Task Selection and Scheduling in Multifunction Multichannel Radars

Mahdi Shaghaghi and Raviraj S. Adve
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada
Email: shaghagh@ece.utoronto.ca, rsadve@ece.utoronto.ca

***Abstract*—In a multifunction radar, several tasks with differing parameters, such as tracking and surveillance, must be scheduled on a timeline. In an overload situation, the scheduling can become a challenging problem, as some of the tasks may need to be delayed or even dropped. With recent advancements in multichannel radars, e.g., multifrequency radars, it is possible to perform multiple tasks on different channels in parallel. This leads to us considering the NP-hard problem of optimal task scheduling for multiple channels. We extend previously proposed heuristic approaches to the multichannel case; but our main contribution is an optimal solution based on the branch-and-bound (B&B) method. The heuristics are suboptimal but have the advantage of low computational complexity. On the other hand, the optimal solution provides significantly better performance than the heuristics, but has high computational burden and is likely impractical for real-time scheduling. However, the B&B approach does provide the performance upper bound on heuristics and can be used to train a cognitive task scheduler.**

I. Introduction

A modern radar may be designed to perform multiple functions, such as surveillance, tracking, and fire control. Each function requires the radar to execute a number of transmit-receive tasks. This raises the problem of assigning radar resources, such as the time, frequency and energy budget, to different tasks. Specifically, a radar resource management (RRM) module makes decisions on parameter selection, prioritization, and scheduling of such tasks [1]. RRM becomes especially challenging in overload situations, where some tasks may need to be delayed or even dropped.

Effective resource management for multifunction radars (MFR) has been addressed in the literature (see for example [2]–[4] and references therein). Each radar task comprises transmission, waiting, and reception intervals. RRM, specifically task scheduling, is, in general, an NP-hard problem. As a common theme in the literature, RRM is split into two steps [1]: first, task parameters such as the priority, dwell time, and revisit interval are determined, such as by using rule-based methods for each task individually [5] or by joint optimization, across all tasks, of an overall utility function, while accounting for the resource constraints [2], [6]. Once the parameters are determined, task selection and scheduling is performed. In the selection phase, based on resource constraints and a figure of merit, a subset of the tasks are chosen and the rest dropped. The scheduling phase then assigns time slots to each task. These two steps can also be repeated iteratively to improve performance.

The key difference in this paper is that, in the previous works, the radar functions are performed on a single frequency channel or "timeline". However, with *multichannel*, e.g., multifrequency, radars becoming increasingly viable, the channels can be used to execute multiple tasks *simultaneously* [7]. In this paper, we consider the problem of joint resource management for a multichannel, multifunction, radar, i.e., one that is able to concurrently execute multiple tasks. We ensure that tasks are not scheduled in the interval between transmission and reception. Interleaving methods can also be studied to investigate the potential of more efficiently using the waiting times [8].

In considering RRM for multichannel radars, we begin with extending previously proposed, heuristic, techniques to the multichannel case. However, the main contribution of the paper is the development of branch-and-bound (B&B) techniques to solve the original NP-hard problem. Using the B&B approach leads to an optimal, in the sense of the chosen metric, solution to the task scheduling problem.

As a solution to an NP-hard problem, a simplistic implementation of the B&B algorithm is computationally infeasible. To make the approach as computationally feasible as possible, we implement multiple branch pruning techniques. As we will see, our efforts allow us to optimally schedule up to 40 tasks in a reasonable time (though, not probably in real-time). However, the main contribution of this approach is likely to be a *cognitive* task scheduler trained with optimal patterns given various task parameters.

This paper is organized as follows. Section II provides the problem formulation. We present two heuristic methods for the task selection and scheduling steps in Section III. The optimal solution for the joint problem of task selection and scheduling can be obtained using the branch-and-bound (B&B) method presented in Section IV. In this section, we also present branch pruning techniques to minimize the computation load. We compare the performance of the proposed methods using numerical simulations in Section V. Finally, Section VI wraps up the paper with some conclusions and discussion.

II. Problem Formulation

We consider N tasks to be executed on K identical channels within a time window T. Each channel is associated with a timeline. Tasks can be performed in parallel on different channels, but cannot overlap on a given timeline. Once a task has started executing, it cannot be stopped.

The duration (also length or dwell time) of the n-th ($1 \leq n \leq N$) task is denoted by ℓ_n. For each task, there is a *starting time* s_n after which the task is ready to be executed. There is also a *deadline* d_n after which the task cannot be scheduled, and, if not scheduled, must be dropped (with an

978-1-4673-8823-8/17/$31.00 ©2017 IEEE

associated *dropping cost* D_n). For example, consider a tracking task. The starting time of the task depends on the required tracking accuracy and the time when the last measurement was made. The deadline depends on the estimated trajectory of the target and the beamwidth of the radar. The task is dropped after the target is assumed to have moved out of the radar beam. The dropping cost depends on the priority of the task and the further actions required to compensate for the dropping.

A scheduled, but delayed, task suffers a *tardiness cost* which is, here, modeled as linearly proportional to the delay. Let e_n be the time when task n begins execution; the tardiness cost is given by $w_n(e_n - s_n)$, where w_n is the weight which scales the delay. Let the binary variable x_n indicate if task n is scheduled ($= 1$) or dropped ($= 0$). Then, the cost associated with the n-th task is given by $x_n w_n(e_n - s_n) + (1 - x_n)D_n$. Our joint task selection and scheduling problem is a minimization of the total cost C, given by

$$C = \sum_{n=1}^{N} x_n w_n(e_n - s_n) + (1 - x_n)D_n. \tag{1}$$

Here, the optimization variables are if a task is selected (choosing x_n) and, if $x_n = 1$, the timeline (out of K) on which the task is executed. The final variable is the execution time $e_n (\geq s_n)$. Our optimization problem is, therefore,

$$\begin{aligned} \{x_n^*, e_n^*\} \quad = \min_{x_n, e_n} & \textstyle\sum_{n=1}^{N} x_n w_n(e_n - s_n) + (1 - x_n)D_n \\ \text{s.t. } & x_n \in \{0, 1\}, n = 1, \ldots, N \\ & s_n \leq e_n \leq d_n, n = 1, \ldots, N \\ & \text{and no tasks overlap in time.} \end{aligned} \tag{2}$$

The scheduling problem without deadlines (and therefore without dropped tasks) is NP-hard [9]. It can be shown that the joint task selection and scheduling is also NP-hard.

III. Heuristic Methods

The joint problem at hand can be solved suboptimally using heuristic methods [10]. The problem is split into task selection and scheduling, and we iterate between these two steps. In this paper, we use the dropping costs of the tasks at the selection phase. Let $\mathbf{S}$ be the sequence of tasks sorted in non-increasing order based on their dropping costs. Furthermore, let $\mathbf{D}$ be the set of selected tasks. Initially, $\mathbf{D}$ is the empty set. In the selection step, we take the task with highest dropping cost from the sequence $\mathbf{S}$ which has not been selected in the previous iterations, and we add it to the set $\mathbf{D}$.

In the scheduling step, a heuristic method is used to schedule the tasks in $\mathbf{D}$ on the K timelines. The schedule is said to be *viable* if the execution times of all the scheduled tasks are before their corresponding deadlines. If the schedule is viable, we keep the last task added to $\mathbf{D}$; otherwise, it is removed from $\mathbf{D}$. Then, the next iteration is performed by going back to the selection step to add a new task from $\mathbf{S}$ to $\mathbf{D}$. This procedure continues until all tasks in $\mathbf{S}$ are checked (see Table I).

TABLE I. Selection and scheduling heuristic

Initialization
Let $\mathbf{S}$ be the sequence of tasks sorted based on the dropping costs.
$\mathbf{D} \leftarrow \{\}$
$i \leftarrow 1$

while $i \leq N$
Add the i-th task from $\mathbf{S}$ to $\mathbf{D}$.
If scheduling the tasks in $\mathbf{D}$ is not viable
Remove task i from $\mathbf{D}$.
$i \leftarrow i + 1$

Given a selected set $\mathbf{D}$, there are different methods that can be used to schedule the tasks on the timelines. In this paper, we consider two well-known methods: the earliest starting time (EST) first and the earliest deadline (ED) first [10]. Let $\mathbf{T}$ be a sequence obtained by sorting the tasks in $\mathbf{D}$. In the EST method, tasks are sorted in non-decreasing order based on their starting times. In the ED method, tasks are sorted based on their deadlines. Essentially, the EST method schedules tasks as soon as they are ready to be executed, and therefore, reduces delay times. On the other hand, the ED method favors tasks that have earlier deadlines, to reduce the dropping of tasks.

After obtaining the sequence $\mathbf{T}$ either using the EST or ED methods, the tasks are sequentially placed on the timelines at the earliest time possible. For scheduling each task, we select the earliest available channel. If two or more channels have the same earliest availability time, the one with the smallest index is chosen. The execution time of the task is set as the maximum of the starting time of the task and the time when the channel is available. We refer to this procedure as the *sequence to schedule mapping*.

After all the tasks from the sequence $\mathbf{T}$ are scheduled, we can check whether the scheduling is viable or not by checking if all the execution times of the tasks are before their corresponding deadlines. In the case that the scheduling is not viable, we can modify the order of tasks in $\mathbf{T}$. Specifically, for $j = 1, 2, \cdots, N - 1$, we swap the order of two consecutive tasks j and $j + 1$ in $\mathbf{T}$ to check whether the swapping results in a viable schedule. The search stops as soon as a viable schedule is found. We call this procedure *task swapping*. At this point, the scheduling step is complete, and we can go to the next iteration of task selection.

IV. Branch-and-Bound Method

The EST and ED heuristics are computationally efficient, but there has been no work on estimating the penalty for this computation efficiency. In order to find the optimal solution of the joint problem of task selection and scheduling as given in (1), we use the B&B procedure which implicitly enumerates all possible solutions by considering partial solutions in a tree structure. Each partial solution is a sequence of tasks. A schedule is obtained using the "sequence to schedule mapping" as explained in the previous section. In this way, instead of searching for the optimal schedule, we can search for the optimal sequence [11], [12].

The node at the root is an empty sequence. Nodes in the tree generate children by partitioning the solution space into smaller regions (branching). Each branch is associated with an unscheduled task in the parent node whose deadline has not passed yet. The node at the end of a branch is a new sequence

which is obtained by appending the task of the branch at the end of the sequence of the parent node.

A depth-first search strategy is used to traverse the nodes of the tree. Crucially, at each node, *bounds and dominance rules* are used to prune off branches and nodes that are provably suboptimal (bounding). When a node is pruned off, all of its children nodes are eliminated from the search. This step is crucial in making the B&B method computationally feasible. Once the entire tree has been explored, the best solution found in the search is returned.

The steps of the B&B method are listed in Table II. The search tree is implemented using a stack data structure (see Algorithm 1 in [12]). Here, we have modified the B&B method of [12] to include task dropping. Each element (node) of the stack is a tuple which consists of a sequence of tasks $\mathbf{T}$ (representing a partial schedule), a set of tasks that can be scheduled right after $\mathbf{T}$ (denoted by the possible first $\mathbf{PF}$ set), a set of tasks which have not yet been scheduled ($\mathbf{NS}$), and a set of tasks which have been dropped ($\mathbf{DR}$).

An upper bound UB holds the cost of the best complete solution obtained during the search. UB is initially set to infinity and an empty sequence with all the tasks in the PF set (which represents the root node of the tree) is pushed on the stack. Then, as long as the stack is not empty, the following procedure is performed: at each iteration, the node on top of the stack is checked. If there are no tasks in the possible first set $\mathbf{PF}$, the node will be removed from the stack. Before removing the node, in the case that there is no tasks left in the not-scheduled set $\mathbf{NS}$ (which represents a complete solution), the overall cost of the solution is compared with UB. If an improvement has been achieved, the current schedule is set as the best solution found so far $\mathbf{T}^*$, and UB is updated.

In the case that the possible first set of the node on top of the stack is not empty, a new node is generated using a task of $\mathbf{PF}$. Specifically, let j be a task in $\mathbf{PF}$. We remove j from $\mathbf{PF}$ and append it at the end of $\mathbf{T}$ to obtain the sequence of the new node (denoted by $\mathbf{T}'$). The possible first set of the new node is set as the union of $\mathbf{PF}$ and $\mathbf{NS}$. The dropped tasks are inherited from $\mathbf{DR}$, and the not-scheduled set of the new node is set to empty. Then, task j is added to $\mathbf{NS}$.

After a new node is generated, we determine whether it should be added to the stack for further investigation in the next iteration or it can be ignored and therefore pruned off. To do so, we first check the *start-times dominance rule* on $\mathbf{T}'$. This rule states that the sequence of execution times of tasks in $\mathbf{T}'$ should be non-decreasing; otherwise, $\mathbf{T}'$ is dominated and cannot result in an optimal solution [13].

We next check the tasks in $\mathbf{PF}'$, and any task whose deadline is before the earliest available time of the channels is dropped. At this point, the sum of the tardiness and dropping costs of the new partial solution can be computed, and it gives a lower bound on any complete solution derived from this node. If this cost is not lower than the cost of the best solution found so far, i.e., UB, the new node can be ignored. Furthermore, if $\mathbf{T}'$ does not map to an *active schedule*, it can be discarded. An active schedule is such that no task can be completed earlier without delaying another task [11]. In order to determine whether $\mathbf{T}'$ is active, the tasks in $\mathbf{PF}'$ are checked to see if any of them can be scheduled right before the last task in $\mathbf{T}'$ (on the same timeline) without imposing a delay.

TABLE II. BRANCH-AND-BOUND METHOD

Initialization
  $UB \leftarrow \infty$
  Let $\mathbf{T}$ be an empty sequence
  $\mathbf{PF} \leftarrow$ {all tasks}
  $\mathbf{NS} \leftarrow \{\}$; $\mathbf{DR} \leftarrow \{\}$
  Push $(\mathbf{T}, \mathbf{PF}, \mathbf{NS}, \mathbf{DR})$ tuple on STACK.

while STACK is not empty
  Let $(\mathbf{T}, \mathbf{PF}, \mathbf{NS}, \mathbf{DR})$ be the tuple on top of STACK.
  if $\mathbf{PF} \neq \{\}$
    Let $j \in \mathbf{PF}$
    $\mathbf{PF} \leftarrow \mathbf{PF} \setminus j$
    $\mathbf{T}' \leftarrow \mathbf{T}|j$; $\mathbf{PF}' \leftarrow \mathbf{PF} \cup \mathbf{NS}$; $\mathbf{NS}' \leftarrow \{\}$; $\mathbf{DR}' \leftarrow \mathbf{DR}$
    $\mathbf{NS} \leftarrow \mathbf{NS} \cup j$
    if $\mathbf{T}'$ follows the start-times dominance rule
      Move any task whose deadline has passed on all timelines from $\mathbf{PF}'$ to $\mathbf{DR}'$.
      $C' \leftarrow$ TardinessCost($\mathbf{T}'$) + DroppingCost($\mathbf{DR}'$)
      if ($\mathbf{T}'$ is active)
        and ($\mathbf{T}'$ is LOWS-active)
        and ($C' < UB$)
          Push $(\mathbf{T}', \mathbf{PF}', \mathbf{NS}', \mathbf{DR}')$ tuple on STACK.
  else
    $C \leftarrow$ TardinessCost($\mathbf{T}$) + DroppingCost($\mathbf{DR}$)
    if ($\mathbf{NS} = \{\}$) and ($C < UB$)
      $UB \leftarrow C$
      $T^* \leftarrow T$
    Remove $(\mathbf{T}, \mathbf{PF}, \mathbf{NS}, \mathbf{DR})$ tuple from STACK.

Next, we check if $\mathbf{T}'$ is a *LOWS-active* (LOcally Well Sorted active) schedule [12]. A LOWS-active schedule is such that no exchange of two adjacent tasks can improve the schedule. The adjacent tasks of task i are defined as the set of tasks scheduled on the final slot of each timeline just before scheduling task i. The conditions for checking if a schedule is LOWS-active are the same as given in [12, Section 4] with the exception that we also need to consider task dropping. The conditions need to be modified such that we consider the event when a swapping of two tasks requires one of the tasks to be dropped. In this case, the availability time of the timeline is used instead of the completion time and the dropping cost replaces the tardiness cost (since the task is not scheduled).

Finally, in the case that the bounds and dominance rules are passed, the new node is pushed on top the stack and the search continues by expanding this new node. Once the entire tree is explored, the best solution found is returned.

V. Simulation Results

We now present the results of simulations illustrating the performance of the heuristics and the B&B method. We consider a multichannel radar with up to $K = 4$ identical channels. There are different numbers of tasks distributed over a timeline window of 100 sec. The objective is to schedule the tasks such that the overall cost of dropping and delaying the tasks, the objective in (2), is minimized. In our simulations, the starting time of the tasks is uniformly distributed on the 100 sec time window, i.e. $s_n \sim \mathcal{U}(0, 100)$. (Here, $x \sim \mathcal{U}(a, b)$

represents a random variable uniformly distributed between a and b).

For each task, the interval between the starting time and the deadline, i.e. $d_n - s_n$, is sampled from $\mathcal{U}(2, 12)$. The task length ℓ_n is distributed according to $\mathcal{U}(2, 11)$. Furthermore, the dropping costs (D_n) and the tardiness weights (w_n) have uniform distributions $\mathcal{U}(100, 500)$ and $\mathcal{U}(1, 5)$, respectively. In the simulations, we first fix the number of channels and tasks, and then we use the Monte Carlo method with 1000 trials to obtain the average performance of each method. We emphasize that the randomness in these parameters is for test and simulation purposes only; in practice, the parameters would be determined by mission requirements.

The percentage of tasks which have been scheduled versus the total number of tasks is plotted in Fig. 1. The number of channels is fixed to $K = 4$. As expected, the probability that a task is dropped grows as the number of tasks (N) is increased. The EST method does not consider the deadlines of the tasks. Therefore, it is more likely for a task to be dropped using the EST method compared with the other algorithms. The ED method schedules the tasks based on their deadlines and, in terms of schedule viability, performs better than the EST method. The performance of both the EST and ED methods can be improved using task swapping as described in Section III. As seen in Fig. 1, task swapping enhances the EST method more than the ED algorithm. The B&B method, which finds the optimal solution, has the best performance as expected.

The average cost of task delaying and dropping (the objective) as a function of the number of tasks is depicted in Fig. 2. Again, the number of channels is set to $K = 4$. The ED method imposes a lower dropping cost compared with the EST algorithm. On the other hand, the EST method imposes a lower tardiness cost since it is designed to schedule the tasks as soon as they are ready. Which of the EST or ED methods has better performance depends on the problem parameters. For the scenario we have considered, the EST and ED methods have similar overall costs. The task swapping technique improves both algorithms with more influence on the EST method. Again, as expected, the B&B method has the best performance and shows a substantial improvement over the heuristics. Specifically, the average cost for $N = 40$ tasks is about half of the best heuristic method (EST with task swapping).

Fig. 3 shows the percentage of scheduled tasks versus the number of channels (timelines). The number of tasks is fixed to $N = 30$. As seen in Fig. 3, the performance of the heuristic methods is close to each other, and the B&B method has the best performance. Importantly, if only one channel is available, a high percentage of tasks are dropped. The percentage of scheduled tasks grows as more channels are used to carry on the tasks with substantial improvements from using just $K = 2$ channels.

This analysis is consistent with Fig. 4 which plots the average cost of task delaying and dropping as a function of the number of channels. The number of tasks is set to $N = 30$. As can be seen, the average cost drops dramatically when using $N = 2$ channels and further reduces as more channels are used for task scheduling. However, as with the previous figures, the greatest gains are by providing a second channel.

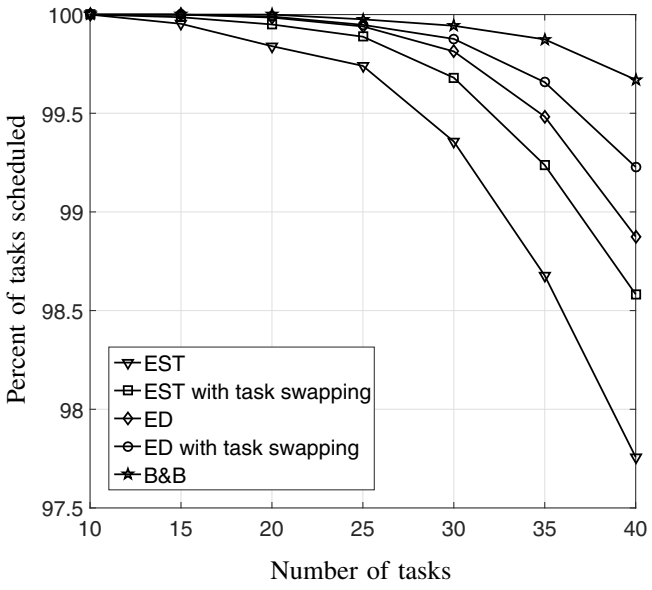

Fig. 1. Percent of tasks scheduled versus the number of tasks. The number of channels is set to 4.

The optimal solution can be obtained using the B&B method. However, the complexity of the B&B algorithm becomes prohibitive for large-size problems. Furthermore, the run-time of the B&B method has a heavy-tail distribution, which means that in some cases, the run-time can be substantially larger than its average value. Fig. 5 shows the histogram of the simulation time of the B&B algorithm for a problem with $N = 40$ tasks and $K = 4$ channels. The run-time of the B&B method has been measured for 1000 instances to produce this histogram. As can be seen, most of the instances have a run-time which is less than a minute, but there are some cases which can take as long as about 130 minutes. Consequently, the B&B method may not be suitable for real-time problems. However, it can be used offline to obtain the optimal solution which can be useful in assessing the performance of suboptimal solutions. Furthermore, it can be used to generate data to train a cognitive scheduler that is able to extend results from a small number of tasks (small N) to realistic values of N (such as N in the range of 50-100).

VI. Conclusion

In this paper, we consider the problem of task selection and scheduling for multifunction multichannel radars. Building on previous wroks, suboptimal solutions are obtained by splitting the problem into two steps of task selection and scheduling. In the selection step, tasks with higher dropping costs are prioritized. Two heuristics known as the earliest starting time (EST) first and earliest deadline (ED) first methods are used in the scheduling phase. The main contribution of this paper is developing the branch-and-bound (B&B) technique to obtain the optimal solution of the joint problem. Although the B&B significantly outperforms the suboptimal methods, it is not applicable to large-size and real-time problems, as it has high computational burden. In our implementation we used dominance rules to prune as many branches as possible. Such

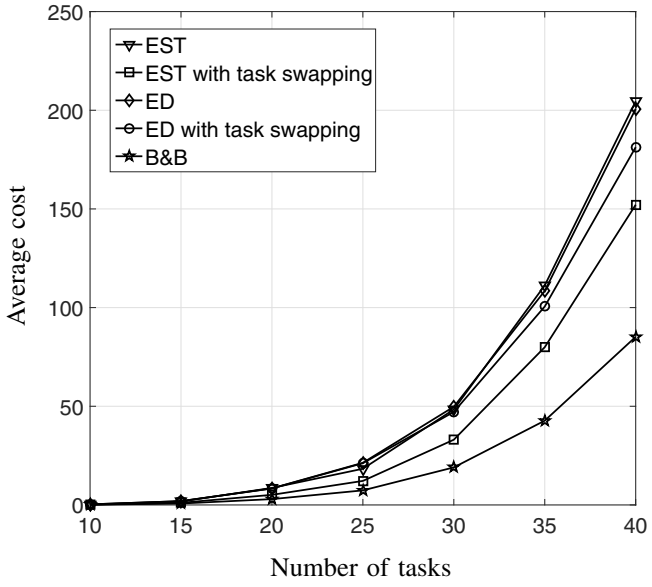

Fig. 2. Average cost versus the number of tasks. The number of channels is set to 4.

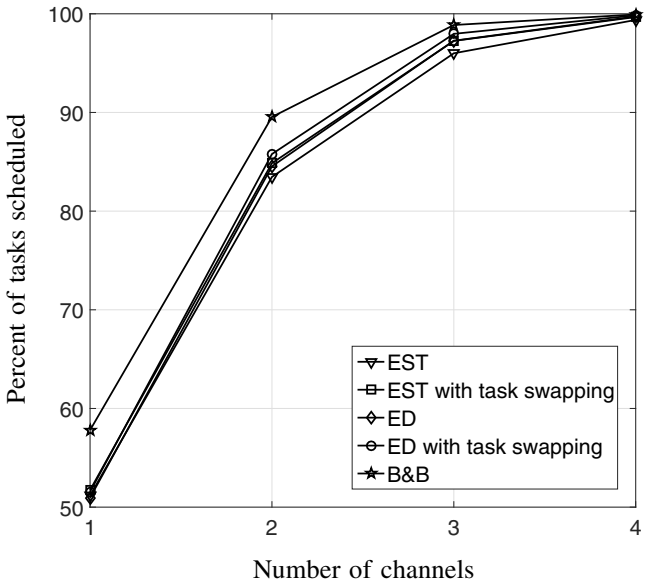

Fig. 3. Percent of tasks scheduled versus the number of channels. The number of tasks is set to 30.

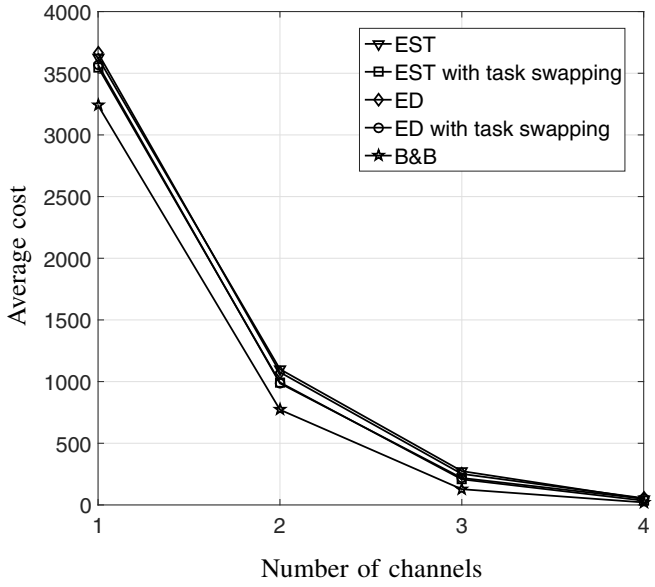

Fig. 4. Average cost versus the number of channels. The number of tasks is set to 30.

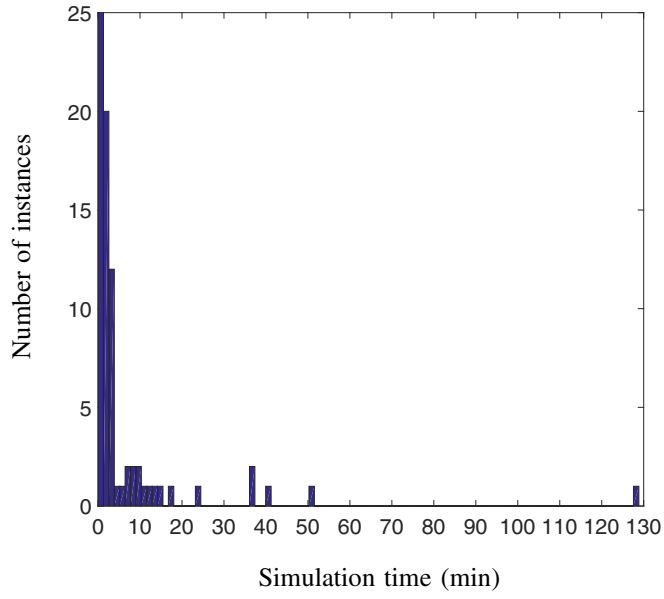

Fig. 5. Distribution of the simulation time obtained from 1000 runs of the B&B method. The number of tasks is set to 40. There are 949 instances in the first bin which has been cut in the figure.

rules help to eliminate nodes of the search tree, and therefore, lead to faster convergence to the optimal solution.

Although the B&B method may not be suitable for real-time applications, it can be used to benchmark heuristic algorithms. However, importantly, we can obtain the optimal schedule of a given problem offline. Then, with machine learning (ML) techniques, such optimal solutions can be used to train a neural net that achieves performance close to the optimal solution with low complexity. The ML (or cognitive) methods can be used to learn patterns of *good* schedules and can, ideally, parse a task list and determine a close-to-optimal schedule with low complexity.

The challenges of obtaining such a cognitive scheduler is that the depth of the neural net may need to be quite large, so that it can capture the patterns of optimal schedules. This will in turn require a large number of samples to train the neural net. These problems can be potentially mitigated by proper design of the scheduler. As an example, instead of a single neural net, we can consider a scheduler with multiple interconnected networks. The design of such networks is open for future work.

ACKNOWLEDGMENT

This work is supported by Raytheon Canada, the Natural Sciences and Engineering Research Council (NSERC) of Canada, and Defence Research and Development Canada (DRDC).

REFERENCES

[1] P. W. Moo and Z. Ding, *Adaptive Radar Resource Management*. Academic Press, 2015.

[2] A. Charlish, K. Woodbridge, and H. Griffiths, "Phased array radar resource management using continuous double auction," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 51, no. 3, pp. 2212–2224, Jul. 2015.

[3] S. L. C. Miranda, C. J. Baker, K. Woodbridge, and H. D. Griffiths, "Comparison of scheduling algorithms for multifunction radar," *IET Radar Sonar Navig.*, vol. 1, no. 6, pp. 414–424, Dec. 2007.

[4] A. M. Ponsford, L. Sevgi, and H. C. Chan, "An integrated maritime surveillance system based on high-frequency surface-wave radars, Part 2: Operational status and system performance," *IEEE Antennas Propag. Mag.*, vol. 43, no. 5, pp. 52–63, Oct. 2001.

[5] S. P. Noyes, "Calculation of next time for track update in the MESAR phased array radar," in *IEE Colloquium on target tracking and data fusion (digest no. 1998/282)*, Jun. 1998, pp. 2/1–2/7.

[6] A. Sinha, Z. J. Ding, T. Kirubarajan, and M. Farooq, "Track quality based multitarget tracking algorithm," in *Proc. SPIE 6236, Signal and Data Processing of Small Targets 2006, 623609*, May 2006.

[7] J. Yan, H. Liu, B. Jiu, B. Chen, Z. Liu, and Z. Bao, "Simultaneous multibeam resource allocation scheme for multiple target tracking," *IEEE Trans. on Sig. Proc.*, vol. 63, no. 12, pp. 3110–3122, June 2015.

[8] H. S. Mir and A. Guitouni, "Variable dwell time task scheduling for multifunction radar," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 463–472, Apr. 2014.

[9] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[10] C. A. Yano and Y. Kim, "Algorithms for a class of single-machine weighted tardiness and earliness problems," *Eur. J. Oper. Res.*, vol. 52, no. 2, pp. 167–178, 1991.

[11] A. Jouglet, P. Baptiste, and J. Carlier, "Branch-and-bound algorithms for total weighted tardiness," in *Handbook of scheduling: Algorithms, models, and performance analysis*. CRC Press, 2004.

[12] A. Jouglet and D. Savourey, "Dominance rules for the parallel machine total weighted tardiness scheduling problem with release dates," *Comput. Oper. Res.*, vol. 38, no. 9, pp. 1259 – 1266, 2011.

[13] F. Yalaoui and C. Chu, "New exact method to solve the $Pm/r_j/\sum C_j$ schedule problem," *International Journal of Production Economics*, vol. 100, no. 1, pp. 168 – 179, 2006.