Exploring Optimal Layouts of Wind Farms with Multi-Objective Monte-Carlo Tree Search

5519: Decision Making Under Uncertainty - Final Project

Michael Klonowski

April 2022

Contents

1 Introduction

Monte-Carlo tree search (MCTS) is a powerful machine learning (ML) algorithm that has found great success in problems across a multitude of domains. The structure of a tree search combined with the randomness of Monte Carlo simulations allows MCTS to effectively traverse large and often intractable state spaces to create an optimal policy at the root node. MCTS has gained immense notoriety in part due to its implementation in *AlphaGo* (Silver et al. 2016), the first algorithm to ever beat a human in the game of Go, a feat considered years ahead of its time and has

While MCTS has become famous for use in exploring state spaces of games, it has also been successfully applied to exploring the design spaces of architectural problems. There have been numerous recent studies applying MCTS to structural design problems (Rossi, Winands, and Butenweg 2021), mobile network planning (Shen and S. Wang 2021), interplanetary trajectory planning (Hennes and Izzo 2015), streaming processing systems design (X. Huang, Shao, and Yang 2020), and many others.

While these implementations generally deal with optimizing a single objective function (or a linear combination of multiple), often a designer has to deal with multiple competing objectives and present a set of solutions that considers the interaction of these objectives. Multi-objective optimization (MOO) is the field that attempts to deal with these issues. Generally, for MOO problems, one wishes to obtain the Pareto-optimal set of solutions which can in turn be used as a tool in designing the desired architecture. The problem with MOO problems is that they are generally intractable, requiring complex algorithms to generate sufficient solution approximations. In this project, we expand on the research of (Bai et al. 2022) and use MCTS to to explore the multi-objective design space of a wind farm dealing with the competing objectives of cost and annual energy production (AEP).

2 Monte-Carlo Tree Search

The steps of Monte-Carlo tree search and its variations have been thouroughly covered throughout the literature (see Browne et al. 2012 and Świechowski et al. 2021), so only a brief description is provided here. Basic knowledge of Markov Decision Processes, the input to a MCTS, is assumed throughout the course of these descriptions.

2.1 MCTS Steps

Pure MCTS comes in four distinct steps: selection, expansion, simulation, and backpropagation. These can be seen in Figure 1 from Świechowski et al. 2021 and are described below:

Selection Starting from the root node, the algorithm traverses the tree using some exploration vs. exploitation policy until it reaches a node that has not yet been expanded (a leaf node).

Expansion Unless the leaf node is a terminal state, from the leaf node a single (or multiple) child nodes are created from a chosen action (uniformly or heuristically).

Simulation From the new child node, a Monte-Carlo simulation is run using some rollout policy until a terminal state or some other condition is reached.

Backpropagation The resultant reward of this simulation is used to update the statistics of all the nodes visited during the tree traversal, including the node's estimated value and visit counts.

These four steps are then repeated until some stopping time, a maximum number of iterations, or some quality of action from the root node is determined.

2.2 MO-MCTS

When dealing at a multi-objective MDP, the biggest issue is how to handle multi-dimensional reward vectors. The method used in this project is that proposed by W. Wang and Sebag 2012, that is to use a multi-objective indicator known as the hyper-volume indicator (HV). The hyper-volume indicator is essentially a measure of the hyper-volume of a collection of points with respect to some reference point defining the minimum value of all points. In MOO problems, it is desired to maximize the HVI of a set of pareto-dominant solutions. Pareto-dominance is defined as follows from W. Wang and Sebag 2012:

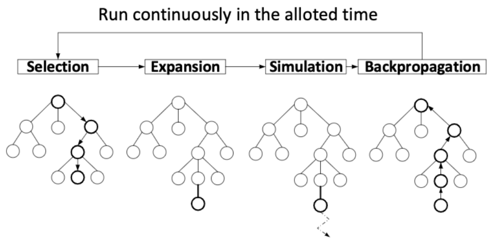


Figure 1: Steps of MCTS (from Świechowski et al. 2021)

Definition 2.1 (Pareto-dominance) *Given two solutions x_a and x_b to a MOO problem with respective vectorial rewards $r_a = (a_1, a_2, ..., a_d)$ and $r_b = (b_1, b_2, ..., b_d)$, r_a is said to Pareto-dominate r_b iff $a_i \geq b_i$ for $i = 1, ..., d$. Given a set of vectorial rewards A, the set P_A defined as the set of Pareto-optima in A is the set of all non-dominated points in A.*

By definition of the hyper-volume indicator, a point that increases the hyper-volume indicator of the P_A by definition is part of the pareto set.

2.3 MO Selection

Typically in MCTS algorithms, the Upper Confidence Bound Applied to Trees (UCT) is the method by which exploration of the tree is weighted against exploitation in the selection phase. In UCT, a child node is selected through action selection to maximize the following quantity:

$$Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \tag{1}$$

where $Q(s,a)$ is the estimated action value of action a at state s, $N(s)$ is the number of times s has been visited, $N(s,a)$ is the number of times a has been selected at s, and C is a constant, usually set to $\sqrt{2}$ for rewards from 0 to 1. In this equation, the first term corresponds to exploitation and the second to exploration. In MO-MCTS, this method is modified to include information from the current pareto set. Since the goal of MOO is to find a set of points that maximizes the hyper-volume indicator of the pareto set, $HV(P)$, the modified MO-UCT is formulated to select the child node whose action value has the possibility to increase $HV(P)$ by the most. We define the modified UCT as follows:

$$HV(Q(s,a) \cup P) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \tag{2}$$

with $Q(s,a)$ now as the multi-dimensional action value of a at s. If $Q(s,a) = Q_{dom}$ is dominated in P, a penalty is added to this equation as the distance from Q_{dom} to the linear piece-wise envelope of the pareto front, since $HV(Q_{dom} \cup P) = HV(P)$. This is to say that the hyper-volume indicator is unaffected by the inclusion of a dominated point.

Clearly, the output of the modified UCT is dependent on the current pareto set, and so the pareto set must be updated with some sort of frequency throughout the MCTS.

3 MCTS for Architecture Design

An architecture design problem formulated as an MDP is distinctly different than a game or sequential process formulated as an MDP. For most design problems the sequence of modifications to the state are irrelevant, and thus what matters is that as many "good" parts of the tree are explored as possible given a time or computational constraint. Furthermore, design problems are typically defined within a continuous state and action space. For MCTS to operate on this type of problem, the spaces must be discretized to some degree, resulting in sparse optimal solutions. A popular method to deal with large action spaces is to dynamically adjust the allowed number of child nodes that can be expanded as a function of parent node visits. This method is called progressive widening, and allows the MCTS algorithm to traverse deeper into the tree. In this project, progressive widening is not used. Rather,

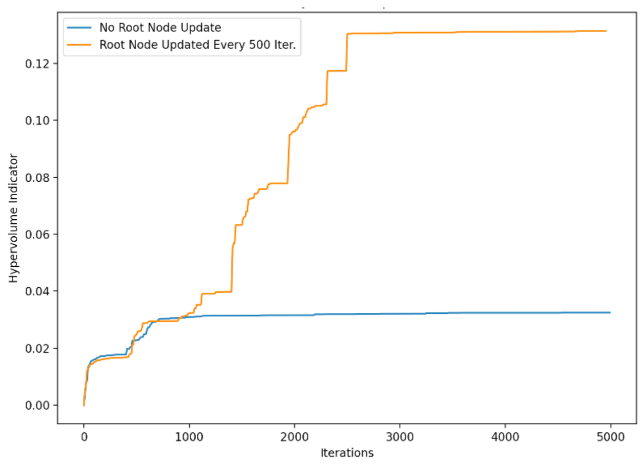


Figure 2: Hyper-volume indicator as a function of MCTS iterations.

to increase the likelihood that the algorithm reaches many different parts of the tree and that it does not waste time around previously found pareto optimal solutions a new root node is sampled from the most recent pareto set and the tree search is reinitialized from that point. Tests have shown that this method allows the algorithm to reach new areas of the tree rather than becoming stuck near local solutions. Figure 2 shows the difference in hyper-volume indicator from an architectural design problem as a result of this resampling.

Furthermore, since order of action selection is not relevant for these problems, updating the pareto set at any given time consists of finding a visited node that is non-dominated in the current pareto set and adding that node to the pareto set.

4 Wind Farm Optimization

This project draws inspiration from (Bai et al. 2022) in exploring the design space of layouts for wind farms. Designing wind farms is a difficult task, and with larger farms can become intractable given the multi-dimensional nature of conflicting objectives. One notorious problem is placing turbines while accounting for the effects of wake from surrounding turbines. Figure 3 from Bai et al. 2022 describes this effect. Large wind farms with many turbines close together will see their annual energy production (AEP) drop due to this effect, while trying to reduce this effect might increase costs as more land is required to reach production goals.

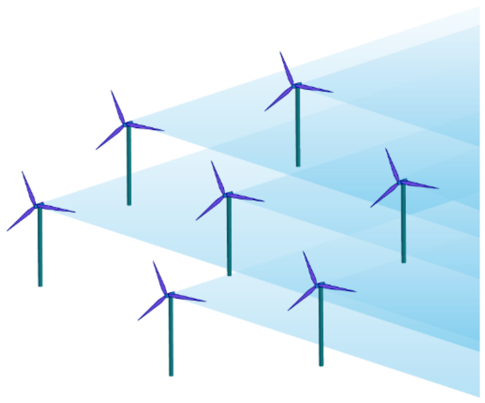


Figure 3: Wake effects in a wind farm (from Bai et al. 2022)

.

4.1 MDP Formulation

With little to no knowledge of wind farm setup, dynamics, or costs, the `Python` package `PyWake` (`https://topfarm.pages.windenergy.dtu.dk/PyWake/index.html`) is used to obtain AEP for any given wind farm layout.

State A state is defined as an $n \times m$ grid represented as a `numpy array`, with `0`s representing an empty location and `1`s representing a turbine placement.

Action Actions are defined as placing a new turbine in an empty slot subject to constraints. Constraints include a maximum number of turbines placed, maximum amout of land used, and a minimum distance each turbine must be from another due to turbine geometry.

Rewards There are many possible rewards to consider in this scenario. The rewards considered in this formulation include annual energy production (AEP), $\frac{1}{\text{num. turbines}}$, and $1 - \frac{\text{land used}}{\text{max land}}$. All rewards are scaled to the interval $[0, 1]$ and are to be maximized. AEP is scaled by some theoretical maximum AEP which is set to 200 GWh for this project and land used includes a rectangular boundary surrounding the land required by the minimum distance constraint.

Terminal States A state is to be considered terminal if the maximum number of turbines are placed or the maximum amount of land is used.

5 Results

For each of the following MO-MCTS runs the wind farm MDP is initialized as a discretized 40×40 grid representing an area of 2km $\times$ 2km. For our constraints, the maximum number of turbines is set at 20, and the proximity constraint is set at 200m. An initial placement of one turbine is chosen to be at [20, 20]. Each of the following runs of MO-MCTS is run 10,000 times and take around 10 minutes.

5.1 Reward: AEP and $\frac{1}{\textbf{num. turbines}}$

For this scenario, we would expect the MO-MCTS to return a pareto set with turbines approximately equally spaced across the grid in a way such that their wakes do not significantly interfere with other turbines. This design tradespace shows what an optimal layout would look like with 1-20 turbines in the grid.

Figure 4 shows how the hyper-volume indicator changes over each MO-MCTS iteration. Figure 5 show two of the optimal solutions found by the MO-MCTS algorithm and then Figure 6 shows the resultant pareto front.

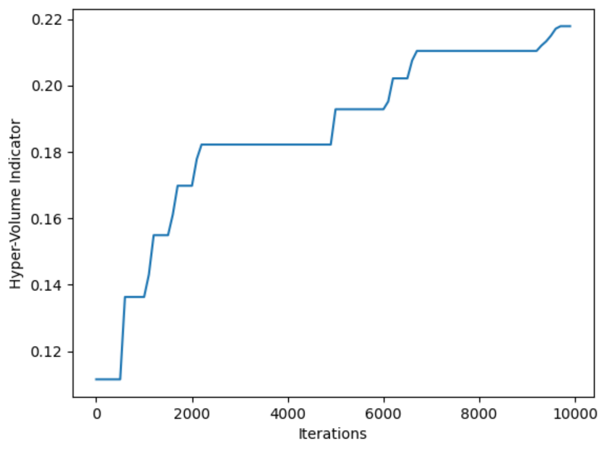
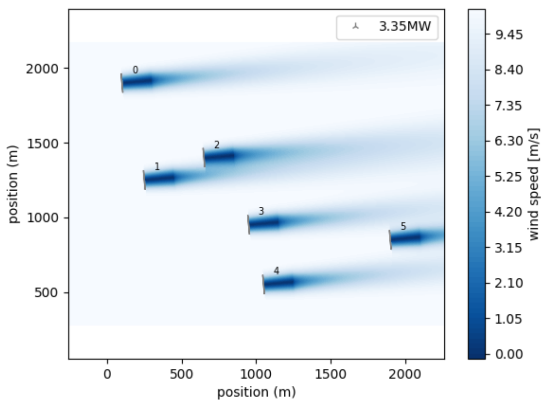
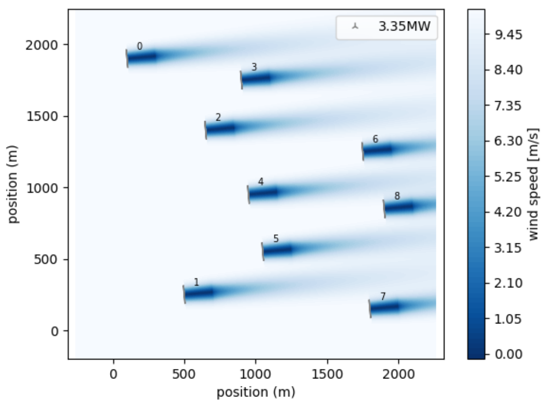


Figure 4: HVI increases per iteration

Of note from these sample optimal layouts in Figure 5 is that the algorithm clearly takes into account the wake effect from the turbines and so places them in a way such that essentially no wakes interact with other turbines. Obviously this is heavily dependent on the wind direction which in these runs was kept as being from a single direction, when in reality winds may shift dramatically over the course of a year. Also of note is that since there are



(a) Optimal layout with 6 turbines



(b) Optimal layout with 9 turbines

Figure 5: Two pareto optimal solutions found in MO-MCTS

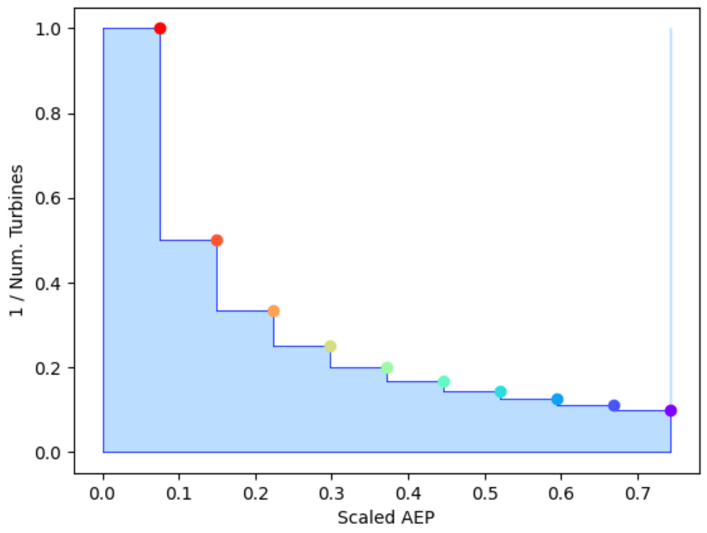


Figure 6: Pareto front for MO-MCTS runs with AEP and number of windmills as rewards.

no constraints on total land area used in this run, MO-MCTS simply places the turbines fairly far apart from each other, another way that it is decreasing the likelihood that the wakes interfere with other turbines.

5.2 Reward: AEP and $1 - \frac{\text{land used}}{\text{max land}}$

For this scenario we would expect the MO-MCTS to return a pareto set with turbines equally spaced across the grid with the total land they encompass somewhat proportional to the amount of turbines in the farm. That is, the optimal layout of two turbines should be a layout that places them as close as possible together instead of opposite ends of the grid. This design tradespace provides the optimal placement of a number of turbines given that more land is more expensive.

Figure 7 shows how the hyper-volume indicator changes over each MO-MCTS iteration. Figure 8 shows an optimal solution found by the MO-MCTS algorithm and then Figure 9 shows the resultant pareto front.

Including land usage as a reward gives slightly different and perhaps unexpected results from the MO-MCTS run. First, in Figure 9, there is an obvious case of clustered solutions on the pareto front, that is, there exist closely related solutions the differences between which are probably trivial. Finding these clustered solutions really only takes away computation time from the MO-MCTS algorithm that it could be using to find more novel solutions, and so is generally an undesired result. Regardless, Figure 8 shows an expected result, minimizing land usage while maximizing AEP should generally result in a straight lineup of turbines normal to the wind direction. Again, including changing wind direction would alter this result, but for cases when wind direction is generally constant this is a good result.

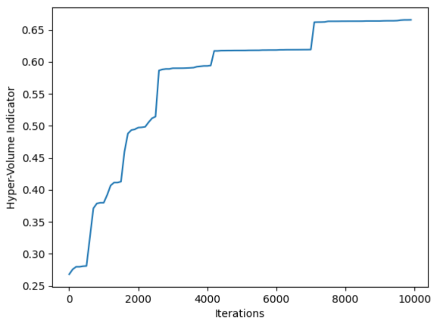


Figure 7: HVI increases per iteration

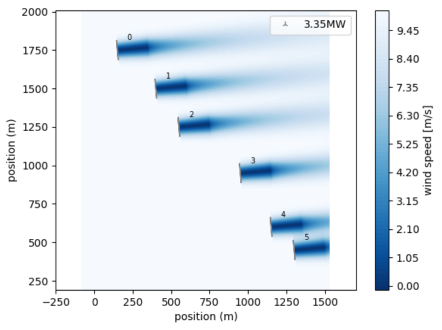


Figure 8: Optimal layout with 6 turbines

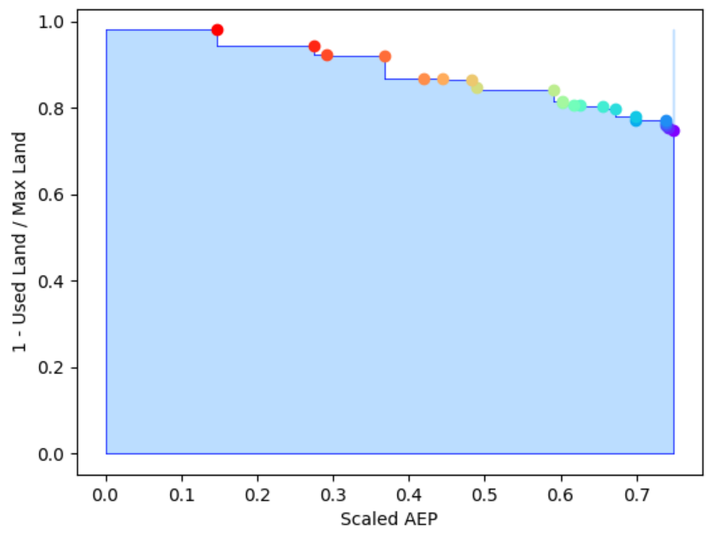


Figure 9: Pareto front for MO-MCTS runs with AEP and total land used by the turbines.

5.3 Reward: AEP, $\frac{1}{\text{num. turbines}}$, and $1 - \frac{\text{land used}}{\text{max land}}$

Finally, the MO-MCTS algorithm is run including all three rewards. Because of the way the penalty in MO-UCT was implemented for the two dimensional case, it does not currently work for the three dimensional case. Therefore, MO-UCT is used without a penalty for dominated points. In previous tests this has not significantly affected performance, but it is something to note in this run. For this case, the MO-MCTS algorithm is run for 50,000 iterations instead of 10,000.

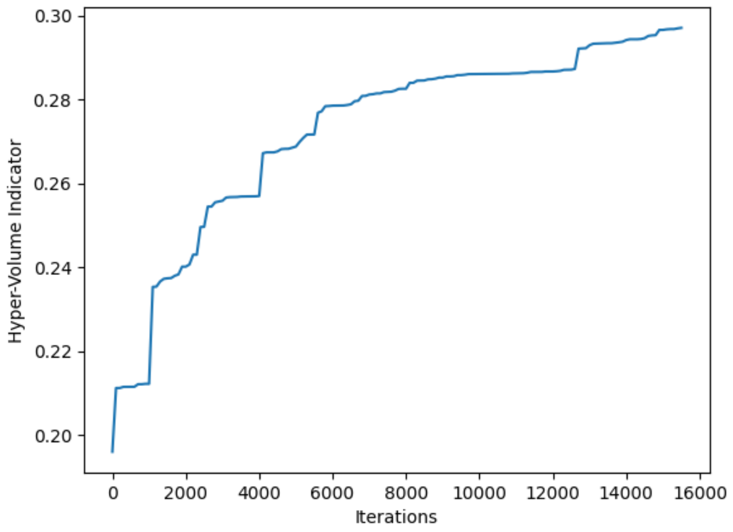
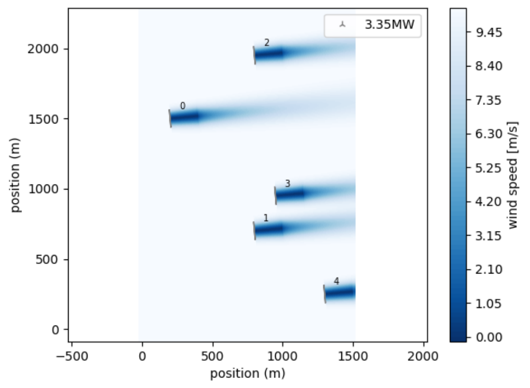
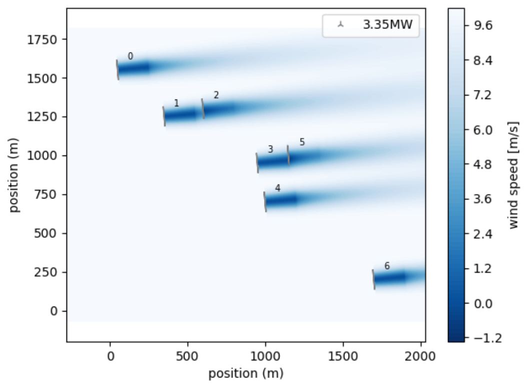


Figure 10: HVI increases per iteration

Figure 10 shows the changes in hyper-volume indicator per iteration in this case and Figure 11 shows two pareto optimal layouts from this run. Interestingly, MO-MCTS finds that in some cases, it may be pareto optimal to place turbines with one nearly directly behind another. This is clearly an attempt by MO-MCTS to reduce the land area used, and it has found that in some cases doing so can increase the hyper-volume indicator of the pareto front. This is an unexpected result and shows the ability of MO-MCTS to come up with novel results for an architectural design problem. The other solution presented seems to be more like the solutions found from optimizing the number of turbines.



(a) Optimal layout with 5 turbines



(b) Novel optimal layout with 7 turbines

Figure 11: Pareto optimal solutions from MO-MCTS with three dimensional reward vector

6 Conclusion

In this project, a multi-objective Monte-Carlo tree search algorithm using the hyper-volume indicator was implemented in `Python` and successfully tested on a simple wind farm MDP. The outputs of the MO-MCTS runs with this MDP consisted of pareto optimal layouts of turbines in a wind farm that took into account multiple competing objectives, including annual energy output (AEP), number of turbines, and total land area used. As expected, it was found that essentially any pareto optimal layout of a wind farm must take into account the wake effects of neighboring turbines. Importantly, the turbines must be arranged in such a way that reduces as much as possible the effect of wake on other turbines, thus maximizing AEP. The author concludes that MO-MCTS is a powerful tool for exploring these types of design state spaces, and that essentially any architecture design problem that can be formulated as an MDP can be run through this algorithm.

Python Packages Used

- `numpy`
- `pymoo`
- `xarray`
- `pywake`
- `scipy`
- `shapely`
- `matplotlib`

Note While the author grants permission for this paper to be made public, this project made use of part of the author's research code which was modified to fit into the project requirements. Code can be requested for review but probably cannot be shared for use or modification at this time.

References

Bai, Fangyun et al. (2022). “Wind farm layout optimization using adaptive evolutionary algorithm with Monte Carlo Tree Search reinforcement learning”. In: *Energy Conversion and Management* 252, p. 115047.

Browne, Cameron B et al. (2012). “A survey of monte carlo tree search methods”. In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1, pp. 1–43.

Hennes, Daniel and Dario Izzo (2015). “Interplanetary trajectory planning with Monte Carlo tree search”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence.*

Huang, Xi, Ziyu Shao, and Yang Yang (2020). “MIPS: Instance Placement for Stream Processing Systems based on Monte Carlo Tree Search”. In: *CoRR* abs/2008.00156. arXiv: 2008.00156. URL: https://arxiv.org/abs/2008.00156.

Rossi, Leonardo, Mark HM Winands, and Christoph Butenweg (2021). “Monte Carlo Tree Search as an intelligent search tool in structural design problems”. In: *Engineering with Computers*, pp. 1–18.

Shen, Linzhi and Shaowei Wang (2021). “Monte Carlo Tree Search for Network Planning for Next Generation Mobile Communication Networks”. In: *2021 IEEE Global Communications Conference (GLOBECOM).* IEEE, pp. 1–6.

Silver, David et al. (2016). “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587, pp. 484–489.

Świechowski, Maciej et al. (2021). “Monte Carlo tree search: A review of recent modifications and applications”. In: *arXiv preprint arXiv:2103.04931.*

Wang, Weijia and Michele Sebag (2012). “Multi-objective monte-carlo tree search”. In: *Asian conference on machine learning.* PMLR, pp. 507–522.