# Cannot avoid penalty? Let's minimize

Chayan Sarkar and Marichi Agarwal

***Abstract*— Multi-robot systems are deployed in a warehouse to automate the process of storing and retrieving objects in and out of the warehouse. The efficiency of the system largely depends on how the tasks are allocated to the robots. Though there exists a number of techniques that can perform multi-robot task allocation quite efficiently, they hardly consider deadline of task completion while assigning tasks to the robots. A careful allocation is of paramount importance when there is an associated penalty with each of the tasks if it is not completed within a stipulated time. In this work, we develop an algorithm, called *Minimum Penalty Scheduling* (MPS) that allocates tasks among a group of robots with the goal that the overall penalty of executing all the tasks can be minimized. Our algorithm provides a robust, scalable, and near-optimal real-time task schedule. By comparing with the state-of-the-art algorithm, we show that MPS attracts up to 62.5% less penalty when a significant number of tasks are bound to miss the deadline. Additionally, MPS is also suitable for real-time multi-processor scheduling since it finds schedule such that a higher number of tasks can meet their deadline.**

## I. Introduction

In a warehouse, goods are systematically stored across a large area and they are taken out of the storage as and when there is an order for the goods. Moving goods in and out of the warehouse account for the majority proportion of its operational cost [1]. Copious amounts of demands flowing into the warehouse management system (WMS) in real-time necessitate designing of an efficient goods movement schedule.

**Motivation**. Multi-Robot task allocation (MRTA) has gained significant attention over the past decade. A multi-robot system increases parallelism, thereby increases system throughput compared to a single-robot system. Multi-robot parallelism galvanizes to answer "which task is to be scheduled on which robot and when?". In this article, we assume a warehouse scenario where a task refers to the good pick up based on the order placed by the customer. There exists a number of methods that look into multi-robot task allocation [2], [3], [4]. However, they usually do not consider any deadline associated with the tasks. On the other hand, handling high volume orders diurnally and seasonally in a warehouse lead to a situation where a large proportion of the tasks are bound to miss their deadline. Unlike classical real-time systems, where a task is usually discarded if it misses its deadline, a task in a warehouse scenario has to be fulfilled even if its deadline is surpassed. This entails formulating a soft real-time stratagem where a task missing its deadline can be tolerated at the cost of some penalty to the business.

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

**Problem description**. In this paper, we address the problem of scheduling atomic tasks to the team of homogeneous autonomous warehouse robots such that the overall penalty incurred to complete all the tasks can be minimized. Each task ($t_i$) is characterized by an execution time ($e_i$), a deadline ($d_i$), and a penalty per time unit ($p_i$). If a task finishes its execution at time instant $c_i$, it incurs a penalty of $(c_i - d_i) * p_i$ units. We assume that a wave of tasks flows into the system at a time. All the tasks in the wave are required to be finished as soon as possible using the available number of resources (robots) and then the next wave is imported.

**Approach**. Existing MRTA algorithms can be classified into two broader categories – centralized and distributed. Distributed algorithms are robust and efficient only when all the information is not known *a priori* (centrally). Since the premise of our work is a warehouse scenario, where all the information about the environment is available at the WMS, we design a centralized algorithm. In the multi-processor systems, where the scheduling is done centrally, it is known to be an NP-Hard problem in a strong sense [5]. Most of the prevalent algorithms in the literature tries to find a schedule for the tasks given they are schedulable. In case, some of the tasks are bound to miss their deadline, the existing algorithms try to minimize the number of deadline miss. Extending these algorithms is not sufficient for the warehouse scenario since minimum deadline miss do not guarantee minimum overall penalty. For example, if a task incurs a very high penalty value, it would be wise to schedule it before its deadline even at a cost of not scheduling two tasks with relatively very low penalty value. Thus, we develop a new algorithm, called *Minimum Penalty Scheduling* (MPS) that aims to minimize penalty while assigning tasks to the warehouse robots. The algorithm works in two phases. In the first phase, we do a tight scheduling of the tasks among the robots using compaction. The goal of this phase is to schedule as many tasks as possible within their deadline with a focus on reducing the overall penalty. In the second phase, we assign the tasks that missed their deadline such that penalty incurred due to their execution can be minimized.

**Contributions**. The main contribution of this article is two-fold.

- To best of our knowledge, MPS is the first algorithm for MRTA that considers deadline and penalty while scheduling tasks to the robots with the goal of minimizing the overall penalty.
- We also reduce the total number of deadline miss. Thus, MPS is suitable for multi-processor task scheduling with sporadic, non-preemptive tasks.

## II. Related Work

In general, assignment problems study allocation of jobs to the available agents under various constraints such that all the jobs can be completed in an optimized way. Multi-robot task allocation (MRTA) is a direct descendant of the assignment problem and a huge body of prevalent work exists in the literature. Similarly, multi-processor task scheduling also looks at real-time task assignment such that a maximum number of tasks can meet their deadline. We review some of the prevalent works from both of these domains.

### A. Multi-robot task allocation

A formal analysis and taxonomy of task allocation in multi-robot systems is provided by Gerkey *et al.* [6]. On the time dimension, task allocation is broadly classified as instantaneous assignment (IA) and time-extended assignment (TA), where IA refers to prompt allocation of tasks to the robots and no future allocation is possible, and TA refers to task assignment over a time period that need not be addressed immediately. An instantaneous assignment can be done optimally using the Hungarian method [7]. However, it is not applicable for distributed setup. Thus, a number of distributed auction-based methods are proposed in the literature [8], [9], [10]. Though the decentralized approaches are more robust, Kartal *et al.* [11] finds a gap in theoretical guarantees for these work and proposes a centralized task allocation mechanism for non-trivial data set using Monte Carlo tree search. Since the warehouse is a controlled environment where most of the information available centrally, a centralized algorithm is more suitable for such cases.

Most of the recent works consider Task Allocation Problem of Abundant Robots (TAPAR), where the warehouse has a sufficient number of robots to finish the tasks of picking one order [12], [13]. Christopher *et al.* [14] proposed *alphabet soup* method for the problem. On the contrary, the less dealt Task Allocation Problem of Insufficient Robots(TAPIR) was investigated by Li *et al.* [15] for a warehouse setup that uses cargo-to-person mode. Recently, Sarkar *et al.* [16] proposed a time-extended task assignment for a warehouse that aims to reduce the overall task execution time using a vehicle routing method where the number of tasks outnumbers the number robots by multiple factors. However, none of these methods consider a deadline associated with the tasks.

There are a few MRTA mechanisms that perform real-time task allocation [17], [18]. They assume a scenario, where tasks are schedulable within their deadline. But, during the peak time, all tasks in a warehouse may not be schedulable within their deadline. Irrespective of the deadline guarantee, all tasks need to be completed sooner or later. This is modeled as attracting a penalty value if a task cannot be completed within its deadline. Our work focuses on real-time task scheduling with the goal of minimizing the overall penalty.

### B. Task scheduling on multiprocessor

Real-time scheduling problems are known to be an NP-hard [5], where the common aim is to maximize the number of tasks scheduled without deadline miss, instead of minimizing the penalty if missing the deadline cannot be avoided. Some work in literature pose penalty as loss accounted on the system in case of task preemption or an accepted task later dropped if deadline constraint could not be met [19], [20]. Most of the prominent works in literature find a solution where preemption is allowed [21], [22], [23], i.e., a task can be preempted on one processor and later can be resumed on another processor. In due course of time, the focus has shifted from full preemption to the case of restricted-preemption where a task can be preempted only after executing for a certain duration [24], [25]. Though there exists a number of non-preemptive task scheduling algorithm [26], [27], but they are either applicable for single processor system or for periodic task system. Since a task cannot be preempted in a warehouse scenario, these techniques are not suitable.

## III. The Problem of Task Scheduling

We provide a formal description of the multi-robot task allocation problem in a warehouse scenario where robots bring the goods from the storage racks spread across a large area to the packaging dock as per customer order. Fetching the individual goods is designated as a task for the robot and the time to fetch the good is termed as the corresponding execution time. Each task is expected to be completed before a stipulated time (deadline), which is decided by the warehouse management system (WMS) based on the customer profile and the order type. Since the available resources (robots) are fixed and the number of task in-flow can be very large during a certain time of the day and during certain periods of the year, a large number of tasks are bound to miss their deadline during these high demand periods. But, unlike the usual real-time systems, where a task is discarded if it cannot be scheduled within its deadline, a task in a warehouse must be completed sooner or later. However, if a task fails to be executed within the deadline, a penalty proportional to the delay is imposed on the business (defined by WMS). The objective of this work is to schedule the tasks in such a way that the total penalty is minimized in case some of the tasks fail to meet their deadline.

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

- Tasks are executed in a wave (like in most of the existing WMS), i.e., after completing a set of tasks, the next set of tasks (wave) is imported for execution.
- WMS provides the task-id, execution time, deadline, and penalty for the tasks in a wave.
- If a customer places an order for multiple goods, picking up each good is treated as a separate task. In this case, each task can have a different execution time. But, they can be associated with the same deadline and penalty.
- All robots are homogeneous, each object can be carried by a single robot, and each robot carries only one object at a time.
- For simplicity, we assume that the time to pick an object from the storage rack is the same for all tasks, and not

included in the task execution time. Thus, task execution time only constitutes the travel duration of the robot.
- Robots have sufficient energy to finish all tasks assigned to it during a wave. Robots have full/sufficient knowledge of the warehouse environment.
- Every task is atomic and non-preemptive. A robot is never idle if there is a task in the queue awaiting execution.

### A. Mathematical Formulation

Suppose, a set of $n$ tasks, $T = \{t_1, t_2, ..., t_n\}$, is available at the beginning of a wave. A task can formally be represented as $t_i = (e_i, d_i, p_i)$, where $e_i$ is the execution time, $d_i$ is the deadline, and $p_i$ is the penalty. These tasks are to be scheduled on $m$ identical warehouse robots, $R = \{r_1, r_2, ..., r_m\}$, with the associated deadline and penalty consideration. As mentioned earlier, we assume soft-deadline, i.e., if a task fails to meet its deadline, a penalty proportional to the lateness is imposed. 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 $t_i$. This signifies that if a task is executed within the deadline, there is no penalty; otherwise the penalty increases with time. Our proposed scheduler decides how to partition these $n$ tasks amongst the $m$ robots and the order of executing the subset of tasks assigned to a robot. Hence, the objective function of our scheduler is given by,

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

### B. Integer linear program (ILP) formulation

Luo *et al.* [17] propose an ILP formulation for tasks with unit execution time and deadline constraint. However, the model does not include any penalty value associated with the task if it fails to be scheduled within its deadline. Thus, we propose a new ILP formulation where the goal is to minimize the overall penalty of the system.

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 given below.

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

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

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

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

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. $K$ is set to a large enough value that is sufficient to complete all the task. We set $K$ to $1.5 * \lceil \frac{n}{m} \rceil$.

## IV. Scheduler Design

The multi-robot task scheduling problem with different execution time for the tasks and deadline is a special case of the generalized assignment problem (GAP), which is known to be an NP-hard problem [17]. Thus, we develop a heuristic algorithm, called *Minimum Penalty Scheduling* (MPS). In the following section, we describe *MPS* that schedules non-preemptive, sporadic tasks with soft deadline on a multi-robot system. It constitutes of two phases – (i) in the first phase, it tries to assign maximum number of tasks to the robots, meeting deadline, with the focus that high penalty tasks are scheduled with higher priority, and (ii) in the second phase, it assigns the non-schedulable tasks, i.e., the tasks that missed their deadline in the first phase, in such a way that the accumulated penalty can be minimized.

### A. Schedule tasks without deadline miss (phase I)

In this phase, the set of $n$ tasks are split into $m+1$ disjoint sets, i.e., $T(r_1), T(r_2), ..., T(r_m), \bar{T}$, where $T(r_j)$ represents the subset of tasks that can be schedulable on robot $r_j$ and $\bar{T}$ is the set of tasks that are not schedulable on any of the robots without violating the deadline constraint of already scheduled tasks. Essentially, this splitting assigns all the schedulable tasks to the robots that can be executed with zero penalties.

One of the prevalent techniques to schedule real-time tasks is to first sort the tasks based on some criteria and then consider them one-by-one from the list. Though *MPS* also does the same, but the sorting criteria and schedulability conditions in *MPS* yields much higher efficiency. Fisher *et al.* [28] provides an extensive study on real-time task scheduling and they concluded that sorting the tasks in non-increasing order of their execution time attains maximum schedulability. However, to achieve a lower penalty, this proves to be inefficient. Thus, MPS first categorizes tasks by their penalty value, giving higher priority to tasks with a higher penalty. For the tasks with the same penalty, the tie is broken by rearranging them in *increasing* order of slack time, which is defined as $d_i - e_i$.

After sorting, *MPS* uses *compaction* based schedulability check such that the tasks are compressed together to make room for other tasks. In other words, it avoids fragmentation of time on every robot. To assure schedulability of task $t_l$ on robot $r_k$, *MPS* defines Eq. 6 and Eq. 7 as the necessary condition. Eq. 6 ensures that there is enough slack to execute $t_l$ after executing all the tasks that are already scheduled at $r_k$ and has deadline smaller than that of $t_l$. Eq. 7 ensures that the scheduled tasks with deadline greater than $t_l$ are still schedulable if we schedule $t_l$ as well. If either of these criteria fails for all the robots, the task is added to $\bar{T}$. Please note, since *MPS* is a heuristic approach, it does not provide

any guarantee about the optimality of this split.

$$d_l - \left( \sum_{\substack{t_j \in T(r_k) \\ d_j \leq d_l}} e_j + e_l \right) \geq 0. \quad (6)$$

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

### B. Schedule delayed tasks minimizing the penalty (phase II)

In this phase, tasks in $\bar{T}$ are assigned based on an execution precedence value. To elaborate, let us assume tasks $t_a$ and $t_b$ (from $\bar{T}$) are assigned to the same robot in the order of $t_a$ first and then $t_b$ and they attract a combined penalty of $P$. If the execution time of $t_a$ is smaller, then the execution of $t_b$ is delayed by a smaller amount of time, attracting a lower penalty. On the other hand, if the execution time of $t_a$ is larger, the delay would be higher as well as the penalty. Thus, it can be attributed that $P \propto e$. On the other hand, if the deadline of $t_b$ is smaller, it waits a long time and accumulates a larger penalty, as opposed to a larger deadline would accumulate a lower penalty; thus, $P \propto 1/d$. Now, if both the tasks have the same execution time and deadline, the one with greater penalty value would attain higher overall penalty ($P \propto p$). Thus, the higher penalty value should be given higher precedence in the order of execution. Hence, the execution precedence value for the tasks in $\bar{T}$ is calculated as,

$$\forall t_i \in \bar{T}, \;\; p_i * (k - d_i)/e_i, \quad (8)$$

where $k$ is the time of earliest available robot. The task with the highest precedence value is removed from $\bar{T}$ and assigned to the earliest available robot. The algorithm continues until all the tasks in $\bar{T}$ are assigned to some robot.

After completing the task assignment, MPS performs load balancing to reduce the makespan, i.e., to finish executing the current wave of tasks as soon as possible. A robot can be overloaded in the first phase since schedulability check only considers idle time pockets on a robot and the deadline of the tasks. In other words, if a task is schedulable on multiple robots, it is assigned to any one of them without considering the accumulated load on them. During load balancing, an idle robot takes up task $t_i$ from an overloaded robot only if it is capable of executing it without violating the schedule of the already assigned tasks. This improves makespan without incurring additional system penalty or deadline miss.

## V. Evaluation

In order to evaluate the performance of *MPS*, a number of 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 [29] 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 the datasets with 100 and 1000 tasks (site locations). As mentioned earlier, each task is represented as a tuple, $t_i =< e_i, d_i, p_i >$, 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. For each of these two datasets (100 and 1000 tasks), we generate four different datasets comprising of pseudo-randomly generated deadlines.

- $d_i = 2e_i$: In this set, the 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 scenario for the scheduler.
- $d_i \in [e_i, 10e_i]$: Deadline is set to a random value in the range of $e_i$ and $10e_i$ (uniformly distributed).
- $d_i \in [5e_i, 10e_i]$: Again, the 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.
- $d \leftarrow mix$: In order to ensure, the scheduler do not favor a particular distribution of deadline, we take a mix of all the previous three datasets with equal probability.

The strategy to associate deadline is similar to [30], which is a standard procedure to generate data in real-time scheduling domain. Now, for each of these 8 sets (4 each for 100 and 1000 tasks), we generate 3 datasets each with a different penalty value.

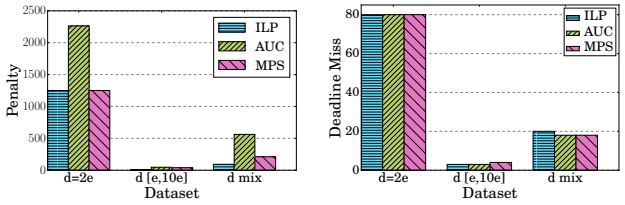
- $p \in [1, 10]$: Penalty is set to a random value generated using a uniform distribution in the range of 1 and 10.
- $p \leftarrow same$: Penalty is set to the same value (e.g., 1) for all the tasks.
- $p \leftarrow extreme(1 \; or \; 10)$: Penalty is set to the either of the two extreme values, i.e.,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

### B. Methodology

We evaluate *MPS* based on three metrics - (i) overall penalty incurred while completing all the tasks, (ii) the number of deadline miss, and (iii) the makespan of the wave. Makespan is defined by $\max_{1 \leq j \leq m} c_{T(r_j)}$,, where $c_{T(r_j)}$ represents the completion time of all the tasks assigned to robot $r_j$. The number of tasks missing the deadline and hence the incurred penalty depends on the number of tasks in a wave and the number of available robots. For the datasets with 100 tasks, we fixed the number of robots to 10. On the other hand, we varied the number of available robots between 10 and 100 for the datasets with 1000 tasks. This confronts two aspects – (i) robustness of the scheduler with

TABLE I: Performance comparison for 12 datasets with 100 tasks and 10 robots.

| Penalty | Deadline | AUC | | | DBF | | | MPS | | | Penalty Reduction (%) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $P(10^2\times)$ | *DL-M* | *MS* | $P(10^2\times)$ | *DL-M* | *MS* | $P(10^2\times)$ | *DL-M* | *MS* | *over AUC* | *over DBF* |
| $p \in [1, 10]$ | $d_i = 2e_i$ | 760 | 64 | 550 | 1409 | 84 | 500 | 531 | 68 | 534 | 30.13 | 62.31 |
| | $d_i \in [e_i, 10e_i]$ | 49 | 7 | 744 | 887 | 55 | 534 | 36 | 10 | 534 | 26.53 | 95.94 |
| | $d_i \in [5e_i, 10e_i]$ | 0 | 0 | 776 | 416 | 30 | 590 | 0 | 0 | 522 | 0 | 100 |
| | $d \leftarrow mix$ | 120 | 14 | 782 | 945 | 57 | 502 | 58 | 17 | 532 | 51.67 | 93.86 |
| $p \leftarrow same$ | $d_i = 2e_i$ | 135 | 64 | 550 | 234 | 84 | 500 | 135 | 64 | 550 | 0 | 42.31 |
| | $d_i \in [e_i, 10e_i]$ | 18 | 7 | 744 | 154 | 55 | 534 | 24 | 12 | 532 | -33.33 | 84.42 |
| | $d_i \in [5e_i, 10e_i]$ | 0 | 0 | 776 | 61 | 30 | 590 | 0 | 0 | 528 | 0 | 100 |
| | $d \leftarrow mix$ | 22 | 14 | 782 | 162 | 57 | 502 | 21 | 15 | 524 | 4.55 | 87.04 |
| $p \leftarrow extreme$ | $d_i = 2e_i$ | 695 | 64 | 550 | 1098 | 84 | 500 | 314 | 70 | 532 | 54.82 | 71.40 |
| | $d_i \in [e_i, 10e_i]$ | 142 | 7 | 744 | 692 | 55 | 534 | 24 | 11 | 530 | 83.10 | 96.53 |
| | $d_i \in [5e_i, 10e_i]$ | 0 | 0 | 776 | 271 | 30 | 590 | 0 | 0 | 542 | 0 | 100 |
| | $d \leftarrow mix$ | 113 | 14 | 782 | 733 | 57 | 502 | 30 | 19 | 528 | 73.45 | 95.91 |




(a) Penalty (b) deadline miss

Fig. 1: Performance comparison for 100 tasks and 10 robots system for unit execution time, three different deadline and the penalty, $p \in [1, 10]$.

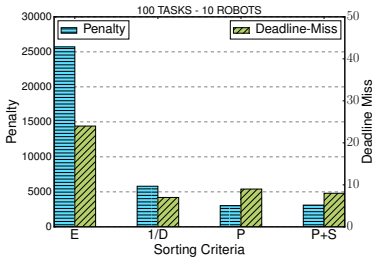



Fig. 2: Penalty (left y-axis) and deadline-miss (right y-axis) for 100 tasks with $p \in [1, 10]$ and $d \in [e, 10e]$ executed using *MPS* with different sorting criteria in phase I.

varying number of resources(robots), and (ii) performance improvement with the increasing number of resources.

Though multi-robot task assignment has been studied extensively, there are only a few instances that consider tasks with deadline [17], [18]. To evaluate the efficacy of *MPS*, we compare it with the *Auction-based Distributed Algorithm* (*AUC*) proposed by Luo *et al.* [18]. Since there exists a large number of work in the multi-processor scheduling domain, it is imperative to evaluate *MPS* from that perspective as well. Specifically, we compare *MPS* with *demand bound function* (*DBF*), a state-of-the-art scheduling technique that considers multi-processor real-time task scheduling for sporadic and non-preemptive tasks [28].

### C. Near Optimality

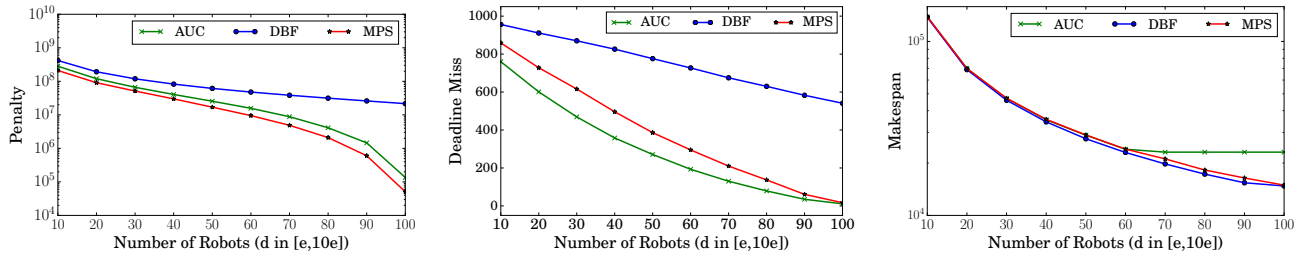
Since the scheduling problem in hand is an NP-hard problem, our scheduler resort to a heuristic approach. Thus, it is essential to assess *MPS* as compared to the optimal solution. Though the ILP formulation can provide an optimal solution for the generalized assignment problem, it becomes intractable even for 100 tasks. Thus, we relaxed the problem by setting unit execution time for all the tasks.

Fig. 1b summarizes the outcome of *ILP*, *AUC*, and *MPS* for 100 tasks and 10 robots with tasks having unit execution time, but different deadlines and penalty. Four different distribution for the deadline is considered as discussed in Section V-A. For the deadline distribution of $d_i \in [5e_i, 10e_i]$, all the tasks are schedulable incurring no penalty; hence we omit the results. From this figure, it is evident that our scheduler closely follows the optimal solution. Though *AUC* achieves a lower deadline miss, the overall penalty for *MPS* is much lower than that of *AUC*. As the primary goal of the scheduling is to schedule tasks such that penalty can be minimized, *MPS* proves to be more efficient than *AUC* and close to the optimal solution.

### D. Robustness of MPS

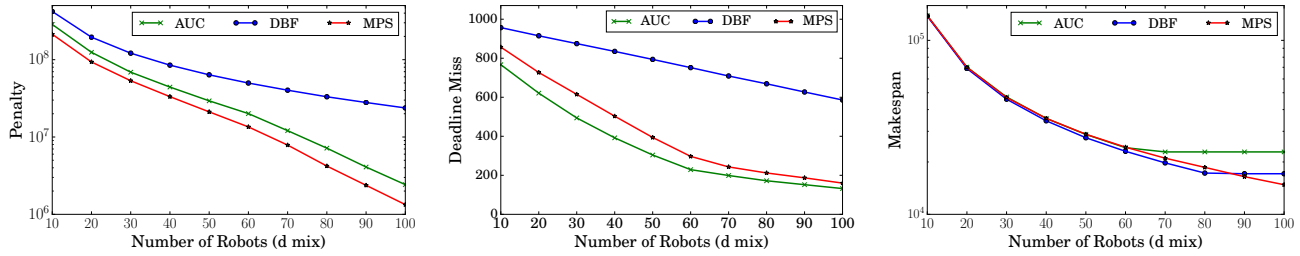
We extensively evaluate *MPS* with different execution time and compare it with both *AUC* and *DBF*. Table I summarizes the results for all the 12 datasets with 100 tasks and 10 robots.

The results show that *MPS* always outperforms *DBF* w.r.t. penalty and deadline miss. With respect to *AUC*, *MPS* outperforms it significantly in terms of overall penalty (most of the time) while the deadline miss is almost equivalent. This is because *MPS* uses compaction while trying to maximize the schedulable tasks. Additionally, *MPS* tries to assign maximum tasks to the busiest robot, keeping some of them as little occupied as possible such that they can be assigned tasks from *phase II* at the earliest possible time. Early beginning of the tasks that have missed their deadline would minimize the overall penalty. However, this poses a drawback for as it can increase the makespan forcing the system a delayed import of the next wave of tasks. *MPS* overcomes this limitation by employing a load balancing mechanism at the end of *phase II*. Additionally, *MPS* uses a different sorting mechanism as compared to *DBF* for both the *phase I* and *phase II* (see Section IV). This results in a considerable




(a) overall penalty (b) deadline miss (c) makespan

Fig. 3: Performance comparison for 1000 tasks and varying number of robots. The deadline and penalty values are uniformly distributed in $[e_i, 10e_i]$ and $[1, 10]$, respectively.




(a) overall penalty (b) deadline miss (c) makespan

Fig. 4: Performance comparison for 1000 tasks and linearly increasing robots. The deadline is a mix of three different distribution and the penalty is uniformly distributed in $[1, 10]$.

improvement in makespan, which is equivalent to DBF.

To conclude whether *MPS* indeed uses a better sorting criterion, we investigate its performance with different sorting mechanisms. From Fig. 2, it is evident that sorting based on penalty yields not only lesser penalty accumulation, but a less number of deadline miss as opposed to higher execution time first (non-increasing $E$) and earliest deadline first (non-increasing $1/D$) strategy.

### *E. Scalability of MPS*

To show the scalability of our algorithm in terms of the number of tasks and robots, we evaluate it for 1000 tasks and a varying number of robots. 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*, *DBF* as well as *MPS*. However, the performance of *MPS* and *AUC* improves multi-fold with increasing number of robots contrary to a slower improvement using *DBF* (Fig. 3 and Fig. 4). It also shows that almost all the tasks are scheduled within their deadline with 100 robots using and *AUC*, while *DBF* still incurs a significant number of deadline miss (Fig. 3b and 4b). These results show that *MPS* outperforms *AUC* in terms of overall penalty in all our test cases when there is a large number of tasks. For example, *MPS* performs 62.5% better over *AUC* algorithm in terms of penalty for 1000 tasks and 100 robots when the deadline is uniformly distributed in $[e_i, 10e_i]$.

Though the makespan of *MPS* is equivalent to *AUC* and *DBF* when there is a relatively lesser number of robots, with the increase in available robots, *MPS* surpasses *AUC* in terms of makespan as well (for 60 robots in Fig. 3c and 4c).

## VI. Conclusion

In this article, we tackle the problem of scheduling a wave of tasks to a group of homogeneous robots in a warehouse. A task in a warehouse is associated with a soft deadline that signifies the desired time by which it should be completed; otherwise, it incurs a penalty on the warehouse management system. However, during the peak times, a large number of tasks are bound to miss their deadline leading to a significant accumulation of penalty. Thus, we develop a multi-robot task scheduling algorithm, called *minimum penalty scheduling* (MPS) that tries to minimize the overall penalty while completing all the tasks. Using a mix of compaction, list-scheduling, and load balancing techniques, MPS achieves a significantly low penalty solution as compared to the state-of-the-art algorithms. Using extensive simulation based on large datasets, we show that MPS is robust, scalable and near optimal.

One natural extension of this is an online task assignment algorithm. Since the modern WMS designers are exploring the option of handling the tasks as-they-come instead of traditional wave-based system, an online assignment algorithm would become a relevant requirement. Another pertinent extension would be for the heterogeneous team of robots. This is realistic even in a warehouse scenario as a warehouse may employ multiple generations of the same robot or robots from different manufacturers.

## REFERENCES


[1] Tushar Khare, "Optimization of Warehouse Cost & Workforce Forecasting," http://www.nectechnologies.in/en_TI/pdf/Whitepaper_OptimizationOfWarehouseCost.pdf, 2015, online; accessed 29 January 2018.

[2] 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*. International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 492–500.

[3] J. Godoy and M. Gini, "Task allocation for spatially and temporally distributed tasks," *Intelligent Autonomous Systems 12*, pp. 603–612, 2013.

[4] E. Nunes and M. L. Gini, "Multi-robot auctions for allocation of tasks with temporal constraints." in *AAAI*, 2015, pp. 2110–2116.

[5] M. R. Garey and D. S. Johnson, "A guide to the theory of np-completeness," *WH Freemann, New York*, vol. 70, 1979.

[6] B. P. Gerkey and M. J. Mataric, "Multi-robot task allocation: Analyzing the complexity and optimality of key architectures," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 3. IEEE, 2003, pp. 3862–3868.

[7] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics (NRL)*, vol. 2, no. 1-2, pp. 83–97, 1955.

[8] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multi-robot coordination," *IEEE transactions on robotics and automation*, vol. 18, no. 5, pp. 758–768, 2002.

[9] S. Koenig, P. Keskinocak, and C. A. Tovey, "Progress on agent coordination with cooperative auctions." in *AAAI*, vol. 10, 2010, pp. 1713–1717.

[10] E. Schneider, E. I. Sklar, S. Parsons, and A. T. Özgelen, "Auction-based task allocation for multi-robot teams in dynamic environments," in *Conference Towards Autonomous Robotic Systems*. Springer, 2015, pp. 246–257.

[11] B. Kartal, E. Nunes, J. Godoy, and M. L. Gini, "Monte carlo tree search for multi-robot task allocation." in *AAAI*, 2016, pp. 4222–4223.

[12] Y. Guo, "Auction-based mulit-agent task allocation in smart logistic center," in *Harbin: Conference Harbin Institute of Technology*, 2010.

[13] H. Ma, X. Wu, Y. Gong, Y. Cui, and J. Song, "A task-grouped approach for the multi-robot task allocation of warehouse system," in *Computer Science and Mechanical Automation (CSMA), 2015 International Conference on*. IEEE, 2015, pp. 277–280.

[14] C. J. Hazard, P. R. Wurman, and R. DAndrea, "Alphabet soup: A testbed for studying resource allocation in multi-vehicle systems," in *Proceedings of the 2006 AAAI Workshop on Auction Mechanisms for Robot Coordination*, 2006, pp. 23–30.

[15] Z. Li and W. Li, "Mathematical model and algorithm for the task allocation problem of robots in the smart warehouse," *American Journal of Operations Research*, vol. 5, no. 06, p. 493, 2015.

[16] C. Sarkar, H. S. Paul, and A. Pal, "A scalable multi-robot task allocation algorithm," in *Robotics and Automation (ICRA), Proceedings of the IEEE International Conference on*. IEEE, 2018.

[17] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithm design for multi-robot task assignment with deadlines for tasks," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3007–3013.

[18] ——, "Distributed algorithms for multirobot task assignment with task deadline constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 876–888, 2015.

[19] S. P. Fung, "Online preemptive scheduling with immediate decision or notification and penalties," in *International Computing and Combinatorics Conference*. Springer, 2010, pp. 389–398.

[20] N. Thibault and C. Laforest, "Online time constrained scheduling with penalties," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–8.

[21] J. H. Anderson, V. Bud, and U. C. Devi, "An edf-based scheduling algorithm for multiprocessor soft real-time systems," in *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*. IEEE, 2005, pp. 199–208.

[22] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems," *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 918–923, 2006.

[23] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*. IEEE, 2006, pp. 101–110.

[24] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*. IEEE, 2005, pp. 137–144.

[25] X. Wang, Z. Li, and W. M. Wonham, "Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on des supervisory control," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 7, pp. 1082–1098, 2017.

[26] S. K. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," *Real-Time Systems*, vol. 32, no. 1-2, pp. 9–20, 2006.

[27] W. Li, K. Kavi, and R. Akl, "A non-preemptive scheduling algorithm for soft real-time systems," *Computers & Electrical Engineering*, vol. 33, no. 1, pp. 12–29, 2007.

[28] 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.

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

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