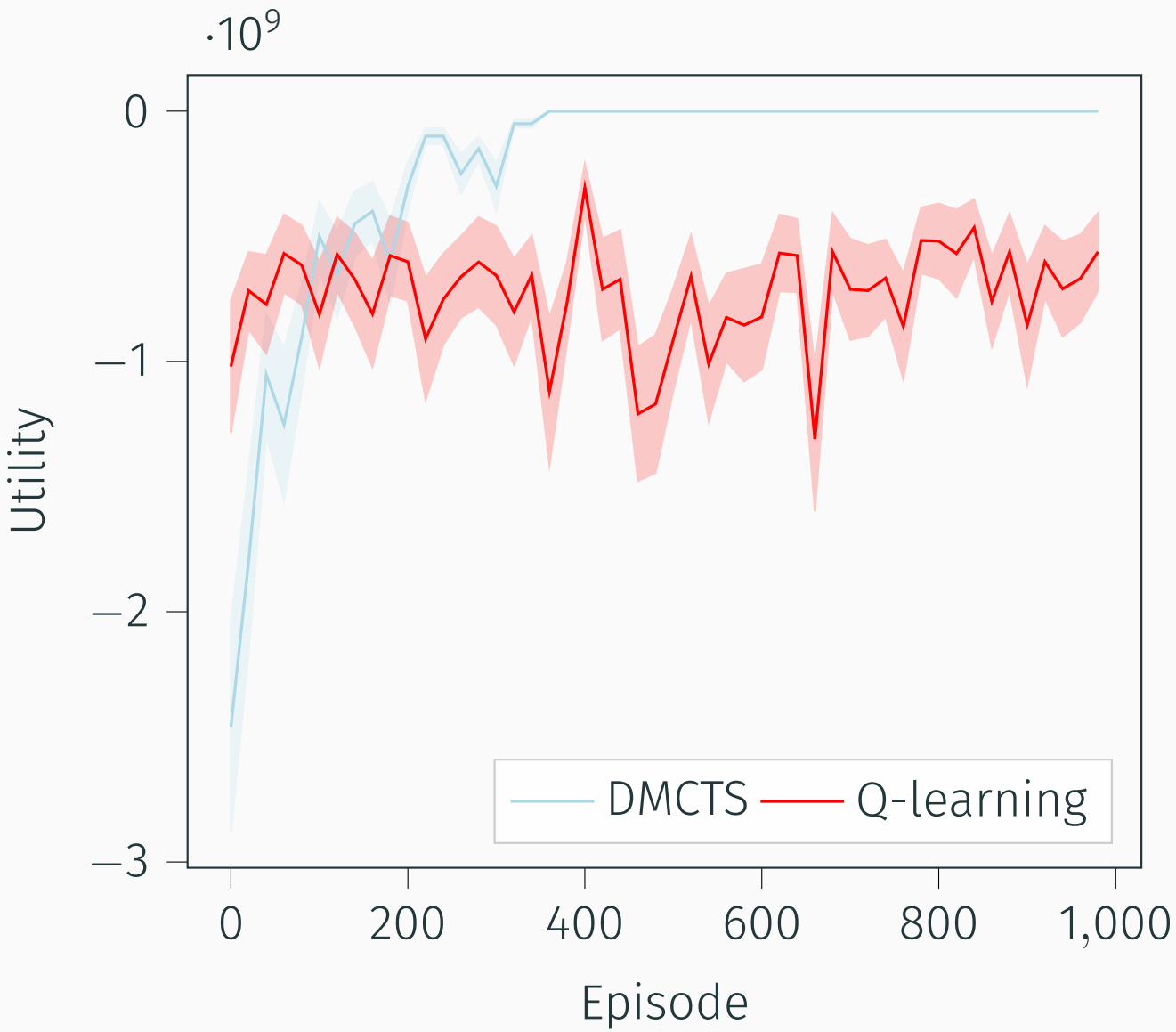
Experiments

- DMCTS is evaluated in the following environments:
  - Risk-aware MDP
  - Dangerous Deep Sea Treasure
  - Renewable Energy Dynamic Economic Dispatch
  - Fishwood
- We average all experiments over 10 runs

Risk-aware MDP

- Risk-aware MDP is a single objective MDP proposed by Shen et al. [5]
- The agent must select from a number of stocks a monetary amount to invest
- The goal is to avoid losing money
- Utility function : $u = 1 - e^{-r_t}$

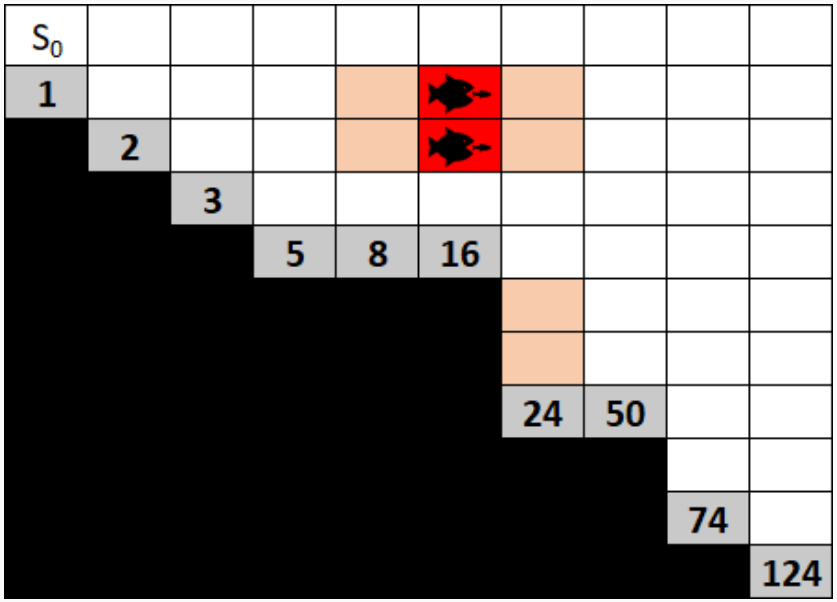
Risk-aware MDP



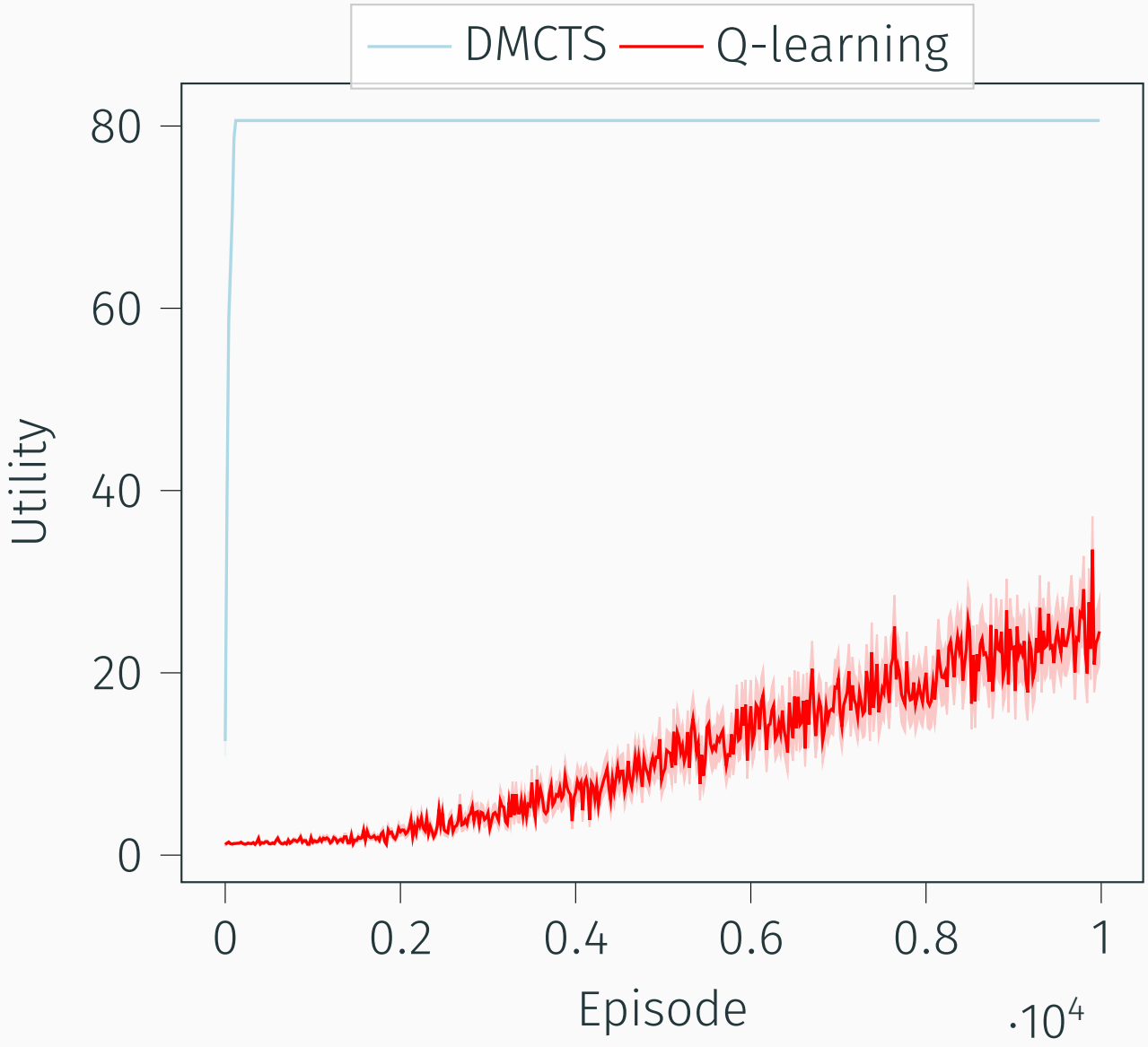
Dangerous Deep Sea Treasure

- Dangerous Deep Sea Treasure (DDST) of MORL environment called Deep Sea Treasure [7]
- Objectives are to maximise treasure, minimise time and minimise danger.
- In DDST there are various states where the agent can be attacked by a shark with a certain probability
- If the agent receives too much damage the submarine crashes and the episode ends
- Utility function : $u(\mathbf{r}) = \arg\max c : \mathbf{r} - c\mathbf{e} > 0$
- The utility function function for DDST is a non-linear utility where $\mathbf{r}$ is a reward vector, $\mathbf{e} = \frac{r_\dagger}{|r_\dagger|}$ where $\mathbf{r}_\dagger$ is a specified target vector, and c is a constant we aim to maximise
- For experimentation we set $\mathbf{r}_\dagger = [54, 0, -14]$

Dangerous Deep Sea Treasure



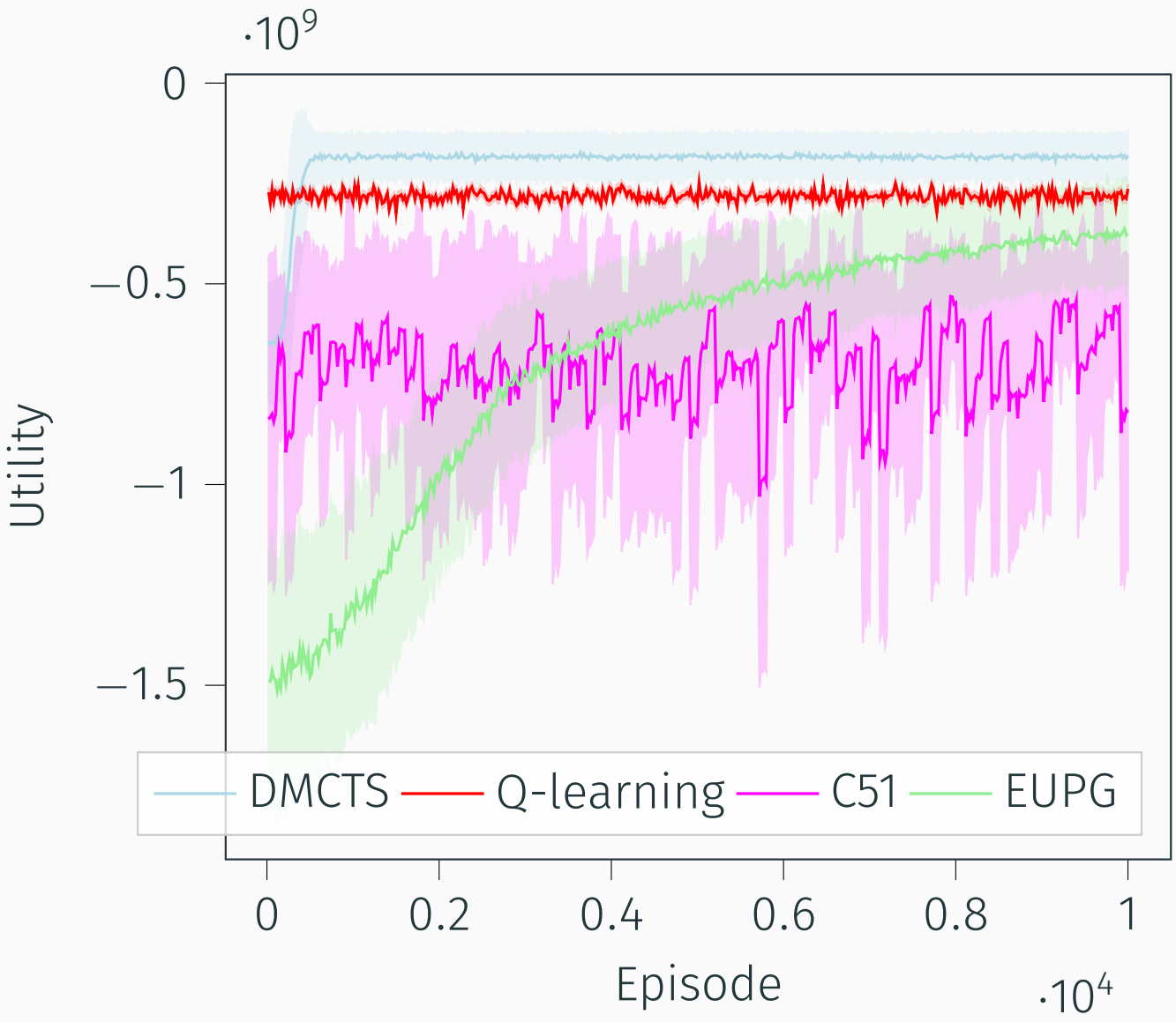
Dangerous Deep Sea Treasure



Renewable Energy Dynamic Economic Dispatch

- Renewable Energy Dynamic Economic Emissions Dispatch (REDEED) is an variation of the DEED environment introduced by Mannion et al. [3]
- Power demand for a town must be met over a 24 hour period
- There are 9 fossil fuel generators and 1 generator powered by renewable energy
- Due to uncertainty in the weather the power output from the renewable energy generator varies
- The agent controls a fossil fuel generator with the aim of learning an optimal policy
- REDEED has three objectives: goal is to minimise cost, emissions and violations for a power plant
- We compare DMCTS against EUPG, C51 [1] and Q-learning
- We optimise under the ESR criterion
- Utility function: $u = -\sum_{o=1}^{O} w_o f_o$ where w_o is the objective weight. We set $w_c = 0.225, w_e = 0.275$ and $w_p = 0.5$

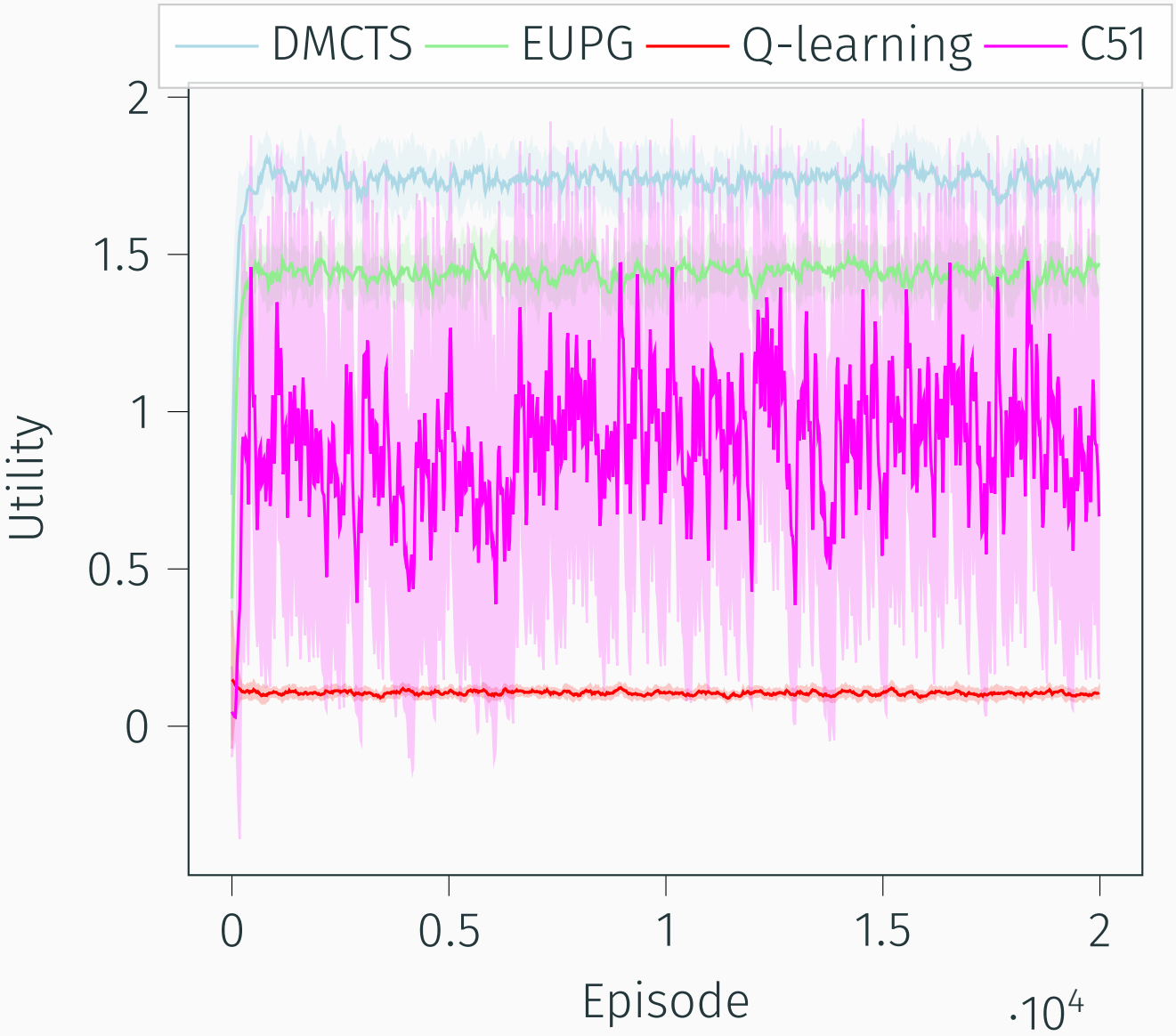
Renewable Energy Dynamic Economic Dispatch



Fishwood

- Fishwood is an ESR benchmark problem proposed by Roijers et al. [4]
- Fishwood has two states: in the words or at the river
- Fishwood has two action: to catch fish (when at the river) and obtain wood (when in the woods)
- At the river the agent has a 0.25 chance of catching a fish
- In the wood the agent has a 0.65 chance of obtaining wood
- For every fish caught, two pieces of wood are required to cook the fish
- Utility function : $u = \min\left(\mathit{fish}, \left\lfloor \frac{\mathit{wood}}{2} \right\rfloor\right)$
- We compare DMCTS against EUPG, C51 [1] and Q-learning [8]

Fishwood


DMCTS
EUPG
Q-learning
C51
Utility
2
1.5
1
0.5
0
0
0.5
1
1.5
2
Episode
·10^4

Conclusion

- DMCTS is able to learn good policies in risk-aware and SER settings
- DMCTS is able to achieve state-of-the-art performance
- To learn optimal policies under the ESR criterion making decision based on the expected returns is not sufficient.
- Instead a distribution over the expected returns must be use to ensure the agent has sufficient critical information at decision time

Future Work

- We aim to extend DMCTS to learn the Pareto front in SER settings.
  - Multi-Objective MCTS [10] is able to learn the Pareto front in problem domains with deterministic rewards. We aim to extend DMCTS to also be able to learn a set of optimal policies in problem domains with stochastic rewards
- We also hope to extend DMCTS to be able to learn an optimal set of solutions for ESR settings

References i

- Bellemare, M.G., Dabney, W., Munos, R.: A distributional perspective on reinforcement learning.
  In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 449–458. JMLR. org (2017)
- Eckles, D., Kaptein, M.: Thompson sampling with the online bootstrap.
  CoRR abs/1410.4009 (2014).
  URL *http://arxiv.org/abs/1410.4009*
- Mannion, P., Mason, K., Devlin, S., Howley, E., Duggan, J.: Multi-objective dynamic dispatch optimisation using multi-agent reinforcement learning.
  International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2016)

References ii

- Roijers, D.M., Steckelmacher, D., Nowé, A.: Multi-objective reinforcement learning for the expected utility of the return.
  In: Proceedings of the Adaptive and Learning Agents workshop at FAIM 2018 (2018)
- Shen, Y., Tobia, M.J., Sommer, T., Obermayer, K.: Risk-sensitive reinforcement learning.
  Neural Computation 26(7), 1298–1328 (2014)
- Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search.
  Nature (2016)

References iii


Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., Dekker, E.: Empirical evaluation methods for multiobjective reinforcement learning algorithms.
Machine Learning (2011)


Van Moffaert, K., Drugan, M.M., Nowé, A.: Scalarized multi-objective reinforcement learning: Novel design techniques.
In: 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), pp. 191–199 (2013)


Veness, J., Ng, K.S., Hutter, M., Uther, W., Silver, D.: A monte-carlo aixi approximation.
J. Artif. Int. Res. 40(1), 95–142 (2011)

References iv


Wang, W., Sebag, M.: Multi-objective Monte-Carlo tree search. pp. 507–522. PMLR, Singapore Management University, Singapore (2012)