Robust Guided Forest Search for Unit Commitment

Vassili Chesterkine
vchester@mit.edu

Paul Theron
paulth@mit.edu

Tom Wright
tomgw@mit.edu

Abstract

Unit Commitment (UC) is a mixed-integer optimization problem (MIP) found in power systems that consists of determining which generators are generating electricity at every point in time. It takes on various forms of complexity, considering each generator's on/off status, power outputs, and often the transmission of power across the grid. It is critical to solve these problems effectively, as they involve satisfying the current electricity demand, while ensuring reliability and efficiency of the network. Exact methods scale poorly for large instances, which is why the linearized, or simplified versions are often considered. Alternatively, heuristics can help expedite the optimization. In this project, we develop a reinforcement learning (RL)-guided tree search framework, across a forest of pruned search trees in order to solve the first-stage binary problem (generator on/off status). This generates robust partial solutions, reducing the MIP to its second stage, which is in this case is, a linear problem (LP) known as Economic Dispatch (ED).

1 Introduction

Our original motivation stemmed from our backgrounds in optimization. Our program combines optimization and machine learning, including thorough studies regarding how each discipline can support one another. The presented approach of solving one stage of this problem with machine learning (ML), and the other with traditional operations research (OR) techniques is a fascinating combination of the two areas of research.

Proper operations of power systems is imperative to the well-being of societies and economies around the globe. Given the $400 billion market in the USA alone, and the often deadly consequences of power outages, any mismanagement of the grid can result in billions of dollars in economic loss, and more importantly, the loss of life. While in a perfect world, we could solve every instance to optimality quickly and without fail, this is not possible. Grid operations at scale are far too complex to be solved exactly at the required frequency. Therefore, developing reliable heuristic methods for power systems is a necessary and highly impactful area of research.

Concretely, UC relates to choosing each generator's on/off status, and consequently power output, over some time horizon. As input, the current status of the grid and various power demand forecasts are given. In a setting with N generators, there are up to 2^N on/off status configurations of grid. Framing the commitment decision as an action in $\{0, 1\}^N$ is therefore much more reasonable than treating it as a classification problem with 2^N classes. That said, the action space is still exponential, and therefore too large for most discrete RL algorithms to be performant when applied directly, creating a very difficult learning environment.

2 Related Work

UC has been a widespread research topic for decades, with a multitude of different approaches being proposed to solve it. UC is fundamentally an optimization problem, often considering uncertainties, and has therefore been solved using traditional MIP strategies, including Benders decomposition to account for stochasticity by minimizing the expectation across demand samples. Alternative methods such as genetic algorithms (Kazarlis et al., 1996), particle swarm optimization (Chakraborty et al., 2012) and simulated annealing (Zhuang and Galiana, 1990) have also been studied.

While the application of RL to power systems tasks is relatively new (Dalal and Mannor, 2015), it has gained traction in recent years. Multiple attempts to leverage RL were based on Q-learning. Yet, the scope of these studies remained quite limited, some only testing on a handful of scenarios (Li et al., 2019) or not taking into account uncertainty (Qin et al., 2021), which raises concerns regarding the generalizability and robustness in a space where worst-case performance is of extreme importancte (Mahmud et al., 2011).

Since vanilla RL models struggle with the exponential scaling of the action space in UC, they are not well fitted to real-world cases, which can include thousands of generators. Another approach that can enable scalability is based on RL-guided tree search, where one selects the optimal actions by considering the total cost incurred at depth H. In fact, even though the tree search associated with UC has an exponential number of nodes with respect to the number of generators, heuristics and RL can provide intelligent strategies to prune the search tree and lead to relevant solutions without extensive exploration. de Mars and O'Sullivan (2021) uses top actions considered by a trained Proximal Policy Optimization (PPO) agent to prune the search tree, removing irrelevant actions from consideration, and then explores the tree exhaustively using Uniform Cost Search (UCS). Subsequently, de Mars and O'Sullivan (2022) uses A* and heuristics to speed up the search, while retaining similar performance.

3 Problem statement

This work seeks to leverage the capabilities of RL in order to solve the crux of power systems problems, namely the integer, or binary, first-stage problem.

3.1 Formulation

We seek to minimize the total cost of generating electricity, summed with the cost of not meeting demand within a predefined region and a given episode. In this study, each episode consists of a full day, and a commitment decision is made every 30 minutes. Each episode is therefore 48 consecutive timesteps. To do so, one must decide which power plants are going to be active at a given timestep, considering a forecast of demand and other inputs. Given N different power plants in a region, an action at time step t is a binary vector $y_t \in \{0, 1\}^N$, where each $y_{i,t}$ represents the on/off status of the i-th plant. This action is chosen in consideration of the current observation. Each observation consists of:

1. Generator up/down times: $u_{i,t} \in \mathbb{Z}$
2. Demand forecasts and corresponding errors [1]: $d_t,\ \epsilon_t^d \in \mathbb{R}$
3. Renewable energy generation forecasts: $w_t,\ \epsilon_t^w \in \mathbb{R}$
4. Current timestep: $t \in \mathbb{N}$

In addition to the data that is input to (an observation for) the RL model, we have generator data which is used to solve the subsequent optimization problem. For each generator i, we have:

1. Minimum and maximum operating limits [2]: $p_i^{min},\ p_i^{max}$
2. Minimum operating down/up times [3]: $t_i^{min\_down},\ t_i^{min\_up}$
3. Quadratic cost curve coefficients: $a_i,\ b_i,\ c_i$
4. Startup cost: c_i^s

These parameters will be used below to define the different costs that are incurred by starting and operating each power plant. Subsequently, the key decisions to be made are, for each generator i and timestep t:

[1]Demand, as well as wind, forecasts and errors are sampled from an Autoregressive Moving Average (ARMA) model with normally distributed noise

[2]The power output of each generator is constrained

[3]Once the process to turn on/off a generator has been initiated, it cannot be interrupted

1. Binary on/off status of the generator: $y_{i,t}$
2. Real-valued power output of the generator $p_{i,t}$ in MW (note: $y_{i,t} = 0 \implies p_{i,t} = 0$)

Given $y_{i,t}\ \forall i \in [N],\ t \in [T]$, the real-valued power outputs $p_{i,t}$ are decided upon by solving ED, which aims to choose how much plants that are committed should generate, constrained by their minimum and maximum operating limits. ED is automatically performed using lambda iteration, an optimization method based on the marginal cost of additional power generation.

In this setting, we aim to find an RL policy π that maps the state space $\mathcal{S}$ to the action space $\mathcal{A} = \{0,1\}^n$, in order to minimize the total costs associated with power generation, which are detailed in the section below.

3.2 Objective

We seek to minimize the costs associated with operating an instance of UC problem, considering unmet demand, over an episode. The formulation below models the costs incurred when generating power. To this end, we treat our reward function as the negative cost, in order for our agent to learn a policy that maximizes reward, as is done in most RL settings.

Operating Costs

For each episode, total operating costs C_t are calculated as the sum of fuel costs C^f, startup costs C^s and lost load costs C^l, summed over all time steps t, all of which are broken down in the sections below:

$$C = \sum_{t=1}^{T} C_t, \text{ where } C_t = C_t^f + C_t^s + C_t^l$$

These costs reflect the actual economic costs that are at play in power generation. Additionally, similar costs were implemented in the Python package we used, which enabled direct comparison of results.

Fuel costs

Fuel costs are non-linear costs individually generated by each generator of the grid, and are calculated according to the quadratic cost curves defined here:

$$C_t^f = \sum_{i=1}^{N} C_{i,t}^f, \text{ with } C_{i,t}^f = \frac{1}{2} y_{i,t}(a_i p_{i,t}^2 + b_i p_{i,t} + c_i)$$

where $y_{i,t} \in \{0,1\}$ is the generator commitment – on/off, and $p_{i,t} \in \mathbb{R}^+$ is the real-valued power output in MW.

Startup Costs

Startup costs are incurred whenever a generator is committed. With c_i^s the startup cost of each generator, $\lambda_{i,t} = \begin{cases} 1 & \text{if } y_{i,t} - y_{i,t-1} = 1 \\ 0 & \text{otherwise} \end{cases}$, startup costs for generator t are:

$$C_t^s = \sum_{i=1}^{n} \lambda_{i,t} c_i^s$$

Lost load costs

When committed generators are unable to meet demand, a lost load cost is incurred:

$$C_t^l = V \left| \bar{d}_t^{net} - \sum_{i=1}^{n} p_{i,t} \right| \tag{1}$$

where V ($/MWh) is the value of lost load. Note that in this setup there is an equal penalty for over-commitment and under-commitment of generation; in practice, the costs of over-commitment are likely to be substantially lower, and could be managed by wind shedding or other means apart from load shedding. As a result, the optimal commitments and reserve allocation strategies in this setup are likely to give greater priority to generator footroom (under-commitment) than would be common in real power systems.

The formulation above naturally induces a MIP, which can be solved using a variety of strategies including branch and bound, decomposition methods or cutting planes. Here, instead of using traditional OR methods, we leverage RL methods to intelligently assign on/off statuses based on observations of the environment. We consider the optimality gap RL methods are able to achieve relative to the true optimal solutions. We further compare solution quality with preexisting methods from literature, both in terms of optimality gap and run time.

4 Method

We will be using `rl4uc`, the Python environment proposed by de Mars and O'Sullivan (2021) which implements the modeling introduced above, and allows for the testing of different RL strategies. The proposed environment is similar to OpenAI Gym API environments, since given an instance of the environment `env` and an action `action`, we can call `env.step(action)` to execute the action in the current state and return a new state, reward and indicator regarding whether the state is terminal (end of the simulation) or not.

Our first contribution will be to implement a multitude of different agents to evaluate their performance as well as measure their corresponding run times. We implemented several baseline models that provide an idea of what performance we can later expect from RL models. These baselines include an agent which always chooses randomly what actions to use, as well as a classical ϵ-greedy multi-armed bandit (MAB). Later on, we programmed Soft Actor-Critic (SAC), Asynchronous Advantage Actor-Critic (A3C), and Proximal Policy Approximation (PPO) agents. Given a state, these agents all output a distribution over the action space, which will be useful for subsequent parts of our strategy.

Due to the exponential growth of the action space with the number of generators, the direct application of RL strategies to UC problems is not suitable. One method that has been developed in order to tackle this problem, is the application of local tree search methods on trained policies. de Mars and O'Sullivan (2021) proposed formulating the Markov decision process (MDP) in the following representation.

Nodes represent observations, with edges being feasible actions. Rooted at state s_0, for each node (state), only the M most likely actions, according to the trained policy π, are considered as possible actions. A probability threshold ρ is used to select the most likely actions. This effectively upper bounds the branching factor of the search tree by $\lfloor \frac{1}{\rho} \rfloor$, as compared to the initial 2^N. This ensures that the complexity of the search tree is not affected by N, and therefore allows the method to scale.

The cost of traversing an edge is the expected operating cost, estimated by a Monte Carlo method, simulating each transition $|S|$ times and calculating the mean, where S is a set of uncertainty scenarios. Their proposed guided tree search algorithm selects actions iteratively, greedily selecting the action at time t that results in the minimum possible costs at time $t + H$, where H is our horizon.

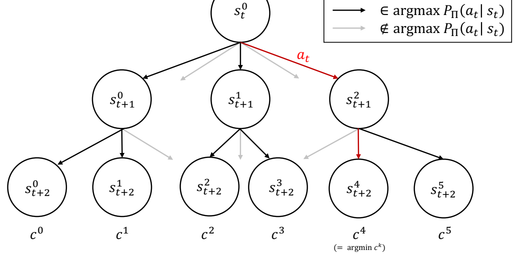

Figure 1: Guided tree search – sub-optimal actions, according to π, are pruned and the remaining scenarios are exhaustively explored up to horizon H (here $H = 2$) to find cost-saving actions

It is notable that this vastly limits the number of actions, and therefore states, explored. We only consider probable actions from the policy initialized on the unperturbed initial state s_0, and then try to minimize an expected cost under uncertainty. This assumes that the policy is stable to varying initial conditions.

That said, through experimentation, we observed that a policy can be, in fact, quite sensitive to initial conditions and state. Across uncertainty scenarios in S – obtained by sampling forecasts and errors with the ARMA model mentioned above, we found that by building trees rooted at each perturbed initial state, we explore at least 7% more of the action space. More precisely, our experiments demonstrated that a perturbation of 5% on the demand and wind scenarios, resulted in 10% more exploration of the action space, for the 5 generators scenario. The number of new actions we explore following a disturbance increases with the number of generators. Moreover, research has shown a gap in performance between unguided tree search – without pruning – and guided tree search, leveraging the decisions of an RL agent. This gap suggests that with a more robust agent, or a better pruning method, there is room for improvement.

Therefore, we propose a new search algorithm, which firstly builds $|S|$ trees, each initialized on an uncertainty scenario $s_0 + \delta_s$, selects the best action within each tree, and finally decides on a final action path via one of two path-finding mechanisms, call it f:

- (1-α)% best worst performance (max-min): Remove the $\frac{\alpha}{2}$ outliers at each tail, and consider the set of actions that **minimize worst case costs across scenarios**.
- (1-α)% best average performance (max-avg): Remove the $\frac{\alpha}{2}$ outliers at each tail, and consider the set of actions that **minimize average costs across scenarios**.

The intuition is that these methods should improve the robustness of the algorithm, which is an important factor to consider when dealing with power systems. That is to say, when supply or demand forecasts are poor, or high in variance, we expect improved performance. We describe the method in detail in the following section.

4.1 Robust Guided Forest Search algorithm

A forest $\mathcal{F}$, is constructed given a trained a policy π using a given RL algorithm of choice (PPO, A3C, etc...). Each tree τ_s corresponds to a perturbation of the initial state for $\delta_s \in S$ and therefore has $s_0 + \delta$ as its root node. From this perturbed state, actions suggested by $\pi(s_0 + \delta_s)$ and with probability greater than a selected threshold ρ are selected as branches from the root node. Subsequently, the tree is similarly expanded to depth H, which we chose to be 4 in our experiments. For each leaf of the tree, a cost is given, summing up all the costs incurred by the actions that led to this terminal state. At this point, the tree search – here performed with Uniform Cost Search (UCS) – retrieves the path that led to the minimal cost and the first action along that path is retrieved. The process is then repeated for each time step t of the episode $\{1, ..., T\}$. Finally, we decide the chosen action with one of the path-finding strategies described in the section above. The algorithm below summarizes this methodology.

The procedure described above selects the best action given a state, based on all $|S|$ uncertainty scenarios explored.

5 Results

As discussed above, while it remains quite easy for RL methods to attain reasonable performance on a limited number of generators, it takes considerable time and computational power to train strong agents for larger instances. For this project, we restricted ourselves to environments with 5 and 10 generators. Assuming that we are able to train (offline) an RL agent that eliminates irrelevant actions and identifies cost-saving actions broadly, our proposed method scales independently of the number of generators, since the branching factor of the search tree upper bounded by a constant.

After training several RL agents, the next step we conducted was to use these agents as guides for a tree search of the best commitment. The results we obtained are presented in table 1 below. [4]

[4]For the sake of clarity, we excluded the MAB agent from the results table, since it had performance comparable to that of the random agent.

Algorithm 1 Robust Guided Forest Search

```
Require: $\pi$ a trained policy and $s_0$ an initial state
  $W_a \leftarrow +\infty,\ a \in \mathcal{A}$
  for $\delta \in \Delta$ do
    $s_\delta \leftarrow s_0 + \delta$
    c($s_\delta$) $\leftarrow 0$
    Initialize tree $\tau_\delta$ with root node $s_\delta$
    for $h \in 0, ..., H-1$ do
      for each node $s$ of $\tau_\delta$ at depth $h$ do
        $a_s^1, \dots a_s^k \leftarrow$ top $k$ actions with regard to $\pi(s)$
        $s_1', \dots s_k' \leftarrow$ successors of $s$ given actions $a_s^1, \dots a_s^k$ respectively
        $r_1, \dots, r_k \leftarrow$ corresponding rewards
        Expand $\tau_\delta$ with nodes $s_1', \dots s_k'$
        $c(s_j') \leftarrow c(s) + r_j,\ j \in 1, \dots, k$
      end for
    end for
    Get first action, total cost $a_0,\ c_\delta^* \leftarrow \arg\min_{s_f \in S_H} c(s_f)$ where $S_H$ set of nodes with depth H
    $W_{a_0} \leftarrow \min\{W_{a_0}, c_\delta^*\}$
  end for
  return $\arg\max_a W_a$
```

Num. generators	Modality	Agent	Reward (mean, \$)	Reward (std, \$)	Optimality gap (mean, %)	Runtime (mean, s)
5	Unguided	–	–234,537	39,348	7.21	78.6
	Guided	Random	–300,295	253,732	37.3	53.2
		SAC	–285,376	39,829	21.1	12.9
		A3C	–263,429	39,077	20.9	**5.34**
		PPO	**–230,650**	**38,618**	**5.51**	36.8
10	Unguided	–	–	–	–	–
	Guided	Random	3,799,066	2,086,457	762	72.2
		SAC	550,767	90,934	20.9	9.0
		A3C	530,964	79,556	21.0	**5.68**
		PPO	**514,979**	**105,403**	**16.9**	12.5

Table 1: Guided tree search performance of various agents – guided search leads to reasonable performance and runtime

We observe that tree search methods yield rewards only marginally worse than the optimal rewards, as the optimality gap column suggests. Overall, A3C and PPO seem to enable greater rewards. However, because our environment is not sample efficient, the training of SAC was shorter than that of A3C and PPO. In a real-world case, RL agents would only have to be trained once, and therefore performance should me the main selection criterion of the agent. As expected, tree search guided by a random agent is significantly worse, leading to both poor and unreliable performance as well as lengthy exploration.

Interestingly, the standard deviation of rewards indicates that guided tree search leads to reasonably stable performance, similar to unguided search. However, guided approaches offer better performance as well as shorter runtimes than unguided ones. Remarkably, unguided tree search did not converge in a reasonable time for 10 generators, which supports the need for guided exploration of the search tree. Although this should be verified by testing with larger instances, these findings support our claim that while unguided tree search is intractable when the number of generators increases (and is in fact impossible for real-life cases with thousands of generators), guided tree search does scale.

We then carry out extensive experiments on our proposed Robust Guided Forest Search method. We test both the A3C and PPO agents, varying the size of the forest, as well as the path-finding strategy that retrieves the final action and the type of uncertainty applied on the initial states – $\pm 10\%$

uncertainty on the nominal forecast values (Auto) or sampling from the normal distribution of noise of the ARMA model (Box).

Num. gen.	Agent	Num. trees	Strategy	Observation uncertainty	Reward (mean, \$)	Reward (std, \$)	Runtime (mean, s)	Opt. Gap (%)
			max-avg	Box	−265,007	38,756	8	22
		10		Auto	−263,665	38,862	8	21
			max-min	Box	−265,007	38,756	8	22
				Auto	−263,665	38,862	8	21
	A3C		max-avg	Auto	−263,420	39,341	27	21
		50	max-min	Box	−265,046	38,549	26	22
				Auto	−263,420	39,341	27	21
			max-avg	Auto	−264,097	38,730	47	21
5		100	max-min	Box	−265,055	38,520	46	22
				Auto	−264,097	38,730	49	21
			max-avg	Auto	−234,203	40,778	32	7
		10	max-min	Box	−251,143	71,291	34	14
				Auto	−234,203	40,778	33	7
			max-avg	Auto	**−230,055**	39,131	155	**5**
	PPO	50	max-min	Box	−246,136	59,009	182	12
				Auto	−230,231	39,308	149	**5**
			max-min	Box	−348,162	61,498	327	50
		100		Auto	−236,945	54,230	332	8
			max-min	Box	−533,973	78,936	12	22
		10		Auto	−531,440	79,131	12	**21**
10	A3C		max-min	Box	−534,017	78,540	43	22
		50		Auto	**−530,930**	80,121	43	**21**
			max-min	Box	−534,026	78,493	82	22
		100		Auto	−532,210	78,948	81	**21**

Table 2: Robust Guided Forest Search with various agents and settings

The main takeaway from these experiments is that the forest search enables more robustness, particularly with larger instances, even though that remains to be proven with settings of more generators. With 10 generators, our best model is able to reach a similar average reward as the tree search, with a considerably smaller standard deviation. This suggests that our proposed method is more resilient to erroneous forecasts and other sources of uncertainty. This is further supported by the worst case performance analysis below. Table 3 lists the minimal rewards incurred for any testing episode, using the RL agent's action prediction directly, on the tree search or on the forest search. We notice that guided forest search enables a slightly better worst case scenario.

Num. generators	Agent	Raw agent	Tree search	Forest search
5	PPO	−725,810	−308,640	−299,774
	A3C	-	−334,631	−326,891
10	PPO	−4,889,781	−779,327	-
	A3C	-	−677,227	−649,537

Table 3: Worse case performance analysis – forest search is more robust

6 Discussion

6.1 Challenges

There are several bottlenecks that certainly constrain the performance of our solutions. First, as discussed previously, the main challenge posed by the UC problem arises when the number of generators increases, and the size of the action space exponentially with it. While our results and intuition suggest that Robust Guided Forest Search can be a robust method that finds cost-saving solutions to the UC problem in reasonable time, further experiments should be performed to validate this. Due to the computational complexity of the problem and our the limited time frame of the project, we were not able to conduct these.

Then, while the implementation of the Python environment must have involved a considerable effort, there remain many possible improvements that would facilitate further development of RL solutions to the UC problem. For instance, the environment does not directly inherit from the traditional OpenAI `gym.Env` class, which causes compatibility issues with other RL libraries. Furthermore, the package operates with a NumPy backend, which slows down the training of all RL models, which relies on PyTorch. While it would be possible to adapt the package to use a PyTorch backend, this would require substantial work and tests, which were not possible to include within the scope of this project. Ulterior progress could therefore be made by upgrading the `rl4uc` package to work with PyTorch.

Finally, more complex preprocessing could be performed more efficiently on the states. The package proposes by default a simplistic preprocessing which only concatenates the current status of generators and timestep, as well as an other preprocessing that also concatenates the demand, and wind forecasts. More advanced versions would ideally include for example transformations that consider the constraints of the problem, as well as a more realistic reward design, which asymmetrically penalizes under- and over-commitment.

6.2 Future work

Still, the results obtained in this project are encouraging and support that RL-based methods can tackle critical real-world problems such as UC. In addition to the further development of software environments to enable and encourage further research in this impactful setting, there are plenty of algorithmic and modeling directions that merit further research and development. Tree search strategies have shown that they can greatly improve the capabilities of RL algorithms in this space. Capabilities of tree search and forest search should be thoroughly tested on larger instances to verify scalability and performance. With these experiments, consideration of different path finding mechanisms should be studied for forest search methods.

7 Conclusion

In conclusion, the Unit Commitment (UC) problem is a critical aspect of power systems optimization that involves determining the generators that will produce electricity at any given time. The problem is complex and involves an exponentially large action space, which has made it difficult to solve at scale using traditional optimization techniques. Ensuring reliability and efficiency of the network by improving worst case performance is critical in meeting demand and mitigating disasters. To this end, a reinforcement learning (RL)-guided tree (forest) search framework is proposed, which generates robust partial solutions and reduces the problem to a linear problem (LP) known as Economic Dispatch (ED). In particular, our method is able to scale, as the complexity of the search does not directly depend on the size of the instance. While it does require an RL agent trained offline, this agent does not need to obtain high performance but simply should be able to exclude poor actions and identify interesting ones. This is quite important, since RL agents have not been able to directly yield good performance. Finally, because our model takes into account the stochasticity of forecasts, it is more robust against the inherent variability of the UC problem. Given the importance of power systems to the economy and society, developing reliable heuristic methods for power systems is a necessary and highly impactful area of research. This work supports our hypothesis that RL can be at the forefront of these developments now, and in the future.

References

Chakraborty, S., Ito, T., Senjyu, T., and Saber, A. Y. (2012). Unit commitment strategy of thermal generators by using advanced fuzzy controlled binary particle swarm optimization algorithm. *International Journal of Electrical Power & Energy Systems*, 43(1):1072–1080.

Dalal, G. and Mannor, S. (2015). Reinforcement learning for the unit commitment problem. In *2015 IEEE eindhoven powertech*, pages 1–6. IEEE.

de Mars, P. and O'Sullivan, A. (2021). Applying reinforcement learning and tree search to the unit commitment problem. *Applied Energy*, 302:117519.

de Mars, P. and O'Sullivan, A. (2022). Reinforcement learning and a* search for the unit commitment problem. *Energy and AI*, 9:100179.

Kazarlis, S. A., Bakirtzis, A., and Petridis, V. (1996). A genetic algorithm solution to the unit commitment problem. *IEEE transactions on power systems*, 11(1):83–92.

Li, F., Qin, J., and Zheng, W. X. (2019). Distributed q-learning-based online optimization algorithm for unit commitment and dispatch in smart grid. *IEEE transactions on cybernetics*, 50(9):4146–4156.

Mahmud, M. A., Hossain, M., and Pota, H. (2011). Worst case scenario for large distribution networks with distributed generation. pages 1 – 7.

Qin, J., Yu, N., and Gao, Y. (2021). Solving unit commitment problems with multi-step deep reinforcement learning. In *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 140–145. IEEE.

Zhuang, F. and Galiana, F. (1990). Unit commitment by simulated annealing. *IEEE Transactions on Power Systems*, 5(1):311–318.

Individual contributions

While all members of the group participated on all, at least at the ideation level, each of us focused more particularly on a specific part of the project. Tom worked on the formulation of the optimization problem that UC poses and on methods to solve it. He also provided baselines which we compared our models to. Vassili focused on implementing various RL agents that were used to conduct search tree pruning in our methodology. Because the `rl4uc` package is not fully compatible with open-source libraries, these agents had to be coded manually. Finally, Paul wrote the code to perform forest search, which required programming trees for the purpose of our tree search. He also wrote and ran most of the scripts that led to the results we presented above. All members contributed to writing this report and preparing the final presentation.