2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Macau, China, November 4-8, 2019

Cannot avoid penalty for fluctuating order arrival rate? Let's minimize

Marichi Agarwal and Chayan Sarkar

Abstract— **Warehouse management system assigns a preferred completion time for every order based on the customer profile and the good(s) that is/are ordered. Even though employing multi-robot systems to manage goods movement bring operational efficiency in a warehouse, it is difficult to meet these soft deadlines of tasks in the peak hours/seasons. This can impact the respective businesses significantly as the lateness of task completion incurs a direct/indirect penalty. In this work, we develop an online task scheduling algorithm for such a multi-robot system, called *Online Minimum Penalty Scheduling* (OMPS). Though there exists a large number of multi-robot task scheduling algorithms, they are not suitable (or less efficient) for a system where each task has a soft deadline and accumulates penalty if it is executed beyond its deadline. Moreover, the lack of knowledge of future tasks (online scheduling) makes task allocation a much more difficult job. OMPS provides a robust, scalable, and near-optimal online task schedule. By comparing with the state-of-the-art algorithm, we show that OMPS attracts up to 78% less penalty when a significant number of tasks are bound to miss the deadline. Additionally, it achieves a competitive ratio of up to 1 when compared with a state-of-the-art offline task scheduling algorithm.**

I. Introduction

Efficient warehouse management system (WMS) ensures fluent delivery of the ordered goods to the respective customers and impacts modern-day businesses significantly. Distributors and e-retailers generally provide an estimated-time-of-arrival (ETA) for any goods delivery and pledge for fast and time-bounded delivery as a value-added service. Though lateness in delivery does not have a catastrophic outcome, it impacts the businesses quite significantly in the short and/or long term. The time of out-flowing the goods from the warehouse, even though is a small fraction of the entire delivery process, plays the role of a steer and thus the most significant in the chain. Naturally, the lateness in goods out-flow must be minimized to the extent possible.

Motivation. Goods movement is estimated to be as much as 55% of the total operating expense of a warehouse [6]. Modern warehouses employ robots for goods movement within a warehouse to minimize ergonomic hazard for the human workers along with achieving higher operational efficiency. This is a classical use-case of multi-robot task allocation (MRTA) where a task for a robot is to bring an object from its storage location to the packaging dock for shipment. Though there exists a large number of existing MRTA techniques [15], [5], [19], generally, they do not consider the specific requirements of a warehouse operation. For example, in typical MRTA scenarios, deadlines are not associated with the tasks, which is generally there in warehouse tasks. Though classical real-time task scheduling handles such scenarios, tasks are discarded when they cannot be executed within their deadline **(hard deadline)**. On the other hand, in a warehouse, all the tasks must be completed to avoid customer abandoning. As a result, MRTA for such a situation must complete all the tasks, sooner or later **(soft deadline)**. A penalty is imposed on the system if a task is finished beyond its defined deadline. The problem magnifies when tasks are not known in advance, i.e., their characteristics are known only when a task is introduced to the system (in other words, ordered by the customer).

Problem description. In this paper, we address the problem of assigning atomic tasks to the team of homogeneous robots on-the-fly such that a minimum penalty is incurred on the system by the tasks failing to meet their deadline. The tasks are non-clairvoyant and each task (t_i) is characterized by an arrival time (a_i), an execution time (e_i), a deadline (d_i) and a penalty value (p_i). If a task misses its deadline and completes execution at time (c_i), it incurs a penalty of $(c_i - d_i) * p_i$ units. Please note when the task in-flow is moderate and there is a sufficient number of resources (robots) to complete the tasks within their deadlines, existing task allocation algorithm works quite efficiently. The scope of this work is when the task in-flow rate fluctuates significantly such that a large number of tasks are bound to miss the deadline given a fixed number of resources.

Approach. Real-time task scheduling in a multi-processor system is known to be an NP-Hard problem in a strong sense [9]. Thus, we develop a heuristic algorithm, called "**O**nline **M**inimum **P**enalty **S**cheduling (OMPS)" that aims at minimizing penalty for tasks' execution that appear on-the-fly. Since all the information about the environment is available at the WMS, we design a centralized algorithm. The tasks arriving in the system are initially queued in a global queue before allocating them to the robots. The algorithm works by segregating tasks into two categories, first, tasks that are schedulable within their deadline; i.e., tasks that have not violated the deadline constraint yet, and second, tasks that have already missed their deadline. For the first category, the goal is to maximize the number of tasks with a higher priority value (equivalent to penalty) to meet their deadline. For the tasks that could not be scheduled within their deadline, the scheduling aim is to keep the incurred penalty as least as possible. To avoid starvation of lower priority tasks due to the constant arrival of higher priority tasks, a starvation prevention mechanism is injected which upgrades the priority value.

M. Agarwal and C. Sarkar are with the Embedded Systems & Robotics division of TCS Research & Innovation, India {marichi.agarwal, sarkar.chayan}@tcs.com.

The major contributions of this work are summarized as follows.

- To best of our knowledge, OMPS is the first online algorithm for MRTA that considers deadline and penalty while scheduling tasks to the robots with the goal of minimizing the overall penalty.
- OMPS also maximizes task scheduling within the deadline and achieves a competitive ratio of up to 1 as compared to the state-of-the-art.
- OMPS guarantees bounded lateness in task execution when tasks cannot be scheduled within their deadline. Thus, it avoids starvation of a task.

II. Related Work

Multi-robot task allocation (MRTA) has been extensively studied by bibliophiles. However, the generic MRTA systems do not cover the specific requirements of a warehouse scenario. Thus, several research activities specifically focused on the warehouse. Here, we discuss some of the relevant works to provide the context of our work and highlight the limitations of the state-of-the-art.

MRTA in Warehouse: Korsah *et al.* [13] provide a taxonomy of multi-robot task allocation. According to this taxonomy, task allocation in a warehouse cannot be instantaneous, but time-extended since the number of tasks is generally much higher than the number of available robots at any point in time. Henn *et al.* [11] presented work on order picking and moving in a warehouse that considers static order list, i.e., all the tasks are known beforehand. A similar task scheduling problem is addressed in [16], where they used an auction-based approach. Recently, Sarkar *et al.* [19] proposed a time extended task allocation mechanism for a warehouse scenario that is very efficient in terms of reducing the overall cost. But, they propose an offline algorithm and also the tasks do not have any deadline associated with them.

Task allocation problem, described in [12], however, deals with the problem with time constraints, i.e., the tasks are associated with a deadline. They applied a Monte Carlo tree search (MCTS) mechanism, which is suitable only for static task set. They do not allow re-planning for dynamically changing task list. A formal framework, spatial task allocation problems (SPATAPs) was introduced in [4] that describes how a team of agents interacts with a dynamically changing set of tasks. SPATAPs allow replanning of tasks at every time step. They assume that the distribution of the orders is known. In this approach, robots may position themselves at locations where the high-priority task likely appears. However, this method would have a negative impact if the distribution of the order type does not follow any pattern.

Online task scheduling: There has been extensive research on online task scheduling on multiprocessor systems [21]. List Scheduling (LS) proposed by Graham *et al.* [20] is known to be the simplest and best classical online scheduling algorithm on parallel machines. LS assigns the current job to the machine that can complete it at the earliest. Motivated by this, Li *et al.* [14] proposed a scheduling algorithm for a semi-online model with relaxed or partial task characteristic known in advance.

MURDOCH [10] is one of the early methods of dynamic task allocation for a group of robots. It is an auction-based task allocation system that is known to be 3-competitive with respect to the optimal off-line solution. Visalakshi *et al.* [23] proposed an improved algorithm based on Particle Swarm Optimization. More recently, Turner *et al.* [22] proposed consensus-based task allocation algorithms (CBBA) that integrates learning with decision making.

Most of these works try to maximize profit by executing a job. They either assume schedulable task set or tasks are dropped if they cannot be scheduled within their deadline. To induce fairness, a penalty value is introduced if a task is dropped. There are different penalty models, e.g., preemption penalty [2] where a penalty is paid each time a task is preempted, non-completion penalty [8] where a penalty is imposed if the task cannot be completed within its deadline, etc. The objective of the existing works is to maximize the system profit by rejecting the tasks as early as possible that cannot meet the deadline constraint. But, in a warehouse scenario, this is not an option as all the tasks must be completed (sooner or later) and cannot be discarded. In our earlier work, we have done non-preemptive task scheduling in a warehouse that minimizes the penalty for the tasks that misses their deadline [17]. However, the algorithm was designed for wave-based task scheduling, i.e., characteristics of all the tasks are known beforehand; thus, not suitable for an online task scheduling scenario.

III. Problem of Online Task Allocation

In online task allocation, characteristics of the future tasks are not known to the system until they are introduced to the system (in other words, ordered by the customer). In the warehouse, where goods are systematically arranged across a large area, the task of a robot is to bring objects from the storage racks to the packaging dock before shipment to the customer. The time required by the robot to bring the object to the packaging dock is the total time it takes to pick the object (actuating time) and to-and-fro travel time between the packaging dock and shall be referred to as the task *execution time*. Every task has a time allocated within which it is expected to be completed and we call it the task *deadline*.

A. Task model

A task t_i is characterized by a four-tuple (a_i, e_i, d_i, p_i) – an arrival time a_i, an execution time e_i, a deadline d_i, and a penalty p_i. Assuming the system (re)starts at time zero, a_i is the time unit after which task t_i arrives at the system and must occupy e_i units of time on a robot. If it starts at time $s_i(\geq a_i)$, it finishes its execution at $s_i + e_i$ time unit. Now, the task shall impose a penalty of p_i per unit time delay in completion beyond the deadline. The penalty function is defined as,

$$P(t_i) = max(0, (c_i - d_i) * p_i), \tag{1}$$

where c_i represents the completion time for task t_i, i.e., $c_i = s_i + e_i$. This signifies that if a task is executed within the deadline, there is no penalty and after that, the penalty increases with time.

B. System model

When an order is placed and a new task is added into the system, the scheduler has to map it to a robot and a time slot when it can be best executed, without any cognizance of the future tasks. Had the release of all the tasks been the same or known before the scheduler begins to assign tasks, it would have been the classical problem of offline task scheduling.

Though we use a generic setup of a warehouse to find a task schedule, the system runs with the following assumptions.

- We assume when a task arrives, the warehouse management system calculates its execution time based on the storage location of the respective goods, assign a deadline based on the overall expected-time-of-arrival pledged to the customer, and assign a penalty value based on customer profile and type of object.
- At any time instant, there are n tasks available in the system that need to be completed by m homogeneous robots. The value of n (and m also) varies significantly over time.
- If a customer places an order for multiple goods, picking of each good is treated as a separate task. In this case, each task can have different execution time, deadline, and penalty.
- All the tasks are assumed to be independent, i.e., completion of one task does not depend on the completion of other(s).
- Every task is atomic and non-preemptive. The robot must execute the task until completion once it is started. Jobs cannot be executed concurrently by more than one robot. We assume that the tasks are independent, i.e, the execution of one task is not contingent on the execution of any other task.

C. Problem formulation

At certain points of time, the number of tasks in a warehouse can be significantly larger than the number of deployed robots. This may lead to a scenario where some/many tasks missing out on being served within their deadline. Since the warehouse promises to complete all the tasks, it incurs a penalty on the system proportional to the delay. The objective hence converges to minimizing the overall loss of the system.

Let's assume, at any time instant, there are m identical robots $\{r_1, r_2, ..., r_m\}$ working in parallel to complete a list of n tasks $\{t_1, t_2, ..., t_n\}$, where task t_i arrived at the scheduler at time a_i. The objective function of our scheduler can be expressed in terms of the incurred penalty (defined in Eq. 1).

$$\min \sum_{i=1}^{n} P(t_i). \tag{2}$$

A scheduler partitions the tasks amongst the m robots and creates a local task queue (ordered task set) R_η^j for robot r_j while minimizing the objective function. Please note that by the time these tasks are completed a new set of tasks may arrive. Also, the fleet of robots may change as some of them may deplete their energy or some new robot may join after recharging their battery. Thus, the online scheduling mechanism needs to accommodate these factors.

D. Integer linear program (ILP) formulation

Luo *et al.* [16] proposed an ILP formulation for tasks with unit execution time and deadline constraint. This model neither includes any penalty value associated with the task if it fails to be scheduled within its deadline nor it considers different task arrival time. Thus, we propose a new ILP formulation where the goal is to minimize the overall penalty of the system and accounts for different task arrival. Since ILP cannot be formulated in an online setup, i.e., the task characteristics are not known beforehand, we take a time-window K where n tasks arrive at different time, but known *a priori*. The value of K is set to $a_i^{max} + \lceil \frac{n}{m} \rceil$ such that all the tasks would finish their execution by the m robots; a_i^{max} is the maximum arrival time among all the tasks.

Let, x_{ij}^k be the variable that takes a value 1 if task t_j can be assigned to robot r_i at time instant k; otherwise it is set to 0. Now, the incurred penalty for this schedule would be $max\{0, (k + 1 - d_j) * p_j\}$, where $k + 1$ is the completion time of the task. The problem can be formulated as follows.

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=d_j}^{K} (k + 1 - d_j) * p_j * x_{ij}^k$$

$$\text{s.t.} \sum_{i=1}^{m} \sum_{k=a_j}^{K} x_{ij}^k = 1, \quad j \in \{1, ..., n\} \tag{3}$$

$$\sum_{j=1}^{n} x_{ij}^k \leq 1, \quad i \in \{1, ..., m\}, k \in \{a_j, ..., K\} \tag{4}$$

$$x_{ij}^k \in \{0, 1\}, \quad \forall i, \forall j, k \in \{a_j, ..., K\} \tag{5}$$

where Eq. 3 ensures that every task is assigned only once and to only one robot and Eq. 4 ensures that multiple tasks are not assigned to a robot at the same time instance.

IV. Algorithm Design

In this section, we describe our algorithm, called *online minimum penalty scheduling* (**OMPS**), that does online scheduling for non-clairvoyant, non-preemptive tasks with soft-deadlines on a multi-robot system. We formulate and solve the problem to assign each of the tasks to one of the available robots such that the overall system penalty can be minimized. The main philosophy behind OMPS is two fold – prioritize scheduling the tasks that would incur a higher penalty if delayed beyond its deadline and carefully schedule the delayed tasks in order to minimize the overall system penalty. However, if there is a low penalty task amongst several high penalty tasks in the system, it can be delayed continuously leading to its starvation. This requires that OMPS ensures fairness in task scheduling.

Algorithm 1: Online Minimum Penalty Scheduling (*OMPS*): A heuristic for scheduling non-preemptive task to minimize overall penalty.

Input: global task queue ($\omega(T)$), available robots (R), local task queues for all robots (R_η), critical task marker for all local queues (h).
Output: updated local task queues (R_η).

```
1  Routine TaskScheduling(ω(T), R, R_η, h)
2      ω̄(T) ← φ ;
3      for (j = 1 : |R|) do
4          for (k = h^j : |R_η^j|) do
5              if (R_η^j[k].priority == 0) then
6                  ω̄(T).append(R_η^j[k])
7              else
8                  ω(T).append(R_η^j[k])
9      R_η ← R_η \ R_η[h :];
10     //schedule critical tasks
11     R_η ← R_η + compaction(ω̄(T)) ;
12     R_η ← R_η + listScheduling(ω̄(T)) ;
13     //schedule safe tasks
14     R_η ← R_η + compaction(ω(T)) ;
15     R_η ← R_η + listScheduling(ω(T)) ;
16     //priority update
17     updatePriority(R_η);
18     return R_η;
```

OMPS maintains a *global queue* (ω) where all the newly arrived tasks are pooled. Every robot maintains a local task queue (R_η^j) where the tasks are sequenced in the order in which they are to be executed by that robot. In the global queue, tasks are queued based on their priority value. Tasks with highest penalty value (e.g., *penalty = 10*) are treated as the top priority tasks (*priority = 1*). To prevent starvation of low priority tasks, the unscheduled tasks upgrade their priority until they are executed or they attain a value equal to zero. Such tasks with priority zero are treated as *critical* tasks and must be scheduled without any further delay. *Safe tasks* form the complement of the critical task set. Each local task queue (R_η^j) has a corresponding critical task marker (h_i). Tasks until this marker (from the beginning of the queue) should never be considered for rescheduling.

When one/multiple robot(s) complete(s) executing the current task, either of the following decision is made.

- *No more task in the global queue or in the robot's local queue* – stay idle.
- *No new task has arrived and no significant delay in executing the previous task* – execute the task from the local queue.
- *New task has arrived in the global queue and/or significant delay in executing the previous task* – follow the routine *TaskScheduling*.

Additionally, if a robot proactively reports breakdown or do not report task completion within a given time-frame after starting a task or a new robot joins the fleet, *TaskScheduling* is invoked.

The *TaskScheduling* sub-routine, when invoked, merges all the unexecuted tasks, in the local queues beyond the critical task marker (h_i), with the newly arrived tasks from the global queue. Intuitively, merging tasks and then determining the schedule leads to a lower accumulated penalty. For example, a task with a very high penalty (proportionally high priority) and smaller deadline may arrive later in the system. This necessitates scheduling it with higher urgency. However, the critical tasks (priority value 0) are not considered when rescheduling is done and they remain at the front of the task queues (lines 3-8). Rest of the tasks are reallocated to the robots (concatenated after the critical tasks) using *compaction* and *listScheduling* (lines 9-15). Finally, the priority of all the tasks that are scheduled beyond their deadline (if any) is updated to avoid starvation.

Compaction - *Scheduling tasks within deadline*: This module of OMPS is inspired by task partitioning using Demand Bound Function (DBF) proposed by Fisher *et al.* [7]. To schedule task t_i within its deadline, it is checked whether e_i continuous time units can be found on any robot for the time interval $[a_i, d_i)$; this is termed as compaction check. Though the goal of DBF is to minimize the number of deadline miss, it does not guarantee the least system penalty. Thus, in our previous work [17], we adapted the necessary constrains of DBF for compaction check that ensures minimal system penalty (Eq. 6 and Eq. 7).

$$d_i - \left(\sum_{\substack{t_j \in T(r_k) \\ d_j \leq d_i}} e_j + e_i \right) \geq 0 \tag{6}$$

$$\forall t_j \in T(r_k) : d_j > d_i : d_j - \left(\sum_{\substack{t_l \in T(r_k) \\ d_l \leq d_j}} e_l + e_j \right) \geq 0 \tag{7}$$

Eq. 6 ensures that there is enough space to execute t_i after executing all the tasks that are scheduled on robot r_k and have deadline smaller than that of t_i. Eq. 7 ensures that the tasks that are already scheduled on r_k and have a deadline greater than t_i are still schedulable if we schedule t_i as well.

List scheduling - *Minimizing penalty for late tasks*: If a task t_i is scheduled to begin in the time interval $[a_i, d_i - e_i]$, it shall complete execution within its defined deadline. On the other hand, if it starts execution after $(d_i - e_i)$, it shall complete execution after (d_i) imposing a penalty on the system. For these tasks that are to be scheduled after time $(d_i - e_i)$, are allocated to the robots using list scheduling [20]. In a nutshell, tasks that cannot be completed within their deadline are sorted by non-decreasing priority value (zero being the highest priority). Then, these tasks are appended to the robots' local queue based on a first come, first serve.

Priority update: Arranging the tasks in the global queue by its initial priority value may keep delaying execution of

a lower priority task, steering it towards starvation. To avoid this, a priority update is done for the tasks that are scheduled using list scheduling, i.e, the tasks that would finish execution after their deadline under the current scheduling. To ensure fair scheduling, the priority of these tasks is increased periodically based on a threshold value (th_i), which is initially set to the absolute deadline of the task. If task t_i fails to begin execution within its defined threshold, the threshold is extended by a value equal to its relative deadline and the priority value is decreased by unit step. This ensures that task t_i becomes a critical task (priority = 0) after $priority_i$ iterations. Such critical tasks are immediately assigned to a robot and cannot be replaced by any other task.

Time-complexity analysis

The decision to invoke sub-routine *TaskScheduling* takes constant time. Thus, the time-complexity of OMPS is equivalent to the time complexity of this routine. Let's assume there are n tasks and $m(= |R|)$ robots in the system. The two nested loops in the Algorithm 1 (at the beginning) have $\mathcal{O}(nm)$ complexity. Then, there are calls to compaction and list scheduling in serial. In list scheduling, each task from a sorted list is appended to a robot's task queue who is available at the earliest. This appending takes $\mathcal{O}(nm)$ where sorting of the tasks takes $\mathcal{O}(nlogn)$. Thus, the time complexity of list scheduling is $\mathcal{O}(max\{nm, nlogn\})$.

Now, during compaction, each task is tried to be scheduled on each robot and tested whether it can still be scheduled within its deadline considering there are some tasks already scheduled on the robot. Trying all the task on a single robot takes $\mathcal{O}(n^2)$ time, which leads to $\mathcal{O}(n^2m)$ complexity for the compaction. Thus, the overall complexity of OMPS is equivalent to compaction, which is $\mathcal{O}(n^2m)$.

V. Evaluation

To evaluate the performance of OMPS, several simulation experiments are carried out using a vast range of datasets. We briefly describe the datasets before discussing the results in details.

A. Datasets

We use the capacity-constraint vehicle routing problem (CVRP) datasets [1] for our evaluation, where the depot location is used as the packaging dock location and site locations are used as the storage location of the objects (to be picked by the robots). Due to the paucity of the space, here, we present results for only two large dataset of 500 and 1000 tasks (site locations) from these sets. As mentioned earlier, each task is represented as a tuple, $t_i =< a_i, e_i, d_i, p_i >$, where the execution time (e_i) is set to the to-and-fro travel time of the robot from the packaging dock to the storage location of the object as proposed in [18]. For a given execution time, we consider the relative deadline for each task to vary according to the following strategies. Please note the absolute deadline of a task is a just summation of the arrival time and the relative deadline.

- $d_i = 2e_i$: In this set, the relative deadline of a task is set to twice the execution time of the task. This is an extreme case as a large number of tasks are bound to miss their deadline when the number of tasks is large compared to the number of robots. Though such a scenario would be rare in practical warehouse scenario, this provides a worst-case view of the scheduler.
- $d_i \in [2e_i, 10e_i]$: The relative deadline is set to a random value in the range of $2e_i$ and $10e_i$ (uniformly distributed).
- $d_i \in [5e_i, 10e_i]$: Again, the relative deadline is set using a uniform distribution in the range of $5e_i$ and $10e_i$. This presents a case where a large number of tasks are schedulable within their deadline, if not all.
- $d \leftarrow mix$: To ensure that the scheduler does not favor a particular distribution of deadline, we take a mix of all the previous three datasets with equal probability.

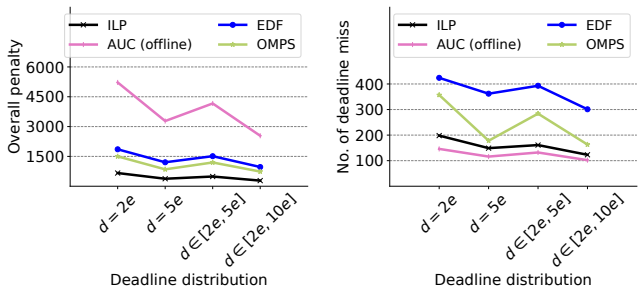
The strategy to associate (relative) deadline is similar to [3], which is a standard procedure to generate data in real-time scheduling domain. Now, for each combination of execution time and deadline, we assign a penalty to each task using 3 different strategies.

- $p \in [1, 10]$: The penalty is set to a random value generated using a uniform distribution in the range of 1 and 10.
- $p \leftarrow same$: The penalty is set to the same value (e.g., 1) for all the tasks.
- $p \leftarrow extreme(1\ or\ 10)$: The penalty is set to the either of the two extreme values, i.e., either 1 or 10 with equal probability.

B. Methodology

We evaluate OMPS based on two metrics – (i) overall penalty incurred while completing all the tasks, and (ii) the number of deadline miss between time τ_0 and τ', where $\tau' > \tau_0$. To validate the performance of the scheduler, we represent a stable system until time τ_0. After time τ_0, a burst of tasks appears in the system until time τ'. The task inflow again reduces after τ'. Two different task arrival pattern during a burst have been graphically shown in Fig. 2 (continuous high inflow rate for a specific time span - *steady high traffic*) and Fig. 3 (too many tasks appear in a tiny time-scale and the pattern continues for some time - *spiking traffic*).

Since task allocation to a multi-robot system is an NP-hard problem, our scheduler adopts a heuristic approach. To scrutinize the optimality of OMPS, we compare it against an optimal solution achieved through the ILP formulation described in Section III-D. An online algorithm is likely to be less efficient in comparison to an equivalent offline algorithm. Thus, the performance of OMPS is further evaluated by comparing the penalty and deadline miss to a state-of-the-art offline *Auction-based algorithm* (AUC in rest of the article) proposed by Luo *et al.* [16]. Though the idea in AUC is to maximize the profit by executing as many tasks as possible within its deadline, we modified the algorithm such that the remaining tasks (that missed their deadline) are scheduled at the end of the schedulable task. This provides the accumulated penalty value for AUC-based scheduling.



(a) Penalty (b) Deadline miss

Fig. 1: Performance comparison for 500 tasks and 10 robots system for unit execution time, four different deadline (relative) distribution and uniform penalty distribution ($p \in [1, 10]$).

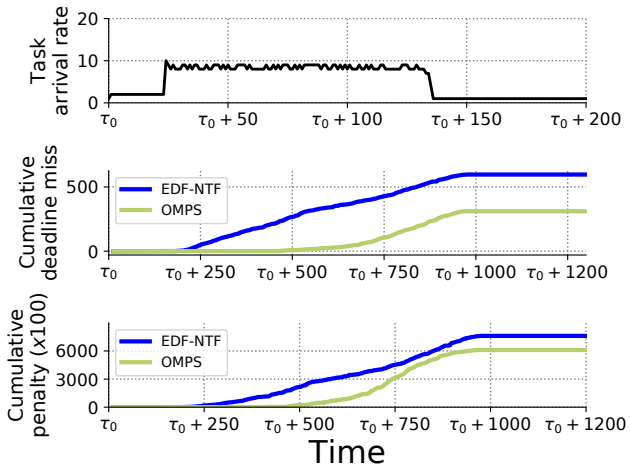


Fig. 2: For tasks arriving between time τ_0 and $\tau_0 + 1200$ (steady high traffic) and 50 active robots, comparison of cumulative deadline miss and cumulative penalty.

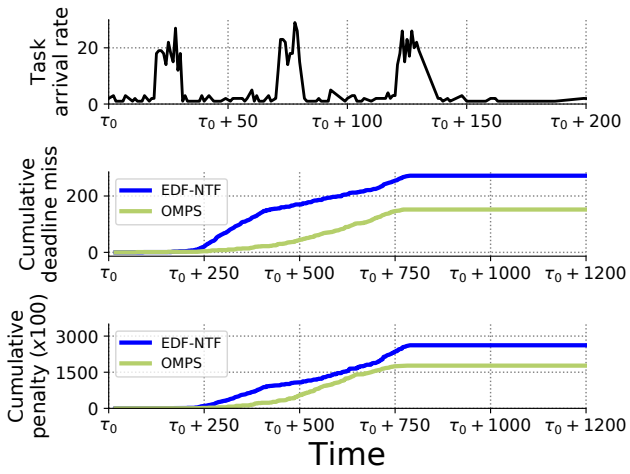


Fig. 3: For tasks arriving between time τ_0 and $\tau_0 + 1200$ (spiking traffic) and 50 active robots, comparison of cumulative deadline miss and cumulative penalty.

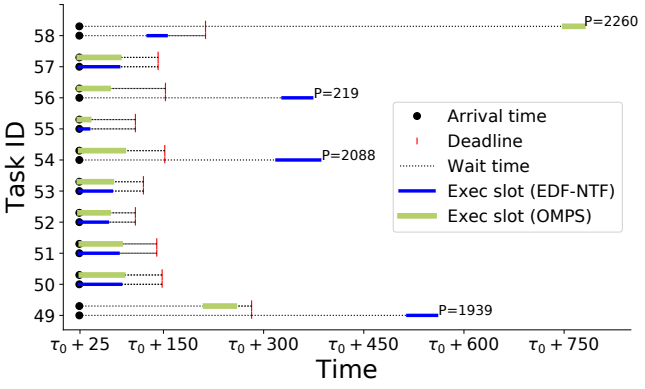


Fig. 4: Execution slot and incurred penalty for tasks arriving at time τ_0+25 units.

To showcase the performance of OMPS with respect to other heuristic based online algorithm, we compare it with a state-of-the-art online task scheduling approach, called earliest deadline first-nearest task first (EDF-NTF) proposed by Turner *et al.* [22].

C. Near Optimality

Fig. 1a and Fig. 1b graphically represents the performance of ILP against *AUC (offline)*, *EDF* and *OMPS*; for 500 tasks (with unit execution time) and 10 robots. Though the ILP yields the optimal solution for the generalized assignment problem, it fails to converge for more than 500 tasks even with unit execution time. We test with four different datasets with different deadline distributions and fixed penalty distribution (uniformly between [1, 10]). Since, $e_i = 1$, EDF-NTF reduces to earliest deadline first (EDF) only. The principle goal is to minimize the penalty in which the ILP outperforms, as expected. Though deadline miss is minimized by AUC, OMPS fulfills the primary goal of minimizing penalty more efficiently over the other two heuristic approaches.

D. Performance under different inflow pattern

The number of tasks missing the deadline and hence the incurred penalty depends on the rate at which the tasks appear in the system and the number of available robots to execute them. We fix the number of active robots to 50 to perceive system stability prior to and post the task burst. Approximately 1000 tasks are added during this high traffic time.

Fig. 2 and Fig. 3 show the number of tasks that missed the deadline and penalty accumulated for the tasks that arrived during a high traffic period. The system was in a stable state before the task explosion occurs. With a fixed number of robots, as the rate of task inflow increases enormously, the system loses stability and a large number of tasks are forced to be executed beyond their deadline. The penalty too gradually starts to accumulate. At the end of the burst, with task arrival rate gradually falling to normal, the system attains stability with no further deadline miss and penalty accumulation. For both kinds of task arrival pattern

TABLE I: Performance comparison for 12 datasets with 1000 tasks and 50 robots.

Penalty	Deadline (relative)	AUC (offline)		EDF-NTF		OMPS		Penalty Reduction (%)		Competitive Ratio (Deadline meet)
		$P(10^2\times)$	*DL-M*	$P(10^2\times)$	*DL-M*	$P(10^2\times)$	*DL-M*	*Over AUC (offline)*	*Over EDF-NTF*	*Over AUC (offline)*
$p \in [1, 10]$	$d_i = 2e_i$	14408	**622**	13126	737	**10482**	699	27.25	20.14	0.79
	$d_i \in [2e_i, 10e_i]$	5343	**197**	5578	405	**3925**	290	26.55	29.64	0.88
	$d_i \in [5e_i, 10e_i]$	2954	**149**	3110	172	**1619**	214	45.17	47.92	0.92
	$d_i \leftarrow mix$	7633	**218**	7608	597	**6098**	311	20.10	19.85	0.88
$p \leftarrow same$	$d_i = 2e_i$	2602	**622**	**2378**	737	2526	624	2.94	-6.23	0.99
	$d_i \in [2e_i, 10e_i]$	966	**197**	1004	405	**918**	205	5.00	8.56	0.99
	$d_i \in [5e_i, 10e_i]$	520	**149**	574	172	**498**	156	4.32	13.29	0.99
	$d_i \leftarrow mix$	1315	218	1370	597	**1212**	**214**	7.77	11.51	1
$p \leftarrow extreme$	$d_i = 2e_i$	13812	**622**	12571	737	**6791**	683	50.83	45.98	0.83
	$d_i \in [2e_i, 10e_i]$	5096	**197**	5381	405	**1172**	243	76.98	78.20	0.94
	$d_i \in [5e_i, 10e_i]$	2419	**149**	2796	172	**686**	181	71.63	75.45	0.96
	$d_i \leftarrow mix$	7086	**218**	7356	597	**2157**	267	69.55	70.67	0.93

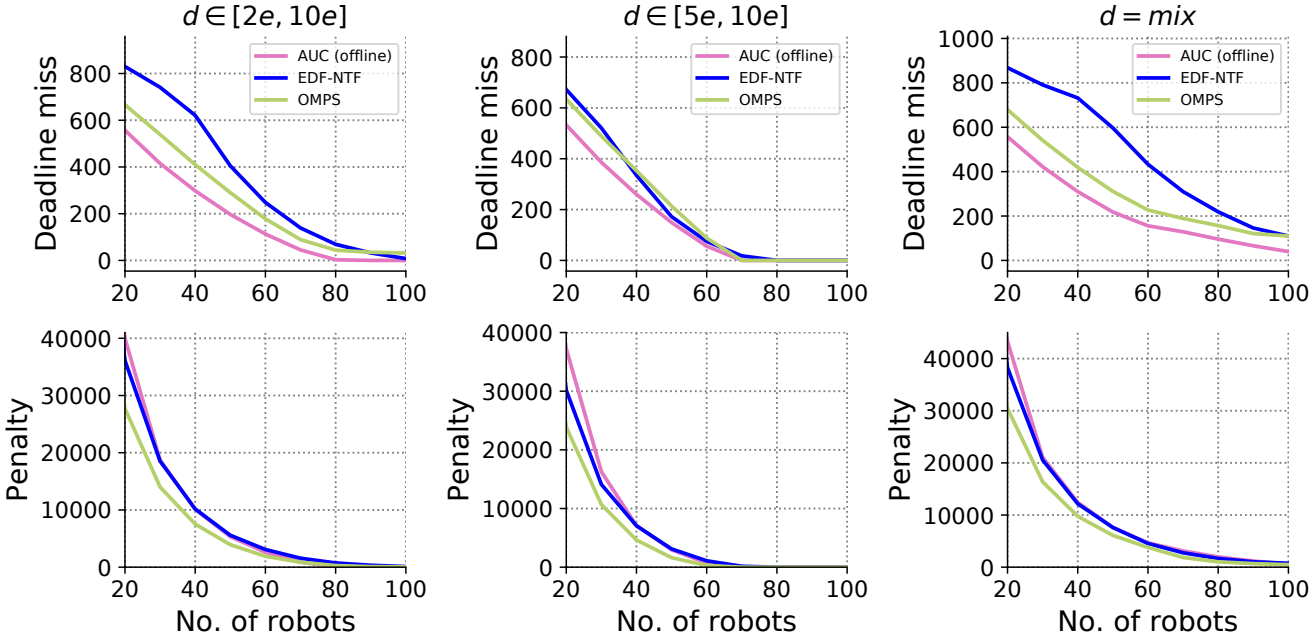


Fig. 5: Performance comparison for steady high traffic, linearly increasing robots, penalty values distributed between $[1, 10]$ and three different deadline (relative) patterns.

(steady high traffic and spiking traffic), the performance of the scheduler is analogous. Experiments henceforth are performed with steady high traffic.

Fig. 4 magnifies the execution pattern for tasks that arrive at time τ_0+25. OMPS schedules traffic such that fewer tasks miss the deadline as compared to EDF-NTF. OMPS delays tasks with lower penalty value (e.g., task id 58) and gives priority to the tasks with a higher penalty, minimizing the overall loss/penalty to the system. However, constant delay in execution of low penalty tasks may lead to their starvation. Thus, OMPS also tackles starvation avoidance as described in Section IV.

E. Robustness

Algorithm design in OMPS does not favor any specific task characteristics. To establish that, we also experiment with datasets of different types of deadline and penalty values. The results are summarized in Table I. We also point out the competitiveness of OMPS against the best offline algorithm (AUC) in terms of the number of tasks meeting the deadline. OMPS attains a maximum competitive ratio of *1* and a minimum of *0.83*.

To show how our algorithm performs in terms of the number of resources, we evaluate it for varying number of robots and three different task deadline patterns, sketched in Fig. 5. Intuitively, when the number of available robots is large, the performance of a scheduler should improve in terms of minimizing the deadline miss and penalty. This trend is reflected by AUC (offline), EDF-NTF as well as OMPS. We varied the number of available robots between 20 and 100. This confronts two aspects – (i) robustness of the scheduler with varying number of resources (robots), and (ii) performance improvement with the increasing number of

resources. With 70 robots and $d \in [5e, 10e]$, i.e., the tasks have a much larger deadline, almost all tasks are successfully executed by OMPS obeying the deadline constraint. To achieve the same performance, EDF-NTF required a higher number of robots. With $d \in [2e, 10e]$ and $d \leftarrow mix$, when a prodigious number of tasks are bound to miss deadline, OMPS outperforms both AUC and EDF-NTF in terms of incurred penalty. Though the number of tasks missing the deadline is always less in AUC, the reason it being an offline approach where the entire task set is known to the scheduler at the beginning. However, OMPS is a natural choice over AUC when characteristics of the future tasks are not known and the objective is to minimize the overall system penalty rather than minimizing the deadline miss.

VI. Conclusions

In this work, we present an online heuristic algorithm, called *online minimum penalty scheduling (OMPS)* for multi-robot task allocation, where future task characteristics, i.e., arrival time, execution time, deadline, etc. is not known to the system. Unlike other real-time systems, where a task is discarded if it cannot be scheduled within its deadline, every task in a warehouse must be completed (sooner or later) to avoid customer abandoning. Moreover, the tasks are non-preemptive and cannot be abandoned once started. To compensate for the delay in execution, a task attracts a penalty if it is completed beyond its deadline. Even though there exists a large number of methods in multi-robot and multiprocessor domain, none of them tackle the same problem, hence underperforms. On the other hand, using compaction and list-scheduling OMPS handles such a situation very efficiently such that the overall penalty to the system is minimal under fluctuating task in-flow. Even though OMPS is targeted for a warehouse scenario, it is equally applicable in other domain. Since it achieves a lower deadline miss as compared to the state-of-the-art algorithm, it can be used as a de facto method for online MRTA use-cases. In the future, we plan to extend this work for heterogeneous types of robots and perform a thorough study for dynamic scenarios.

References

[1] Capacitated Vehicle Routing Problem Library. http://vrp.atd-lab.inf.puc-rio.br/index.php/en/. Accessed on: 2017-09-01.

[2] S. Amador Nelke and R. Zivan. Incentivizing cooperation between heterogeneous agents in dynamic task allocation. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 1082–1090. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[3] T. P. Baker. Comparison of empirical success rates of global vs. partitioned fixed-priority and edf scheduling for hard real time. 2005.

[4] D. Claes, F. Oliehoek, H. Baier, and K. Tuyls. Decentralised online planning for multi-robot warehouse commissioning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 492–500. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[5] G. P. Das, T. M. McGinnity, S. A. Coleman, and L. Behera. A distributed task allocation algorithm for a multi-robot system in healthcare facilities. *Journal of Intelligent & Robotic Systems*, 80(1):33–58, 2015.

[6] R. De Koster, T. Le-Duc, and K. J. Roodbergen. Design and control of warehouse order picking: A literature review. *European journal of operational research*, 182(2):481–501, 2007.

[7] N. Fisher and S. Baruah. The partitioned multiprocessor scheduling of non-preemptive sporadic task systems. In *14th International conference on real-time and network systems*, 2006.

[8] S. P. Fung. Online preemptive scheduling with immediate decision or notification and penalties. In *International Computing and Combinatorics Conference*, pages 389–398. Springer, 2010.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[10] B. P. Gerkey and M. J. Mataric. Sold!: Auction methods for multi-robot coordination. *IEEE transactions on robotics and automation*, 18(5):758–768, 2002.

[11] S. Henn, S. Koch, and G. Wäscher. Order batching in order picking warehouses: a survey of solution approaches. In *Warehousing in the global supply chain*, pages 105–137. Springer, 2012.

[12] B. Kartal, E. Nunes, J. Godoy, and M. Gini. Monte carlo tree search with branch and bound for multi-robot task allocation. In *The IJCAI-16 Workshop on Autonomous Mobile Service Robots*, 2016.

[13] G. A. Korsah, A. Stentz, and M. B. Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[14] R. Li, X. Cheng, and Y. Zhou. Online scheduling for jobs with nondecreasing release times and similar lengths on parallel machines. *Optimization*, 63(6):867–882, 2014.

[15] L. Lin and Z. Zheng. Combinatorial bids based multi-robot task allocation method. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 1145–1150. IEEE, 2005.

[16] L. Luo, N. Chakraborty, and K. Sycara. Distributed algorithm design for multi-robot task assignment with deadlines for tasks. In *2013 IEEE International Conference on Robotics and Automation*, pages 3007–3013. IEEE, 2013.

[17] C. Sarkar and M. Agarwal. Cannot avoid penalty? lets minimize. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

[18] C. Sarkar, S. Dey, and M. Agarwal. Semantic knowledge driven utility calculation towards efficient multi-robot task allocation. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 144–147, Aug 2018.

[19] C. Sarkar, H. S. Paul, and A. Pal. A scalable multi-robot task allocation algorithm. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[20] J. M. Schutten. List scheduling revisited. *Operations Research Letters*, 18(4):167–170, 1996.

[21] N. Thibault and C. Laforest. On-line time-constrained scheduling problem for the size on\kappa machines. In *8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'05)*, pages 20–24. IEEE, 2005.

[22] J. Turner, Q. Meng, G. Schaefer, and A. Soltoggio. Distributed strategy adaptation with a prediction function in multi-agent task allocation. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 739–747. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[23] P. Visalakshi and S. Sivanandam. Dynamic task scheduling with load balancing using hybrid particle swarm optimization. *Int. J. Open Problems Compt. Math*, 2(3):475–488, 2009.