Planning under Uncertainty with Weighted State Scenarios
Supplementary Material

Erwin Walraven
Delft University of Technology
Mekelweg 4, 2628 CD
Delft, The Netherlands

Matthijs T. J. Spaan
Delft University of Technology
Mekelweg 4, 2628 CD
Delft, The Netherlands

In this supplement we provide detailed descriptions of the problem domains discussed in the paper, as well as implementation details of our algorithms, including pseudocode. Section 1 describes the smart grids problem domain. The option trading domain is discussed in Section 2. More implementation details regarding the POMCP algorithm are provided in Section 3.

1 SMART GRIDS DOMAIN

In this section we provide additional details regarding our Scenario-POMDP formulation for matching demand and supply in smart grids. In particular, the description of the state variables is much more detailed than the high-level description in the paper. We also explain how the size of the action space is reduced from exponential to constant using a rotating token.

1.1 STATE DESCRIPTION

The state description and corresponding dynamic Bayesian network are shown in Table 1 and Figure 1. In the paper we mentioned that m_s^i denotes the state of task i. To make this more detailed, we factor m_s^i into a tuple $(m_r^i, m_d^i, m_s^i, m_c^i)$. The variable m_r^i encodes the release time of task i. The variable m_d^i represents the number of steps task i can still be postponed, which can be used to encode the deadline. The variable m_y^i represents the number of steps task i still has to run, which can be used to encode the length of the task. The last decision of agent i is represented by m_c^i, which is an auxiliary variable which we need to reduce the size of the action space. This variable can be either run (R) or idle (I). The factored state variables can be used to determine which actions are feasible to execute given the current state, which we discuss below.

1.2 REDUCING ACTION SPACE SIZE

We apply a technique to reduce the size of the action space from exponential to constant. The state variable m_a represents a token, which can be owned by one agent at the same time. If the variable m_a equals i, then agent i owns the token and the action that is executed applies to the task that is being controlled by agent i. The decision that is made is recorded in the state variable m_c^i. The additional variable is needed because the reward also depends on decisions made by other agents. Initially the token is owned by the first agent, and after executing an action the token is forwarded to the second agent, until every agent has made a decision. In general, m_a is always incremented by 1, but when m_a equals n then it is reset to 1, which represents that the token is returned to the first agent. The Scenario-POMDP state variable t is only incremented when agent n makes a decision, because within one real-world timestep n actions are executed. The same technique to reduce the size of the action space is used by Scharpff et al. (2013).

1.3 ACTIONS AND STATE TRANSITIONS

In our model we define two actions: RUN and IDLE, corresponding to the decisions that can be made for each task. If m_a equals j (i.e., agent j owns the token), then action RUN is feasible if $m_y^j > 0$ and $t \geq m_r^j$. Thus, action RUN can be executed if task j has been released and has not been completed yet. For agent j, action RUN decrements m_y^j by 1, sets m_d^j to 0 and sets m_c^j to R.

Similarly, if m_a equals j, then action IDLE is feasible if $m_d^j > 0$. It decrements m_d^j by 1 if $m_d^j > 0$ and sets m_c^j to I. IDLE is also executed if the task has been completed (i.e., m_y^j equals 0).

After each action execution, the token variable m_a is updated such that the next agent receives the token. If m_a equals n, then t is also incremented by 1 to proceed to the next timestep.

1.4 REWARD FUNCTION

In this section we define the reward function, assuming that the current state is defined as follows:

$$s = (m_r^1, m_d^1, m_y^1, m_c^1, \cdots, m_r^n, m_d^n, m_y^n, m_c^n, m_a, x, t).$$

Table 1: Task Scheduling State Variables.

VARIABLE	DESCRIPTION
$m_r^i \in \{1,\dots,T\}$	release time of task i ($i = 1,\dots,n$)
$m_d^i \in \{0,\dots,T\}$	delay steps of task i ($i = 1,\dots,n$)
$m_y^i \in \{0,\dots,T\}$	remaining st. task i ($i = 1,\dots,n$)
$m_c^i \in \{\text{R}, \text{I}\}$	decision of agent i ($i = 1,\dots,n$)
$m_a \in \{1,\dots,n\}$	agent owning the token
$x \in X$	scenario
$t \in \{1,\dots,T\}$	time

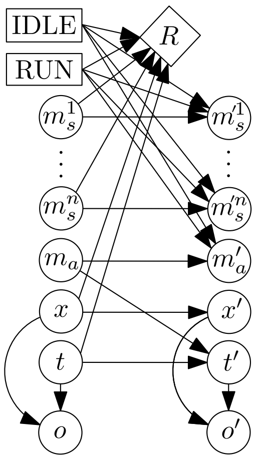

Figure 1: Dynamic Bayesian network of the Scenario-POMDP for task scheduling.

After executing the action IDLE, the reward for an individual task is always 0.

After executing the action RUN, the reward is equal to the cost of running the task, multiplied by -1. The cost is dependent on the decisions made by other agents, because they may have consumed cheap renewable energy already. If agent j owns the token (i.e., m_a equals j) and decides to run, then the decisions $m_c^1,\dots,m_c^{j-1}$ have to be taken into account. The demand of an arbitrary task i, depending on the decision made by the agent, can be computed as follows:

$$p_d(i) = \begin{cases} p_i & \text{if } m_c^i \text{ equals R} \\ 0 & \text{otherwise} \end{cases}$$

Now the total demand of agents $1,\dots,j-1$ can be written as $\sum_{i=1}^{j-1} p_d(i)$, which is the total demand of agents that have already made a decision. The available renewable supply q that is still left for agent j can be defined as follows:

$$q = \max\left(u_t - \sum_{i=1}^{j-1} p_d(i), 0\right)$$

```
input: set of tasks $J$, horizon $T$, scenario set $X$,
       threshold $\rho$
$m_a \leftarrow 1$
foreach $j_i \in J$ do
    define initial task state $m_s^i$
end
for $t = 1,\dots,T$ do
    $o_t \leftarrow$ observed wind speed
    $o_{1,t} \leftarrow (o_1,\dots,o_t)$
    $w \leftarrow$ WEIGHTS$(o_{1,t}, X, \rho)$
    for $i = 1,\dots,n$ do
        $a \leftarrow$ POMCP$(w, t, m_a, m_s^1,\dots,m_s^n)$
        execute action $a$ and update $m_s^i$
        $m_a \leftarrow 1 + (i \mod n)$
    end
end
```

Algorithm 1: Task Scheduling Algorithm.

where u_t is the number of renewable units available, generated by wind. This can be computed using the sigmoid power curve function Z defined in the paper. The final reward of agent j when running task j is:

$$-1 \cdot c(\max(p_j - q, 0))$$

where $c(u)$ is the grid cost function giving the cost of consuming u units from the grid.

1.5 PSEUDOCODE

Algorithm 1 shows the task scheduling algorithm, which is an adaptation of the scenario planning algorithm presented in the paper. The algorithm starts with defining the state for each task in the set of tasks J. For each timestep, it makes a decision for each agent i, and updates the corresponding state m_s^i. After every decision, the token is forwarded to the next agent. For clarity and readability reasons we did not factor m_s^i into a tuple in the description of the algorithm. However, the actual implementation uses factored states of tasks, and the factored state variables are updated according to the description above.

1.6 OTHER ALGORITHMS

In the experiments we compared the performance with the consensus algorithm proposed by Ströhle et al. (2014). The algorithm is shown below in Algorithm 2, where the symbol $\perp$ denotes scheduling nothing and $\mathcal{L}(x|o_1,\dots,o_t)$ represents the likelihood that x predicts the future given observations $o_1,\dots,o_t$. Consensus uses an offline greedy algorithm as subroutine, which is shown in Algorithm 3.

```
input  : partial schedule $S$, set of tasks $J$, observations
         $(o_1, \ldots, o_t)$, scenario set $X$ and current timestep $t$
output: set of tasks $J_t$ starting at time $t$

$J_t = \emptyset$
do
    $f_i \leftarrow 0 \quad (i = 1, \ldots, n)$
    $f_\perp \leftarrow 0$
    foreach $x \in X$ do
        $S_x \leftarrow$ OFFLINEGREEDY$(J, S, x)$
        $p \leftarrow$ set of tasks starting at time $t$ in $S_x$
        if $J_t = p$ then
            $f_\perp \leftarrow f_\perp + \mathcal{L}(x|o_1, \ldots, o_t)$
        else
            foreach $j_i \in J - J_t$ do
                if $j_i$ starts at time $t$ in $p$ then
                    $f_i \leftarrow f_i + \mathcal{L}(x|o_1, \ldots, o_t)$
                end
            end
        end
        $k \leftarrow \arg\max_{i \in \{1,\ldots,n\}} f_i$
        if $j_\perp > f_k$ then
            $j^* \leftarrow \perp$
        else
            $j^* \leftarrow j_k$
            $J_t \leftarrow J_t \cup \{j_k\}$
            add task $j_k$ to $S$ with starting time $t$
        end
    end
while $j^* \neq \perp$;
```

Algorithm 2: Consensus.

2 OPTION TRADING DOMAIN

In this section we give additional details regarding our Scenario-POMDP formulation for trading financial options. We start with a definition of the function B_c to calculate the Black-Scholes value of an option. We also give a detailed description of the state transitions and reward function, as well as pseudocode of the implementation.

2.1 BLACK-SCHOLES EQUATION

If the current stock price is S, then the value of a European call option with strike E can be determined using the function $B_c(S, E, t, r, v)$, where t is the time to expiry in years, r is the annual interest rate and v is the volatility of the market. Both the interest rate r and volatility v are parameters between 0 and 1. The formula to compute the option value, also known as the Black-Scholes value (Black and Scholes, 1973) is as follows:

$$B_c(S, E, t, r, v) = S \cdot N(d_1) - E \cdot e^{-r(H-t)} \cdot N(d_2),$$

```
input  : tasks $J$, partial schedule $S'$ and scenario $x$
output: schedule $S$

sort tasks in $J$ by decreasing length
$S \leftarrow$ empty schedule
for $t = 1, \ldots, T$ do
    $u_t \leftarrow$ units available at time $t$ in scenario $x$
end
deduct renewable units consumed by tasks in $S'$ from $u$
foreach $j_i \in J$ do
    $h_i \leftarrow$ minimum cost starting time of $j_i$ given $u$
    assign starting time $h_i$ to task $j_i$ in schedule $S$
    deduct renewable units consumed by $j_i$ from $u$
end
```

Algorithm 3: OFFLINEGREEDY.

Table 2: Option Trading State Variables.

VARIABLE	DESCRIPTION
$m_o \in \{\mathrm{T}, \mathrm{F}\}$	represents whether agent owns a call
$m_e \in \mathbb{N}$	strike price of the call
$m_t \in \{0, \ldots, 9\}$	time to expiry
$x \in X$	scenario
$t \in \{1, \ldots, T\}$	time

where N represents the cumulative standard normal distribution, and d_1 and d_2 are defined as shown below:

$$d_1 = \frac{\log(S/E) + (r + \frac{1}{2}v^2) \cdot t}{v \cdot \sqrt{t}},$$
$$d_2 = d_1 - v \cdot \sqrt{t}.$$

To prevent division by zero, it is necessary that the time to expiry is strictly greater than zero. The value at expiry can be determined by computing the payoff $\max(S(H) - E, 0)$, where $S(H)$ denotes the stock price at the time of expiry H.

2.2 ACTIONS AND STATE TRANSITIONS

The state description and corresponding dynamic Bayesian network from the paper are shown in Table 2 and Figure 2. Feasibility of actions can be determined using the fully observable state variables m_o, m_e and m_t. The state transitions of the environment are deterministic, so the transition probabilities are either 0 or 1.

The action BUY is feasible if m_o equals F, because then the agent does not own an option. It sets m_o to T, m_e is set to the current stock price, and m_t is set to 9. We assumed that an option expires after 10 days, but after one state transition one day has passed already, so we use the value 9.

The action SELL is feasible if m_o equals T, because then

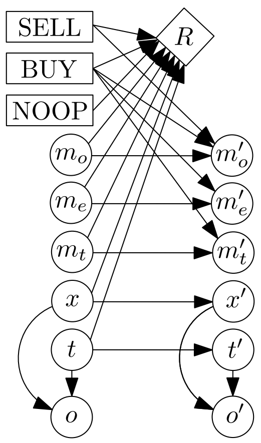

Figure 2: Dynamic Bayesian network of the Scenario-POMDP for option trading.

there is an option to be sold. It sets m_o to F. SELL must be executed if m_o equals T and m_t is 0, because then the option has expired.

The action NOOP is always feasible, except when m_o equals T and m_t equals 0, because then the agent must sell the option.

After any action execution, the time counter t is incremented. The state variable x never changes.

2.3 REWARD FUNCTION

In this section we define the reward function, assuming that the current state is $s = (m_o, m_e, m_t, x, t)$.

After executing the action BUY, the agent receives the reward $-1 \cdot B_c(x_t, x_t, 10/365, r, v)$, where x_t is the stock price at time t in scenario x.

After executing the action SELL, the agent receives the reward $B_c(x_t, m_e, m_t/365, r, v)$, where x_t is the stock price at time t in scenario x.

After executing the action NOOP, the reward is 0.

2.4 PSEUDOCODE

Algorithm 4 below shows the planning algorithm for trading financial options. Compared to the general algorithm in the paper, there are a few modifications. The algorithm defines the observable state using the variables m_o, m_e and m_t. For each timestep $t \geq 4$, it computes weights based on the last three observations, and calls the POMCP algorithm with a time parameter equal to 4. This allows for a rolling horizon implementation, where weights are assigned based on the last three observations, and the scenarios predict the future observations starting from the current timestep. When choosing action BUY or SELL, the observable state variables are modified and the account balance z is updated. The state transitions are deterministic, so we can use the variables m_o, m_e and m_t to keep track of the state of the environment.

```
input  : horizon $T$, scenario set $X$, threshold $\rho$, interest
         rate $r$, volatility $v$
output: account balance $z$

$m_o \leftarrow$ F
$m_e \leftarrow 0$
$m_t \leftarrow 0$
$z \leftarrow 0$
for $t = 1, \ldots, T$ do
    $o_t \leftarrow$ observation from the environment
    if $t \geq 4$ then
        $U \leftarrow (o_{t-3}, o_{t-2}, o_{t-1})$
        $w \leftarrow$ WEIGHTS$(U, X, \rho)$
        $a \leftarrow$ POMCP$(w, 4, m_o, m_e, m_t)$
        if $a = BUY$ then
            $m_o \leftarrow$ T
            $m_e \leftarrow o_t$
            $m_t \leftarrow 9$
            $z \leftarrow z - B_c(o_t, o_t, 10/365, r, v)$
        else if $a = SELL$ then
            $m_o \leftarrow$ F
            $z \leftarrow z + B_c(o_t, m_e, m_t/365, r, v)$
        end
    end
end
```

Algorithm 4: Option Trading Algorithm.

3 POMCP IMPLEMENTATION DETAILS

Our planning algorithm uses POMCP to select actions (Silver and Veness, 2010). The algorithm needs two minor modifications to be able to deal with the factored state representation of the Scenario-POMDP model. A default POMCP implementation samples states from a state probability distribution in the search procedure. However, in our case there are weights associated with scenarios. Therefore, POMCP samples scenarios x from X with a probability proportional to their weight. After sampling a scenario x, a state can be constructed using the factored state variables t and m, which are also given as input. The POMCP algorithm described by Silver and Veness uses UCB as action selection heuristic. In the smart grids domain, we use an ε-greedy heuristic, where the probability to select random actions decreases linearly during the simulations. For each search a new search tree is created, because there is no explicit link between the weights assigned to scenarios in consecutive timesteps.

References

Black, F. and Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. In: *The Journal of Political Economy* 81.3, pp. 637–654.

Scharpff, J., Spaan, M. T. J., Volker, L., and De Weerdt, M. M. (2013). Planning under Uncertainty for Coordinating Infrastructural Maintenance. In: *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, pp. 425–433.

Silver, D. and Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. In: *Advances in Neural Information Processing Systems*, pp. 2164–2172.

Ströhle, P., Gerding, E. H., De Weerdt, M. M., Stein, S., and Robu, V. (2014). Online Mechanism Design for Scheduling Non-Preemptive Jobs under Uncertain Supply and Demand. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 437–444.