A Radar Task Scheduling Method Using Random Shifted Start Time with the EST Algorithm

Zhen Qu, Zhen Ding, and Peter Moo
Defence Research and Development Canada
DRDC Ottawa Research Centre
Ottawa, Ontario, K1A 0Z4, Canada
zhen.qu@drdc-rddc.gc.ca

***Abstract*—A radar task scheduling method is proposed by using the Monte Carlo simulation with the earliest start time (EST) algorithm. The method gets an initial solution by using the EST. Then multiple Monte Carlo simulations are used for more solutions. At each Monte Carlo simulation, the start time of each task is randomly shifted in its schedulable time window, and a new solution is achieved by using the EST again. Among all the solutions, the one with the minimal cost is the final solution. The performance of the proposed method is evaluated numerically. When 1000 Monte Carlo simulations are used, the proposed method's cost is about 1.4 to 9.1 times less, depending on the scenario, than the EST. The proposed method is also very efficient and therefore practical. A full cycle of scheduling takes only a few tens of milliseconds.**

Keywords—multi-function radar; radar resource management; task scheduling; earliest start time; Monte Carlo simulation

I. INTRODUCTION

Phased array radar is widely used in many fields nowadays, such as the aerospace, public transportation, robotics, autonomous driving, military, etc. A radar task can be generally defined as a single activity that a radar system is going to process, including the signal transmission, and the receiving from the remote target signal [1]. The time to start the first signal transmission is defined as a start time (t_{start}) of one task, and then the time consumed in all the transmission/reception cycles is defined as the dwell time (t_{dwell}) of the task. A phased array radar is typically responsible for multiple functions, and each function could also have multiple tasks to be processed [2]–[4]. All the tasks from different functions could have different t_{start}, t_{dwell}, and task priorities (p). One of the important problems is the radar resource management (RRM), which coordinates all the tasks from all functions, such as the selection, prioritization, and scheduling, before the execution of them [2], in order to optimize the radar effectiveness and efficiency. The RRM is required to assign the tasks into a schedulable sequence without conflicts or overlaps between each other.

Practically, when the tasks are passed to the radar scheduler, one or several of them may have conflicts or overlaps with the others, so that one or more of those involved tasks need to be rescheduled at a different time (t_{res}) in order to avoid such conflicts. In some situations, one or more of the tasks need to be dropped so that the remaining tasks can be scheduled in the limited time window. Although time shifts and/or task drops help the radar system run properly, it could also be costly especially when important tasks are dropped or executed at a delayed, or even an earlier time. Thus, to schedule the tasks in an optimized way is always desired. In the meanwhile, the time spent on a scheduling cycle should also be practical, especially for overwhelm situations such as operations in the battlefield. Therefore, scheduling the radar tasks in an effective and efficient way is meaningful, and this topic is being studied for decades [5]–[9].

Among different radar missions, the radar system would have different objectives, hence the strategy of task scheduling could be different from one mission to another. Generally, radar task scheduling is a NP-hard problem [10] and, as a result, the complexity and the time consumption to complete one scheduling could be rather high in order to find a global optimal solution. Many scheduling methods were proposed in the literature under specific assumptions, such as the earliest-start-time (EST), the earliest deadline (ED), greedy scheduling, heuristic scheduling, branch-and-bound, and some artificial intelligence (AI) based approaches such as the use of the neural networks (NN) [5], [9], [11]–[14]. Among all, the conventional scheduling methods, for instance, the EST and ED, consume less time, but with poor performance in terms of the cost. In contrast, the AI approaches, for instance the recurrent NN and deep learning, significantly improve the performance of the task scheduling, however the relatively long computational time is their major drawback, especially when the number of the tasks is high. It is acceptable if the training process takes a long time as it can be done offline. But the trained NN shall be fast for real time operations.

Our objective is to find a sub-optimal scheduling solution, which is realistic for real time applications. In other words, we want to schedule the tasks within a short time, a minimal drop rate and a minimal cost. In this paper, we propose a radar task scheduling method which consists of the Monte Carlo simulation and the EST scheduling algorithm.

II. PROBLEM FORMULATION

In this paper, we consider N tasks, to be scheduled within a defined time window with a length L. Assume that the radar system is ideally designed for handling N tasks, then the averaged t_{dwell} of these tasks should be L/N to make the total

978-1-7281-1679-2/19/$31.00 ©2019 Crown

t_{dwell} of all the tasks equal to L. To generalize the problem in this paper, a normalized time window is used, i.e. $L = 1$, and hence the ideal averaged $t_{dwell} = 1/N$. Note that for scheduling in a practical mission, the actual number of the input tasks (N_{actual}) is not necessary to be the same as N as it was designed, for instance, in practice the scheduler could be under an under loaded situation if $N_{actual} < N$, or over loaded if $N_{actual} > N$.

In this study, it is assumed that five parameters of a task are known, including the aforementioned t_{start}, t_{dwell}, and p. Two other input parameters are the earliest start time ($t_{earliest}$) and the latest start time (t_{latest}) that a given task is allowed to be executed [2]. In a mission, all the tasks are firstly passed to the scheduler for a scheduling, then the decision (a schedule) is made based on all the received tasks with their 5 parameters, and finally, part or all of the tasks are executed, according to the determined schedule. Note that for the sake of simplicity, we do not consider the task dwell interleaving, i.e. no other tasks will be interleaved with one task within its t_{dwell}.

Once the scheduling is accomplished, a task could suffer from cost due to the delay or advance of its actual execution time (t_{exe}), or the drop of itself. In this paper, a total cost (J) of a scheduled task sequence is defined as the summation of all the individual task cost $C(n)$ in the mean squared error format. The equations are expressed as follows:

$$J = \sum_{n=1}^{N_{actual}} C(n), \tag{1}$$

$$\Delta t = |t_{res}(n) - t_{start}(n)|, \tag{2}$$

$$\tau = |t_{earliest}(n) - t_{start}(n)|, \textbf{ when } t_{res}(n) < t_{start}(n),$$
$$= |t_{latest}(n) - t_{start}(n)|, \textbf{ when } t_{res}(n) \geq t_{start}(n), \tag{3}$$

$$C(n) = \frac{1}{N_{actual}}\left[p(n) \cdot \frac{\Delta t}{\tau}\right]^2,$$
$$\textbf{when } t_{earliest}(n) \leq t_{res}(n) \leq t_{latest}(n),$$
$$= \frac{1}{N_{actual}}\left[p(n) \cdot C_{dp}\right]^2, \textbf{ otherwise,} \tag{4}$$

where Δt is the time difference between the original t_{start} and the rescheduled t_{res}. τ is the time difference between either the $t_{earliest}$ or t_{latest} and the original t_{start}. C_{dp} is a scalar that represents the task drop penalty.

When the t_{res} is within the schedulable time window [$t_{earliest}$, t_{latest}], the ratio $\Delta t/\tau$ in Eq. (4) reflects the percentage of the t_{res} deviating to the original t_{start}. In this study, we consider that a task will have no cost if its t_{res} is the same as the original input t_{start}, otherwise the cost will be increased if the t_{res} is away from the original t_{start} until it reaches the $t_{earliest}$ or t_{latest}. Thus, this ratio term results in a number between 0 and 1. If no possible t_{res} can be decided within the schedulable time window, such a task needs to be dropped and a drop penalty C_{dp} will be applied to give a great cost to Eq. (1) (because dropping a task is not desired). Either the ratio term $\Delta t/\tau$ or the C_{dp} will be multiplied by the task's priority $p(n)$. As a result, a task with higher priority will cost more than that with a lower priority, if they have the same Δt.

In this paper, $p(n)$ is a random number from 0.1 to 0.9 for each task, and the C_{dp} is always assigned to 10. Since the ratio term $\Delta t/\tau$ is between 0 (no time difference, $t_{res} = t_{start}$) and 1 (task is moved to its most schedulable time, $t_{earliest}$ or t_{latest}), the product of p and the ratio term $\Delta t/\tau$ will be between 0 (the least prior task) and 0.9 (the most prior task), when a task is executed. On the other hand, when a task is dropped, the product of p and C_{dp} will between 1 (the least prior task) and 9 (the most prior task). As one can see that even the least prior task will cost more if it is dropped, than the most prior task scheduled at $t_{earliest}$ or t_{latest}.

III. Proposed Scheduling Method

We propose a task scheduling method, which uses the Monte Carlo simulations with the EST scheduling algorithm. The proposed method aims to find a sub-optimal solution of the task schedule within a relatively short time, makes it practical for real time applications.

The EST algorithm is a conventional method [15], which is relatively simple and the computational time is rather short, even the number of given tasks is high. Thus, scheduling tasks using the EST algorithm is fast enough for most problems.

Principally, the EST scheduling algorithm places one task right after an earlier one, i.e. $t_{exe}(n+1) = t_{exe}(n) + t_{dwell}(n)$. Therefore, the merit of this algorithm maximizes the occupancy in the time window, so that no or only few time gaps will be found in between the executed tasks. It is also noticed that, the original sequence of all the tasks, i.e. the sequence of all the t_{start}, would be the most important factor to control the overall cost in the EST algorithm, since the algorithm does not consider the task priority, it only processes the tasks in a first-come-first-serve role.

In practice, for a under loaded scenario ($N_{actual} < N$), the EST scheduling has less chance to drop tasks since the time window could be long enough to fill all the tasks. Whereas for an over loaded scenario ($N_{actual} > N$), many tasks with later t_{start} are possibly to be dropped despite of their priorities p, because the time window is possibly not long enough for filling all the tasks in, i.e. $\sum t_{dwell} > L$. Therefore, the total cost could be rather high if many tasks with high priority are being dropped.

For any given task, there is an allocated schedulable time window [$t_{earliest}$, t_{latest}], hence adding a random shift to the original t_{start} of every single task (resulting t_{shift}), a new time sequence will be formed. Consequently, this new sequence of t_{shift} may possibly yield a different total cost J from that calculated by using the original sequence of t_{start}. In principle, there must be a sequence of t_{shift} that yields a global minimal cost (J_{min}) using the EST scheduling algorithm. However, depending on the value of N_{actual}, the possible combinations of all the task sequence could be a rather huge number, which is practically impossible to find the solution. Therefore, statistically, Monte Carlo simulations could be used to efficiently find a sub-optimal solution instead of a global optimal solution [16], [17]. In our case, Monte Carlo simulation will first randomly generate a t_{shift} (within the allowed time range), termed random shifted start time (RSST), to every single task, then send all the RSST tasks to conduct the EST scheduling and calculate the total cost J. By repeating these steps K times, K solutions (i.e. K different resulting schedules) and their corresponding total costs J will be obtained. Finally, the schedule that resulting the J_{min} among all

978-1-7281-1679-2/19/$31.00 ©2019 Crown

will be considered as the final solution, and the tasks will be executed according to that schedule. Such a method is termed RSST-EST scheduling method, and depicted in Table 1:

TABLE 1. RSST-EST Scheduling Method

1. Conduct ***EST Scheduling Algorithm*** using t_{start}
2. $J_{min} = J$
   $t_{exe} = t_{res}$
3. **For** $k = 2: K$
4. $A = 0$ **IF** $t_{earliest} < 0$
   $A = t_{earliest}$ **IF** $t_{earliest} >= 0$
   $B = 1$ **IF** $t_{latest} > 1$
   $B = t_{latest}$ **IF** $t_{latest} <= 1$
   t_{shift} = **RANDOM** (between A and B)
5. Conduct ***EST Scheduling Algorithm*** using t_{shift}
6. **If** $J < J_{min}$
   $J_{min} = J$
   $t_{exe} = t_{res}$
   End
7. **End**

In the RSST-EST method, we assign that $t_{shift} = t_{start}$ in the first iteration ($k = 1$, i.e. the procedure 1 in Table 1) so that the proposed method is equivalent to the EST scheduling algorithm in its first iteration. Therefore, it guarantees that the proposed method at least has the same performance as the EST, even though in the case that the K-time Monte Carlo simulations does not find a better solution of tasks schedule, than that from the EST algorithm using original t_{start}.

It is worth noting that, although the procedure 1 and 5 in Table 1 are done by the EST algorithm in this paper, the main factor that improves the scheduling performance is the RSST based on the use of Monte Carlo simulation. Therefore, our proposition is not limited by the EST, it could be any other scheduling algorithm as substitution.

IV. Numerical Simulations

Several numerical simulations are conducted based on the Eqs. (1)-(4) and the flow in Table 1. The radar task scheduler is firstly simulated as being designed to handle (N =) 20, 30, 40, and 50 tasks, respectively, so that the ideal averaged t_{dwell} of one task (= $1/N$), is 1/20, 1/30, 1/40, and 1/50, respectively, in each simulation. Then, the total number of task N_{actual} that actually passed to the scheduler is assigned from $0.5N$ to $2N$, which represents a varying loading rate from 50% (under loaded) to 200% (over loaded), with an increment of 10%. For instance, in a simulation run that the scheduler is designed to handle 50 tasks (N), 25 to 100 tasks (N_{actual}) with a 5-task increment, are input to the proposed scheduling method, respectively. The purpose is to evaluate the performance of the RSST-EST scheduling method under different loading situations.

In each simulation, N_{actual} tasks are created. Each task has a t_{dwell} that is randomly generated between 0 and $2/N$, so that the averaged t_{dwell} over all the tasks is approximately equaled to $1/N$. The t_{start} of every task is randomly generated between 0 and $1 - 1/N$. In this paper, τ in Eqs. (3) and (4) are both assigned to 0.5. The priority p of every task is selected as introduced in Section II.

After the creation of a group of tasks, the drop-rate (γ) and total cost J of the proposed RSST-EST method are compared with those obtained by the EST. The tasks are first scheduled by the EST, then the same group of tasks are scheduled by the RSST-EST. By the accomplishment of each method, dropped tasks are counted (termed as $N_{dropped}$) and the J are calculated. A task drop-rate γ is defined as follow:

$$\gamma = \frac{N_{dropped}}{N_{actual}} \times 100\%. \quad (5)$$

Finally, both methods are computed 1000 times in order to find their statistical averages. Fig. 1 and Fig. 2 respectively show the γ, and J, with respect to the task loading rate using the EST (indicated by red-crosses), and the RSST-EST (indicated by blue-circles). Results of the RSST-EST method in Figs. 1 and 2 are all determined from 1000-time Monte Carlo RSST attempts (K = 1000). The result of J is normalized by the maximal cost (defined as the total cost when all tasks are dropped) in each simulation.

Overall Figs. 1 and 2 reflect that the RSST-EST method has better performance than the EST. Both the averaged γ and J of the RSST-EST are less than those of the EST under most of the loading rate situations.

In Fig. 1, all the results with different N values show the same tendencies: both methods experience monochromatic increase of their γ with respect to the loading rate, while γ of the RSST-EST increases slower than that of the EST. At the 200% of the loading rate, the RSST-EST has about 5% less in γ than that of the EST. Note that at 50% and 60% loading rates, neither method drops any task due to the very sufficient time window.

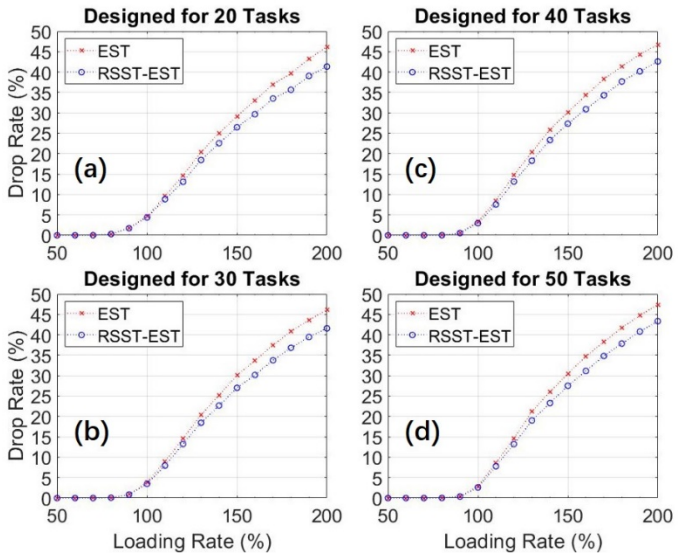

Fig. 1. The comparisons of task drop-rate between the EST (red-crosses) and the RSST-EST (blue-circles) methods with respect to task loading rate, with (a) N = 20, (b) N = 30, (c) N = 40, and (d) N = 50. K = 1000 for all the RSST-EST scheduling.

978-1-7281-1679-2/19/$31.00 ©2019 Crown

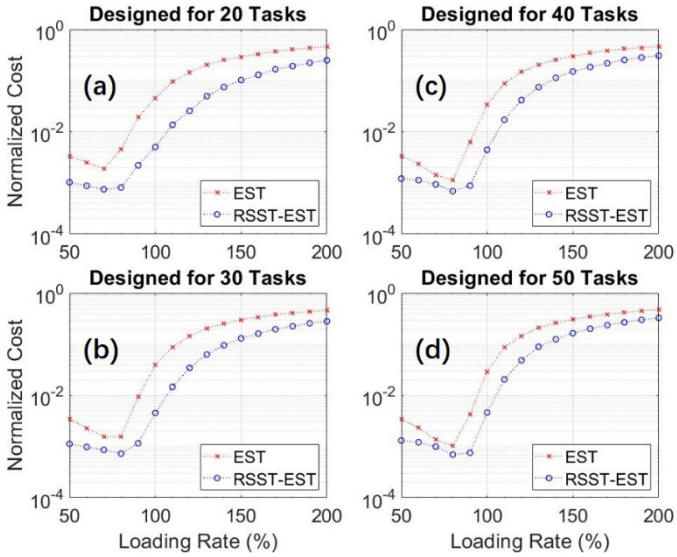

Fig. 2. The comparisons of total cost between the EST (red-crosses) and the RSST-EST (blue-circles) methods with respect to task loading rate, with (a) N = 20, (b) N = 30, (c) N = 40, and (d) N = 50. K = 1000 for all the RSST-EST scheduling.

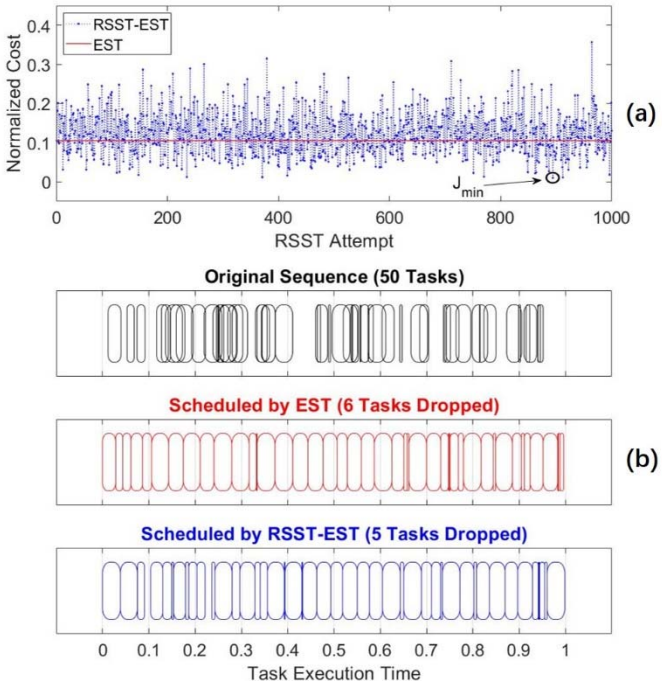

Fig. 3. Task scheduling example with N = 50, N_{actual} = 50, and K = 1000. (a) Calculated total cost J, of the RSST-EST method (blue-dots) and the EST method (red-line), at each random shift attempt. (b) The original task sequence (black), the EST schedule (red), and the RSST-EST schedule with least cost J_{min} (blue).

On the other hand, in Fig. 2, J of both methods are first decreased and reach their minimums at 80% of the loading rates (except for N = 20 where the minimums for both methods are found at the 70% loading rate, and N = 30 where that of the EST is found at the 70% loading rate), then start to increase with respect to the loading rate. Overall the cost of the RSST-EST is always less than that of the EST. Among all the comparisons of J, the least difference between two methods is found at the 70% loading rate when N = 50, where the cost of the RSST-EST is 1.4 times less than that of EST. The greatest difference is found at the 100% loading rate when N = 20, where the cost of the RSST-EST is 9.1 times less than that of the EST. One should notice that, despite of what method is actually used, naturally as higher the loading rate, more tasks have to be dropped from the limited time window, which causes the increases of both γ and J in very high overloading situations.

One example of tasks rescheduling can be seen in Fig. 3. Such simulation assigns N = 50 with the 100% loading rate, and K = 1000. Fig. 3(a) shows the progressive calculation results of J with respect to the RSST attempts, where the RSST-EST method is indicated by blue-dots and the EST method is indicated by red-line. The J_{min} is found at the 894th RSST attempts with a value of (J_{min} =) 0.01, while as a comparison, the total cost of the EST is 0.11. Fig. 3(b) illustrates the facts of the task sequence: the original sequence of t_{start} (plotted with black color), the schedule made by the EST method (red), and the schedule made by the RSST-EST method (blue). In Fig. 3(b), a round bar represents an individual task, the width of the bar represents the task's t_{dwell} and its left edge indicates the t_{start} or t_{exe} of the task. As it can be seen that there are several overlaps among the tasks in the original sequence (top/black) in Fig. 3(b), then the overlaps are both resolved after the EST and the RSST-EST scheduling. The execution sequences of the tasks from the two methods are visibly different, in which 6 tasks are dropped by the EST while such number for the RSST-EST is 5.

According to the principle of the Monte Carlo simulations, more random searches will be helpful to find an even better solution of the problem, hence more RSST attempts (or a higher K value) in our problem to the RSST-EST method should further lower both γ and J (until they reach their global minima), however it also requires more computational time, which may make the method impractical. Other than the value of K, the actual number of the tasks, N_{actual}, also affects the time spend in computation. In principle, the more tasks to be scheduled, the longer the time of computation.

Two groups of simulations at N = 50 are implemented using different K values to investigate the time spend in the task scheduling of the proposed RSST-EST method. The first group is carried out at three RSST attempt numbers, K = 100, 1000, and 10000, respectively, over the loading rate ranged from 50% to 200% (25 to 100 actual tasks). At each loading rate, the time spend is averaged over 1000-time computations and the averaged results with respect to the loading rate are showed in Fig. 4(a). The second group is carried out at the 100% loading rate (N_{actual} = 50), the time spend in the task scheduling is then calculated at K = 100, 500, 1000, 5000, 10000, 50000, and 100000, respectively. Same as the first group, results are averaged over 1000-time computations, and shown in Fig. 4(b), with respect to the K value.

978-1-7281-1679-2/19/$31.00 ©2019 Crown

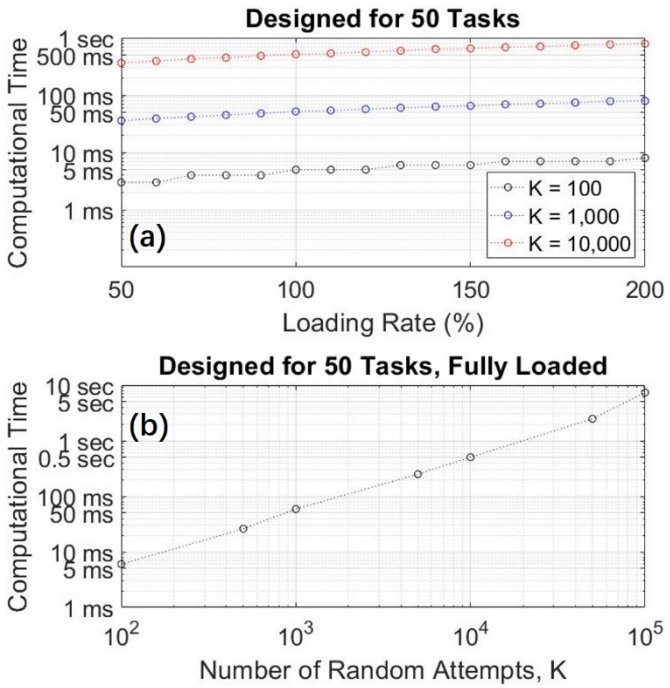

Fig. 4. Computational time of the RSST-EST method (a) with respect to loading rate at K = 100 (indicated by black-circles), 1000 (blue), and 10000 (red); and (b) with respect to K value under 100% loading rate.

It can be seen that the time spend in the RSST-EST is increased with the N_{actual} at every given K value. And the time spend under a certain N_{actual} is also proportional to the K. Approximately, by given 40 to 50 tasks, the RSST-EST method may need about 50 ms to schedule them with 1000 RSST attempts (K = 1000). Overall to schedule a group of 25 to 100 tasks, when using K = 1000 the computational time is ranged between 30 and 90 ms. A further higher K would prolong the computational time greatly. It is worth noting that, when dealing with similar number of tasks, a tree search based approach with machine learning was proposed to find a solution much less costly than the EST, yet such a method needs a few to a few tens of seconds to find the solution [18].

All the simulations and numerical calculations presented in this paper are implemented through MATLAB 2018a in a Windows 10 operation system, the computer is equipped with a 3rd generation Intel i7 CPU, 8 GB RAM, and a traditional hard disk drive (HDD), in which the hardware specifications are somewhat out of date. Besides, it is also known that the running speed of some alternative software such as C/C++ and Python could be much faster than MATLAB. Therefore, the proposed RSST-EST method could be principally more time efficient than the numerical results presented here, when a better and up-to-date software/hardware system is equipped.

V. Conclusion

In this paper, we proposed a radar task scheduling method based on using the random shifted start time (RSST) and the EST algorithm (RSST-EST). The Monte Carlo simulation is used to generate the RSST for the tasks then they are scheduled by the EST algorithm. The RSST-EST procedures are repeated many times to find a solution with the lowest cost. The performance of the RSST-EST method has been evaluated, with a cost improvement of 1.4 to 9.1 times to the EST algorithm. The proposed method is very efficient under the fully or over loaded situations. In principle the RSST could be combined with any other scheduling algorithm such as the ED. The RSST-EST method requires 30 to 90 ms in our computer system, to find a scheduling solution from 1000-time Monte Carlo simulations. The number of Monte Carlo simulation runs could be adjusted so that the computation time is within control.

References

[1] M. I. Skolnik, *Radar handbook*. London: McGraw Hill, 1990.

[2] P. Moo and Z. Ding, *Adaptive radar resource management*. Academic Press, 2015.

[3] D. R. Billetter, *Multifunction array radar*. Norwood, MA: Artech House, 1989.

[4] S. Sabatini and M. Tarantino, *Multifunction array radar- system design and analysis*. Norwood, MA: Artech House, 1994.

[5] A. J. Orman, C. N. Potts, A. K. Shahani, and A. R. Moore, "Scheduling for a multifunction phased array radar system," *Eur. J. Oper. Res.*, vol. 90, no. 1, pp. 13–25, 1996.

[6] 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, 2007.

[7] 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, 2014.

[8] M. Shaghaghi and R. S. Adve, "Task selection and scheduling in multifunction multichannel radars," *IEEE Radar Conf.*, 2017, pp. 0969–0974.

[9] C. Duron and J.-M. Proth, "Multifunction radar: task scheduling," *J. Math. Model. Algorithms*, vol. 1, no. 2, pp. 105–116, 2002.

[10] J. K. Lenstra, A. H. G. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Ann. Discret. Math.*, vol. 1, pp. 343–362, 1977.

[11] A. Izquierdo-Fuente and J. R. Casar-Corredera, "Optimal radar pulse scheduling using a neural network," *IEEE Int. Conf. Neural Networks*, 1994, vol. 7, pp. 4588–4591.

[12] S. L. C. Miranda, C. J. Baker, K. Woodbridge, and H. D. Griffiths, "Fuzzy logic approach for prioritisation of radar tasks and sectors of surveillance in multifunction radar," *IET Radar, Sonar Navig.*, vol. 1, no. 2, pp. 131–141, 2007.

[13] S. Ghosh, R. Rajkumar, J. Hansen, and J. Lehoczky, "Integrated QoS-aware resource management and scheduling with multi-resource constraints," *Real-Time Syst.*, vol. 33, no. 1–3, pp. 7–46, 2006.

[14] M. Shaghaghi and R. S. Adve, "Machine learning based cognitive radar resource management," *IEEE Radar Conf.*, 2018, pp. 1433–1438.

[15] P. Bratley, M. Florian, and P. Robillard, "Scheduling with earliest start and due date constraints," *Nav. Res. Logist. Q.*, vol. 18, no. 4, pp. 511–519, 1971.

[16] Z. B. Zabinsky, "Random Search Algorithms," *Wiley Encyclopedia of Operations Research and Management Science*. 2011.

[17] S. Chantaravarapan, a. Gunal, and E. J. Williams, "On Using Monte Carlo methods for scheduling," *Winter Simul. Conf. 2004.*, vol. 2, pp. 789–794.

[18] M. Shaghaghi, R. S. Adve, and Z. Ding, "Multifunction cognitive radar task scheduling using Monte Carlo tree search and policy networks," *IET Radar, Sonar Navig.*, vol. 12, no. 12, pp. 1437–1447, 2018.

978-1-7281-1679-2/19/$31.00 ©2019 Crown