Monte Carlo Tree Search for Multi-function Radar Task Scheduling

Marc Vincent, Amal El Fallah-Seghrouchni, Vincent Corruble, Narayan Bernardin, Rami Kassab, Frédéric Barbaresco

Monte Carlo Tree Search for Multi-function Radar Task Scheduling

Marc Vincent[1,2], Amal El Fallah Seghrouchni[2,3], Vincent Corruble[2], Narayan Bernardin[1], Rami Kassab[1], and Frédéric Barbaresco[1]

[1] Thales Land and Air Systems, 91338 Limours, France
marc.vincent@thalesgroup.com
[2] LIP6, Sorbonne Université, 75252 Paris, France
[3] AI Movement – International Artificial Intelligence Center of Morocco, University Mohammed VI Polytechnic, Rabat, Morocco

Abstract. Multi-function radars require efficient resource management strategies to fulfill their missions. In particular, task scheduling is crucial to mitigate difficult situations such as when all tasks cannot be accomplished. However, current approaches may prove insufficient in the face of emerging threats. In this article, we present a new formulation for the scheduling problem. Our model allows building schedules in a flexible way, which facilitates the discovery of high-value solutions using heuristics or tree search. We show that our algorithms provide noticeable performance improvement over similar methods proposed previously.

Keywords: Artificial Intelligence · Multi-function Radar · Combinatorial Optimization · Scheduling.

1 Introduction

Multi-function radars (MFR) are a class of radars that are able to concurrently perform a range of functions that would otherwise have to be carried out by several distinct radars. MFRs have been made possible by the development of phased-array antennas, which enhance the radar's flexibility by enabling greater control over waveforming and beamforming. An MFR's functions usually include search, tracking, and various combat-assistance roles such as missile guidance. Given that a radar runs under constraints of time, power, and processing, a key challenge of MFR design is radar resource management, which consists in allocating these resources between functions [7]. Resources must be allocated according to the radar's mission, which determines the priority level of each function.

Among these resources, time budget is the most critical, as the time spent performing a given task closely reflects its importance. Each function consists of one or more tasks, for example tracking a specific target; each task is carried out by performing a number of dwells characterized by requirements in desired execution time, duration, and power. Based on the current situation, the radar continuously generates dwell requests corresponding to its different functions.

The resulting list of requests is regularly transmitted to a scheduler, whose role is to decide which dwells should be executed and at what time. Efficient scheduling is crucial in overload situations, which happen when temporal constraints prevent from executing all dwells, forcing the scheduler to drop some of them according to their level of priority.

This scheduling problem is usually tackled with heuristics, yet such methods become increasingly suboptimal as the complexity of the situations radars face grows. These situations involve new threats, like drone swarms, which are likely to create overload, and hyper-maneuvering, high-velocity, and furtive targets, which might require more resources to track efficiently. Proposed alternatives make use of a range of methods, including expert systems, metaheuristics, and neural networks [5, 7, 10, 11].

In this work, we treat task scheduling as a sequential decision problem. In recent years, in the wake of significant progress in the field of combinatorial games, such as for Go [9], this kind of approach has been extended successfully to "single-player" combinatorial settings [6]. We build upon the work of Gaafar et al. [4], who first proposed to apply this type of technique to radar task scheduling.

First, we present our main contribution, a formulation of the radar task scheduling problem as a Markov decision process that allows building schedules in a flexible way. Then, we exemplify the usefulness of our model with a heuristic and a variant of Monte Carlo Tree Search (MCTS) adapted to this formulation. Finally, we evaluate our algorithms and demonstrate the improvement in performance brought by our approach.

2 Framework

We formalize the problem of radar task scheduling similarly to Gaafar et al. [4]. As we mentioned above, our model of an MFR scheduler processes one fixed set of tasks[1] at a time. An updated set of tasks is received at regular intervals. In the meantime, no tasks may be added to the set that is being processed. Each task in a set $I = \{1, ..., n\}$ is characterized by its temporal constraints and its priority level. The temporal constraints of a task i are defined by its length L^i, its start time T_s^i and drop time T_{dr}^i (respectively the earliest and latest date at which it can start executing), and its due time T_{du}^i, which is the desired execution time. In this work, for any given instance, these constraints are all held constant over the course of the scheduling process: we do not consider preemptive tasks (e.g. for dwell interleaving) or variable duration time. Our goal is to determine for each task i in I whether to schedule it ($x^i = 1$, else 0), and if so at what execution time t^i, or to drop it ($y^i = 1$, else 0). Scheduled tasks must be entirely contained in a temporal frame $[0, T_{max}]$. In order to arbitrate between tasks in case of conflicts, each task is ascribed a drop cost C_{dr}^i and a delay cost C_{de}^i which reflect its priority level. The drop cost is incurred only when the corresponding task is

[1] In the rest of this article, we will use the standard terminology for scheduling, where "task" refers to the basic elements of a schedule—in our case, radar dwells. It should not be confused with the radar tasks mentioned earlier.

dropped while the delay cost determines how much the difference between the actual and ideal execution times is penalized; this difference is an absolute value, unlike in [4] where due times were not distinct from start times. How to assign relevant values to these costs in an operational context is left to future work. We write instances of the problem as $P = \{(L^i, T_s^i, T_{dr}^i, T_{du}^i, C_{dr}^i, C_{de}^i)\ \forall i \in I\ ; T_{max}\}$, and its (partial) solutions, or schedules, as $s = \{(x^i, y^i, t^i)\ \forall i \in I\}$. The objective is to minimize the sum of costs (or total cost) $C_P(s)$. We summarize the problem below, where $\oplus$ represents an exclusive or:

$$\min C_P(s) = \sum_{i \in I} x^i \left| t^i - T_{du}^i \right| C_{de}^i + y^i C_{dr}^i$$
$$\text{s.t.} \begin{cases} T_s^i \leq t^i \leq T_{dr}^i\ \forall i \in I \\ t^i + L^i \leq T_{max}\ \forall i \in I \\ t^i + L^i \leq t^j \oplus t^j + L^j \leq t^i \text{ if } x^i = x^j = 1,\ \forall (i < j) \in I^2 \\ t^i \in \mathbb{R}^+\ \forall i \in I \\ x^i, y^i \in \{0,1\} \text{ with } x^i = 1 - y^i,\ \forall i \in I \end{cases} \tag{1}$$

This problem can be solved to optimality with mixed-integer programming (MIP). Unfortunately, since problem (1) is in NP[1], the computation time for MIP grows exponentially with the number of tasks, making it prohibitive for radar applications. In order to give further insight into the structure of the problem, we offer its detailed formulation for MIP:

$$\min C_P(s) = \sum_{i \in I} l_{de}^i C_{de}^i + (1 - x^i) C_{dr}^i$$
$$\text{s.t.} \begin{cases} \left.\begin{array}{l} T_s^i \leq t^i \leq T_{dr}^i \\ t^i + L^i \leq T_{max} \\ l_{de}^i \geq t^i - T_{du}^i \\ l_{de}^i \geq T_{du}^i - t^i \end{array}\right\} \forall i \in I \\ \left.\begin{array}{l} t^i + L^i \leq t^j + M(n^{ij} + o^{ij}) \\ t^j + L^j \leq t^i + M(n^{ij} + 1 - o^{ij}) \end{array}\right\} \forall (i < j) \in I^2 \\ n^{ij} = 2 - x^i - x^j\ \forall (i < j) \in I^2 \\ x^i \in \{0,1\},\ t^i, l_{de}^i \in \mathbb{R}^+\ \forall i \in I \\ o^{ij} \in \{0,1\},\ n^{ij} \in \mathbb{R}^+\ \forall (i < j) \in I^2 \end{cases} \tag{2}$$

where M is an arbitrarily large number such that $M \gg T_{max}$. Of interest in formulation (2) is the presence of binary variables o^{ij} in addition to x^i. The values of o^{ij} determine the order in which the scheduled tasks are placed. This highlights the fact that problem (1) can be subdivided in three successive sub-problems:

1. *inclusion*: determine which tasks to schedule;
2. *ordering*: determine the order in which these tasks should be scheduled;
3. *time setting*: determine the execution times of these ordered scheduled tasks.

[1] This can be proven via a polynomial reduction with the knapsack problem by setting $T_s^i = C_{de}^i = 0$ and $T_{dr}^i = T_{max}$ for all i in I.

Note that problem (1) can also be solved with heuristics like earliest start time first (EST) or earliest deadline first (EDF). These heuristics (and variations thereof) are common in radar resource management because they are fast and easy to implement. However they will often produce poor solutions, for two reasons: first, because they operate on the premise that the schedule must be built by adding tasks in chronological order; second, because they do not account for task priority.

By contrast, we aim to develop approximate algorithms for radar task scheduling that can approach optimal solutions while keeping reasonable computation times. Our first step is to model problem (1) as a Markov decision process (MDP). MDPs are the most common formalization of sequential decision problems. An MDP is defined by: a finite state space $\mathcal{S}$, a finite action space $\mathcal{A}$, a distribution over initial states $\mu : \mathcal{S} \to [0, 1]$, a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, which gives the conditional probability $\mathcal{T}(s' \mid s, a)$ of transition to the next state s' given the previous state s and selected action a, and a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ that gives the reward associated with a transition. We can associate a strategy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ to an MDP, i.e. a probability distribution over which action to take in a given state. The objective is usually to maximize the value function, which is the expectation of the sum of rewards: $V^\pi(s_0) = \mathbb{E}_{a_t \sim \pi, s_t \sim \mathcal{T}} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) \right]$ where $\gamma \in [0, 1]$ is the discount factor and T the final step of an episode. Once we have defined a suitable MDP, we will need an algorithm that outputs a strategy for this MDP, which we call the decision algorithm.

Following Gaafar et al. [4], we define a state as a partial schedule, where a number of tasks have been scheduled at certain execution times and a number of others dropped while respecting the constraints of (1). The only initial state s_0 for a given instance of the problem is the associated empty schedule, where no tasks are scheduled or dropped. We also define an action as the choice of one task to schedule in the set of available tasks $A = \{i \in I \mid x^i = y^i = 0\}$. Terminal states are states where no actions are available, that is, complete schedules, where all tasks are scheduled or dropped. We can define the reward as the cost difference between the two states involved in a transition, i.e. $\mathcal{R}(s, a, s') = C_P(s) - C_P(s')$. With $\gamma = 1$, the value function becomes $V^\pi(s_0) = \mathbb{E}_\pi \left[C_P(s_0) - C_P(s_T) \right]$, which corresponds to our initial objective of finding a schedule s that minimizes $C_P(s)$. In practice, in our algorithms, we reason directly on costs.

The choice of the transition function is the most crucial aspect of this formalization. Transitions must deterministically decide what execution times to set and which tasks to drop when a new task is added to a partial schedule.

The simplest option, used in [4], is to add tasks chronologically: any newly added task i is assigned an execution time greater than that of all previously scheduled tasks, and tasks that cannot be placed after i are dropped. However, this restricts the range of viable strategies, since the decision algorithm has to solve both the inclusion and the ordering sub-problems at the same time, while the transition function only deals with the time setting sub-problem.

Instead, we propose a transition model that allows building schedules in a more flexible way. Our idea is to model the transition function such that it solves both the ordering and the time setting sub-problems at each step. That is, given a set S of scheduled tasks, the transition determines the order of tasks and their execution times by minimizing the total delay cost:

$$\begin{aligned} &\min \sum_{i \in S} l_{de}^i C_{de}^i \\ &\text{s.t.} \begin{cases} \left.\begin{array}{l} T_s^i \leq t^i \leq T_{dr}^i \\ t^i + L^i \leq T_{max} \\ l_{de}^i \geq t^i - T_{du}^i \\ l_{de}^i \geq T_{du}^i - t^i \end{array}\right\} \forall i \in S \\ \left.\begin{array}{l} t^i + L^i \leq t^j + M o^{ij} \\ t^j + L^j \leq t^i + M(1 - o^{ij}) \end{array}\right\} \forall (i < j) \in S^2 \\ t^i, l_{de}^i \in \mathbb{R}^+ \; \forall i \in S \\ o^{ij} \in \{0, 1\} \; \forall (i < j) \in S^2 \end{cases} \end{aligned} \tag{3}$$

Given a partial schedule $s = \{(x^i, y^i, t^i) \; \forall i \in I\}$, in order to schedule a new task $j \in I$, we solve sub-problem (3) for $S = \{i \in I \mid x^i = 1\} \cup \{j\}$. If there is no solution to the sub-problem, j has to be dropped. After a task is scheduled, all remaining tasks in A can be tested for dropping using this procedure.

Sub-problem (3) can also be treated with a MIP solver; however this requires finding the order and execution times of all tasks in S at every transition. This is inefficient, because with this transition model, when we move from a partial schedule s to a new one s' by adding a task i, the execution times in s are already optimal with regard to the tasks scheduled in s. This means that if for example i can be placed at its desired execution time without interfering with already scheduled tasks, then there is no need to recompute the execution times of these tasks. To exploit this and a number of other optimizations, we implement a custom solver for sub-problem (3) which we do not detail here due to space limitations: the key idea is to recursively generate and evaluate permutations of the scheduled tasks while limiting the number of generated permutations by exploiting temporal constraints. Note that this way, we can check the availability of a task faster, by interrupting the recursion as soon as we find a feasible permutation.

Our transition model allows adding tasks in any order by delegating part of the optimization—ordering and time setting—to the environment. Crucially, for any schedule s reached through this transition model, the execution times are optimal given the tasks scheduled in s. (We can also limit the number of generated permutations to lower the computation time, although we then lose the optimality guarantee.) This opens new possibilities for the choice of the decision algorithm, whose role is to solve the hardest part of the problem: inclusion.

3 Methods

The first decision algorithm we propose is a heuristic that we call highest cost-length ratio first (HCLR), which is similar to some knapsack problem heuristics. It consists in sorting tasks by their ratio between drop cost and length, then scheduling them in decreasing order (if they are still available when their turn comes). Equivalently, at each step, we choose the following action:

$$a = \text{argmax}_{a \in A} \frac{C^i_{dr}}{L^i}$$

This heuristic empirically performs better than scheduling the task with the highest drop cost first, because it accounts for situations where for example two shorter, lower-priority tasks have a higher total drop cost than one longer, higher-priority task which they are incompatible with. A similar criterion has been proposed for radar resource management in Qu et al. [8]; however, the authors used the ratio in a task selection phase that preceded the scheduling itself, which is based on EST. Using our transition model, we can instead directly select and schedule each task in turn.

Our next decision algorithm is a version of Monte Carlo Tree Search (MCTS) adapted to task scheduling, which is based in large part on the version of Gaafar et al. [4]. MCTS uses a search tree where nodes represent MDP states and branches correspond to actions taken in the parent node [1]. The tree is constructed by successive rollouts: starting from the root node, which corresponds to the initial state, we select an action, apply the transition function to get the next node (which is created if needed), and repeat. Once a terminal state is reached, we can compute its total cost C, then backpropagate C up the path we just followed to update the best cost reached from each state-action pair: $C(s,a) \leftarrow \min\{C(s,a), C\}$. This value is then used in the computation of the upper-confidence bound $U(s,a)$ which determines which actions are chosen in the selection phase: $a = \text{argmax}_{a'} U(s,a')$ with $U(s,a) = \frac{P(s,a)}{C(s,a)^\tau(1+N(s,a))}$ where τ is a temperature, $N(s,a)$ is the number of rollouts where action a was taken in state s, and $P(s,a)$ is a prior probability function over actions. This selection rule is designed to balance exploration of the search tree and exploitation around the best solutions found so far. The prior, especially, plays a prominent role in steering exploration; a simple way to parameterize it is to sort the m available tasks according to a criterion, then assign them a respective prior probability of $p(1-p)^m$ for m ranging from 0 (for the first task) to $m-1$. For our experiments, we set $\tau = 2$ and $p = 0.6$, similarly to [4], but we diverge by using HCLR as our sorting criterion instead of EST.

One issue is that when using our transition model, since tasks can be scheduled in any order, it is possible to reach the same terminal state via multiple different rollouts. In order to prevent this, we structure our search tree similarly to a branch-and-bound (B&B) tree [2]: when a new action is taken in node s, leading to a new node s', all actions explored in s in previous rollouts are made unavailable in s' and its descendants. This makes sure there is only one path

to each terminal state in the search tree. However, it also means that we may reach terminal nodes that are not complete schedules, when some tasks can still be scheduled, but have all been made unavailable by the previous rule. To limit the number of such situations, at each visit of a node s, we check if there exists a terminal node s' descended from s such that all tasks that are unexplored in s are scheduled in s'; if so, these tasks are made unavailable in s. Additionally, to avoid performing the same rollout twice, if a node has no more available actions, the action that led to it is also made unavailable in the parent node, as in [4]. We call this algorithm B&B-MCTS.

Usually, in MCTS, when we reach a new state, we want to know which actions are available. With B&B-MCTS, this allows making the most effective use of pruning, according to the above rules. However, with our transition model, it requires partially solving (3) for each a priori available task to check if it has to be dropped. In larger instances, this involves significant computation, which reduces the number of rollouts that can be carried out in a given time. For this reason, we program B&B-MCTS so that it attempts to schedule tasks without prior verification, and drops them only if the attempt fails.

4 Results

To enable comparisons, we run experiments on instances from the same distribution as in [4]: $L^i \sim U(2, 15)$, $T_s^i \sim U(0, T_{max} - 12)$, $T_{du}^i = 0$, $T_{dr}^i - T_s^i \sim U(2, 12)$, $C_{dr}^i \sim U(100, 500)$, $C_{de}^i \sim U(1, 15)$, $T_{max} = 100$. The difficulty of an instance mostly depends on the density of tasks; by keeping T_{max} fixed, the difficulty can be controlled by setting the number of tasks.

We compare our two methods, HCLR and B&B-MCTS, with the EST heuristic and with the version of MCTS proposed in [4] (which we term EST-MCTS). The run time of both versions of MCTS is limited to 1 second. All these algorithms are implemented in Python. We compute the optimal solution using MIP on instances where this resolution can be performed in a reasonable amount of time. The MIP solver we use is CBC [3]. These five algorithms are run on the same instances, 1000 per number of tasks. We also compare our results to those reported by [4] for a reinforcement learning-based extension of EST-MCTS inspired by AlphaZero [9].

Our results show that our approach provides a significant performance improvement over EST-MCTS, and even surpasses the reinforcement learning algorithm of [4], which used deep learning in conjunction with MCTS. Strikingly, the performance gain of B&B-MCTS over HCLR proves minimal. Moreover, with our implementation, HCLR's run time averages 36 milliseconds on 50-task instances, which would presumably make it more suitable for radar use cases than MCTS-based methods.

Unlike in [4], our approach allows due times distinct from start times, thus reflecting radar requirements more closely. We also run experiments on a similar task distribution but with $T_{du}^i \sim U(1, 4)$ and $T_{dr}^i - T_s^i \sim U(1, 8)$; our results match those of the first distribution very closely.

Table 1. Detailed statistics on three different instance types.

Algorithm	Avg. cost	Cost std. dev.	Dropped tasks (%)	Optimal solutions (%)	Avg. runtime (milliseconds)	Avg. number of rollouts
Number of tasks = 25, identical start and due times						
EST	4658.69	634.65	59.06	0.0	1.93	nan
EST-MCTS	3716.55	506.0	52.15	0.87	1002.88	727.06
HCLR	3390.6	551.68	48.2	21.35	13.12	nan
B&B-MCTS	3299.39	526.16	48.44	57.73	975.32	96.12
MIP-optimal	3268.43	523.85	48.65	100.0	1730.82	nan
Number of tasks = 50, identical start and due times						
EST	12031.8	857.18	77.64	nan	2.32	nan
EST-MCTS	10897.03	758.17	73.23	nan	1004.79	587.63
HCLR	9130.98	799.9	63.62	nan	36.7	nan
B&B-MCTS	9080.49	780.86	63.79	nan	991.72	30.1
Number of tasks = 25, distinct start and due times						
HCLR	3301.15	550.93	47.53	22.0	19.98	nan
B&B-MCTS	3219.61	524.51	47.61	56.5	994.69	70.96
MIP-optimal	3185.86	520.0	47.88	100.0	2219.44	nan

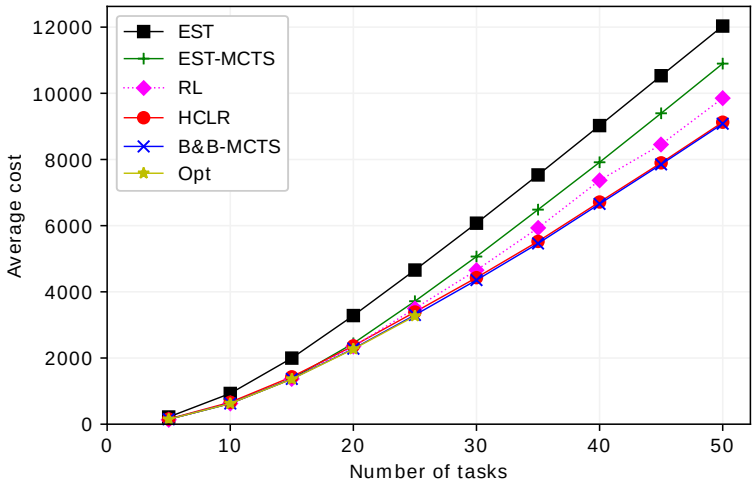

Fig. 1. Cost plotted against instance size for identical start and due times.

5 Conclusion

Through a reformulation of the problem of radar task scheduling, we were able to implement an efficient framework whose algorithmic applications can return quasi-optimal solutions in a limited amount of time. We see potential improvements for this framework, both by optimizing its run time to make it usable by real-world radars, and by increasing its flexibility; for example, being able to remove a task from a partial schedule could allow exploring the solution space more effectively. Furthermore, our B&B-MCTS algorithm could be extended with an AlphaZero-style reinforcement learning procedure which would make it able to adapt to various task distributions.

References

1. Browne, C.B., et al.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (Mar 2012)
2. Clausen, J.: Branch and Bound Algorithms - Principles and Examples. Tech. rep., University of Copenhagen (2003)
3. COIN-OR Foundation: Cbc (COIN-OR Branch-and-Cut solver), https://github.com/coin-or/Cbc
4. Gaafar, M., et al.: Reinforcement Learning for Cognitive Radar Task Scheduling. In: 2019 53rd Asilomar Conference on Signals, Systems, and Computers (Nov 2019)
5. Jeauneau, V., Guenais, T., Barbaresco, F.: Scheduling on a fixed multifunction radar antenna with hard time constraint. 14th International Radar Symposium (IRS) **1**, 375–380 (2013)
6. Laterre, A., et al.: Ranked Reward: Enabling Self-Play Reinforcement Learning for Combinatorial Optimization. ArXiv (2018)
7. Moo, P.W., Ding, Z.: Adaptive Radar Resource Management. Elsevier (2015)
8. Qu, Z., Ding, Z., Moo, P.: A Machine Learning Task Selection Method for Radar Resource Management (Poster). In: 2019 22th International Conference on Information Fusion (FUSION). pp. 1–6 (Jul 2019)
9. Silver, D., et al.: Mastering the game of Go without human knowledge. Nature **550**(7676), 354–359 (Oct 2017)
10. Winter, É., Baptiste, P.: On scheduling a multifunction radar. Aerospace Science and Technology **11**, 289–294 (2007)
11. Zheng, Q., Barbaresco, F., P.Baptiste: On scheduling a multifunction radar with duty cycle budget. Cognitive Systems with Interactive Sensors pp. 1–6 (2009)