Online Multi-Agent Supervisory Control for Warehouse Automation: Prioritized Tasks

Moeto Kasahara and Kai Cai

Abstract— **In this paper we consider a problem of controlling multi-agent discrete-event systems to serve tasks dynamically appearing in the environment, where the tasks have different priorities. To address this problem, we propose an effective online supervisory control approach which uses different queues to store tasks of different priorities, and assigns agents to serve tasks in the order of their priorities. Moreover, to prevent lower-priority tasks from being unserved due to constantly incoming higher-priority ones, a timer is further associated with each task; once the timer of a task ticks down to zero, regardless of the task's priority, it will be moved to a special queue with the highest priority to be served next. We then apply this online control scheme to model and control a warehouse automation system using multiple mobile robots with prioritized tasks; the effectiveness of this scheme is demonstrated on a case study.**

I. Introduction

In [1], [2], [3], [4], [5], we have introduced a problem of controlling multi-agent discrete-event systems (DES) to serve multiple tasks, and extended supervisory control theory (SCT) [6], [7] to provide effective solutions. The study of this problem is motivated by logistic automation systems using a team of autonomous robots. A prominent application is Kiva systems in Amazon's warehouses and distribution centers [8].

The tasks considered in [1], [2], [3] are static: the information of the tasks is completely known at the outset, and no newly added tasks are considered. Moreover, which task is assigned to which robot is assumed to be given *a priori*. For this static setup, [1] shows that the standard SCT may be adapted to compute a safe and deadlock free solution for multiple agents to accomplish multiple tasks. To relieve computational burden, [2], [3] further adapt an online SCT based on a limited-lookahead strategy [9]. In this online approach, a supervisor is recomputed at the occurrence of *every* event.

In [4], [5], a more realistic setup is considered in which tasks can appear dynamically; when and where the tasks appear are unknown at the outset. An extended online supervisory control approach is proposed that recomputes a supervisor when (and only when) there are unassigned tasks and available agents. This online scheme is thus distinct from that in [2], [3], [9], and allows the recomputed supervisor to be adaptive to newly appeared tasks. Furthermore, [5] combines the online approach with optimal task assignment [10] to improve efficiency of serving dynamic tasks: the sum of (unweighted) distances from agents to tasks is minimized.

This work was supported in part by JSPS KAKENHI Grant no. 21H04875. The authors are with Department of Electrical and Information Engineering, Osaka City University. Emails: kasahara@c.info.eng.osaka-cu.ac.jp (M. Kasahara), kai.cai@eng.osaka-cu.ac.jp (K. Cai)

In this paper, we build on and further extend [5] to take into account an important realistic aspect of warehouse automation systems that tasks have different priorities. Think of a warehouse for an e-commerce platform, where it is common that there are regular users and prime members. Prime members paying annual membership fees are promised to enjoy fast shipping and deliveries. Accordingly, the products ordered by prime members have a higher priority than those ordered by regular users; hence the tasks of picking up those prioritized products need to be served sooner. This issue is dealt with by introducing two task queues, storing respectively higher-priority and lower-priority tasks; our strategy is to serve tasks in the lower-priority queue only when there are available agents and no unserved tasks in the higher-priority queue. (The situation with more than two priority levels may be similarly dealt with.)

One potential problem of the above-mentioned strategy is that if higher-priority tasks keep coming at a rate that constantly exhausts available agents, then lower-priority tasks may never get served or suffer from significant delays. This (starvation) situation is undesirable, as regular users also deserve to be served in a reasonable time frame. To address this problem, we further introduce timers to tasks, with higher-priority tasks having shorter timers. Once the timer of a task becomes zero, the task will be moved to a third queue with the highest priority to be next served. Both higher-priority and lower-priority tasks may enter the third queue as long as their timers tick down to zero. This augmented strategy effectively resolve the starvation problem.

In the sequel, we present the extended online supervisory control approach that effectively address the situation where tasks have different priorities, and illustrate the approach by a warehouse automation case study.

II. Preliminaries

A. Supervisory control basics

In SCT [6], [7], the plant to be controlled is modeled by a finite state *automaton* $\mathbf{G} := (Q, \Sigma, \delta, q_0, Q_m)$, where Q is the finite state set, $q_0 \in Q$ the *initial state*, $Q_m \subseteq Q$ the set of *marker states*, Σ the finite *event set*, and $\delta : Q \times \Sigma \to Q$ the (partial) *state transition function*. We extend δ such that $\delta : Q \times \Sigma^* \to Q$, and write $\delta(q,s)!$ to mean that $\delta(q,s)$ is defined. The event set Σ is partitioned into a subset Σ_c of *controllable* events and a subset Σ_u of *uncontrollable* events; only controllable events can be enabled or disabled by an external entity, called *supervisor*, introduced below.

The closed behavior of $\mathbf{G}$ is the set of all strings that can be generated by $\mathbf{G}$, namely $L(\mathbf{G}) := \{s \in \Sigma^* | \delta(q_0, s)!\} \subseteq \Sigma^*$.

On the other hand, the *marked behavior* of $\mathbf{G}$ is the subset of strings that can reach a marker state, i.e. $L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) | \delta(q_0, s) \in Q_m\} \subseteq L(\mathbf{G})$. $\mathbf{G}$ is *nonblocking* if $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$ ($\bar{\cdot}$ means *prefix closure*), namely every string in the closed behavior may be completed to a string in the marked behavior.

A language $E \subseteq \Sigma^*$ is said to be *controllable* (with respect to $\mathbf{G}$) if $\overline{E}\,\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{E}$. Let $K \subseteq L_m(\mathbf{G})$ be a specification language imposed on the plant $\mathbf{G}$. Denote by $C(K)$ the family of controllable sublanguages of K, i.e. $C(K) := \{K' \subseteq K | \overline{K'}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K'}\}$. Then the supremal controllable sublanguage of K exists and is given by $\sup C(K) = \cup\{K' | K' \in C(K)\}$. Let $\mathbf{SUP}$ be a (nonblocking) automaton such that $L_m(\mathbf{SUP}) = \sup C(K)$. We call $\mathbf{SUP}$ the supervisor for plant $\mathbf{G}$ that enforces $\sup C(K)$. The control action of $\mathbf{SUP}$ after an arbitrary string $s \in L(\mathbf{G})$ is to enable an event in the following set

$$\gamma(s) := \{\sigma \in \Sigma_u | s\sigma \in L(\mathbf{G})\} \cup \{\sigma \in \Sigma_c | s\sigma \in L(\mathbf{SUP})\}. \quad (1)$$

B. Optimal task assignment

The task assignment problem is an optimization problem of finding a one-to-one correspondence between an agent and a task so as to minimize the sum of the costs included in the assignment. Consider n agents, n tasks, and $c_{i,j}$ for each pair of $i, j \in \{1, \ldots, n\}$ is the cost when agent i is assigned to serve task j. Then the task assignment problem is formulated as follows.

$$\begin{array}{ll} \text{minimize} & z = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{i,j} c_{i,j} \\ \text{subject to} & (\forall j \in \{1, \ldots, n\}) \sum_{i=1}^{n} x_{i,j} = 1 \ \& \\ & (\forall i \in \{1, \ldots, n\}) \sum_{j=1}^{n} x_{i,j} = 1 \ \& \\ & (\forall i, j \in \{1, \ldots, n\}) x_{i,j} \in \{0, 1\} \end{array}$$

The (indicator) variable $x_{i,j}$ is equal to 1 when agent i is assigned to serve task j and 0 otherwise.

To find the optimal task assignment is NP-hard, although there are many polynomial time algorithms available to compute approximate solutions. We shall employ the well-known Hungarian algorithm (or Kuhn-Munkres algorithm) whose time complexity is $O(n^3)$ [10].

Remark 1. While the above formulation of the task assignment requires that the number of agents be the same as the number of tasks, the more general case where the numbers are different can be easily addressed. If (without loss of generality) the number of agents is greater than the number of tasks, we simply need to add 'dummy' tasks to match up the numbers and these 'dummy' tasks should have significantly high costs so that they will never be chosen.

III. Online multi-agent supervisory control for prioritized tasks

Consider that tasks are divided into higher-priority ones and lower-priority ones. We use three queues Q_0, Q_1, Q_2 to store these tasks, and initialize Q_0 to be empty, Q_1 the set of higher-priority tasks, and Q_2 the set of lower-priority tasks. Further, each higher-priority (resp. lower-priority) task is associated with a timer t_h (resp. t_l); t_h, t_l are positive integers such that $t_h \leq t_l$. The timers decrement by one after every occurrence of $k(\geq 1)$ events. Tasks in Q_1 and Q_2 whose timers become zero will be moved to the highest-priority queue Q_0.

Consider n agents (these may be real agents or virtual ones due to heterogeneous payloads). We can now present the new online supervisor control approach that addresses tasks with different priorities.

(1) Initialize three queues Q_0, Q_1, Q_2 such that Q_0 is the emptyset, Q_1 the set of higher-priority tasks, and Q_2 the set of lower-priority tasks. Each task in Q_1 has timer t_h, and each task in Q_2 has timer t_l.
(2) Collect the first n tasks from the three queues in the order Q_0, Q_1, Q_2. As mentioned in Remark 1, it is without loss of generality to consider that there are at least n tasks. Remove the collected tasks from the queues.
(3) Use the Hungarian algorithm [10] to compute an optimal task assignment such that each agent $i(\in \{1, \ldots, n\})$ obtains one task.
(4) Compute for each agent $i(\in \{1, \ldots, n\})$ the shortest paths for accomplishing the assigned task.
(5) Create the finite state automata $\mathbf{G}_1, \ldots, \mathbf{G}_n$ based on the shortest paths of each agent.
(6) Create a control specification model $\mathbf{SPEC}$ (also a finite state automaton) that imposes a behavioral constraint on the multi-agent system.
(7) Based on the agent models $\mathbf{G}_1, \ldots, \mathbf{G}_n$ and the specification model $\mathbf{SPEC}$, compute by the standard SCT [7] a supervisor $\mathbf{SUP}$. This $\mathbf{SUP}$ ensures safe (i.e. the specification is satisfied) and nonblocking controlled behavior.
(8) After every occurrence of $k(\geq 1)$ events, decrement the timers of all the tasks in Q_1 and Q_2 by one. If a task's timer becomes zero, move the task to Q_0 (with zero timer). If a higher-priority (resp. lower-priority) task newly appears, add the task to Q_1 with timer t_h (resp. Q_2 with timer t_l).
(9) Return to (2) whenever there are unassigned tasks (i.e. at least one queue is nonempty) *and* available agents for assignment.

IV. Case study

We demonstrate how to apply the proposed online multi-agent supervisory control procedure to model and control a warehouse logistic system automated by multiple mobile robots with prioritized tasks.

A. Warehouse Environment

Different warehouses have different configurations. For a concrete case study, we consider the grid-type layout as displayed in Fig. 1. Mobile robots are assumed to be initially waiting for tasks at the top area, items to be picked up stored on storage shelves in the black-rectangle areas, and item-delivery destination locations at the bottom.

The occurrence of tasks is uncontrolled: when and where they occur are completely unknown *a priori*. We consider two types of tasks: higher-priority ones and lower-priority ones, and they are stored in two separate queues Q_1 and Q_2, respectively. We assume that the number of higher-priority

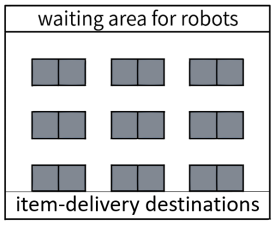

Fig. 1. Warehouse grid environment: items to be picked up are stored in black-rectangle areas

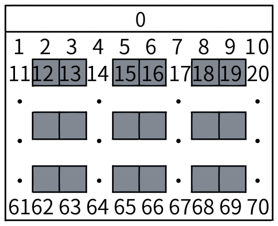

Fig. 2. Warehouse grid assigned with numbers

(resp. lower-priority) tasks never exceeds the capacity of Q_1 (resp. Q_2), and for simplicity that one robot can be assigned exactly one task.

All robots initially wait in the top waiting area and only move when they are assigned a task. Robots are allowed to enter the storage areas only when they are fetching their assigned items. After retrieving the items, the robots move to their designated delivery destination areas at the bottom.

If a robot completes a task, it will be either assigned a new task (if one has appeared and not yet served) or controlled to return to the top waiting area.

B. Automata Models of Robots

We start by assigning sequential (state) numbers to the warehouse as shown in Fig. 2. Specifically, we assign 0 to the waiting area, and the other areas (or cells) are natural numbers starting from the top left corner. When a task is assigned to a robot, one of the delivery areas numbered $61,\dots,70$ will be the marker (or goal) state. In the case where no task is assigned to a robot (number of tasks is smaller than number of robots), 0 is the robot's marker state. Each robot shall move in one of the four directions: up, down, left and right. All robots are assumed to be initially located in the waiting area and eventually return to the waiting area after finishing all assigned tasks.

Consider $n(>1)$ robots serving the warehouse. Each of these n robots has an automaton model $\mathbf{G}_i$ $(i \in \{1,\dots,n\})$:

$$\mathbf{G}_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i}).$$

Here Q_i is a set of states on the paths of robot i (using the numbers assigned to the warehouse as in Fig. 2). Σ_i is a set of four events, given in Table I. All events are assumed to be controllable. δ_i is the state transition function defined according to the paths of robot i; $q_{0,i}$ is the initial state (i.e. the starting point of the robot's paths); and $Q_{m,i}$ is the set of marker states (i.e. the ending points of the robot's paths).

TABLE I
EVENT NUMBERS OF EACH ROBOT $i \in \{1,...,N\}$

go up	$i \times 10 + 1$
go right	$i \times 10 + 3$
go down	$i \times 10 + 5$
go left	$i \times 10 + 7$

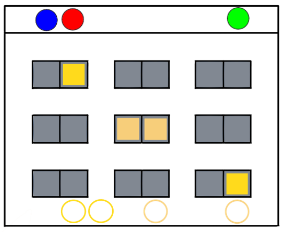

Fig. 3. Initial arrangement of the three robots (discs: red R_1, blue R_2, green R_3) and the four tasks (yellow squares: items; yellow circles: destinations)

The paths needed to obtain the automaton model $\mathbf{G}_i$ are computed as follows. There are two cases.

(Case 1) When robot i is assigned a task, compute (i) the shortest paths from its current location to the item location; and (ii) the shortest paths from the item location to the delivery area. In this case, $q_{0,i}$ is the state number of the robot's current location, and $Q_{m,i}$ is the singleton subset of state numbers $61,\dots,70$, which is the destination area of the item delivery.

(Case 2) When a robot finishes its assigned task and is not assigned a new task, compute the shortest paths from the robot's current position to the top waiting area. In this case, $q_{0,i}$ is the state number of the robot's current location, and $Q_{m,i}$ is the singleton set $\{0\}$.

C. Example of Three Robots

Consider three robots R_1, R_2, R_3 and four tasks T_1, T_2, T_3, T_4. Two tasks T_1, T_4 are of higher-priority, while the other two tasks T_2, T_3 are of lower-priority. The initial arrangement of the robots and tasks is shown in Fig. 3. All robots are initially in state 0, the two higher-priority tasks are located at 13, 59, and the two lower-priority tasks at 35, 36.

The first step of the proposed online supervisory control is to initialize three queues:

$$Q_0 = \emptyset, \quad Q_1 = \{T_1, T_4\}, \quad Q_2 = \{T_2, T_3\}.$$

The two higher-priority tasks stored in queue Q_1 are assigned with timer $t_h = 3$; whereas the two lower-priority tasks stored in queue Q_2 are assigned with timer $t_l = 6$.

In Step 2, we collect $n = 3$ tasks, including the two from Q_1 and one from Q_2 (say T_2). One lower-priority task (T_3) in Q_2 is left out from assignment at this time. After this

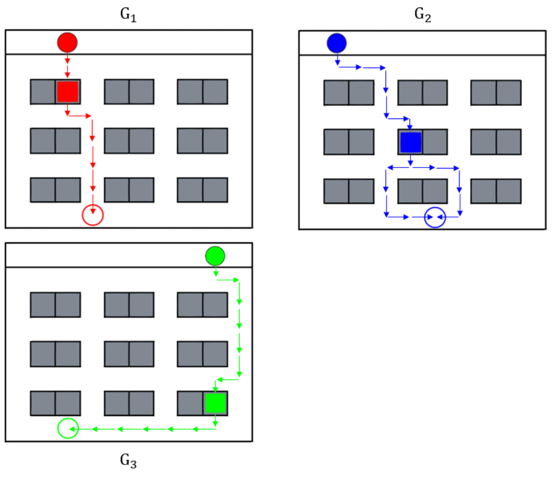

Fig. 4. Shortest paths of the three robots

selection, the three queues are

$$Q_0 = \emptyset, \quad Q_1 = \emptyset, \quad Q_2 = \{T_3\}.$$

In Step 3, we optimally assign the three tasks T_1, T_2, T_4 to the three robots. For this, we generate the 3×3 cost matrix C as follows:

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} = \begin{bmatrix} 2 & 6 & 12 \\ 3 & 7 & 13 \\ 8 & 8 & 8 \end{bmatrix}$$

The entries of C are the (unweighted) distances between the robots and the items (these distances can be weighted to encourage go-straight movements). Having matrix C, we apply the Hungarian algorithm to derive

$$C^* = \begin{bmatrix} 0 & 0 & 6 \\ 0 & 0 & 6 \\ 4 & 0 & 0 \end{bmatrix}$$

Observe that in C^* one can select 0 from each row and each column without duplication: namely $(1,1),(2,2),(3,3)$. These 0-entries yield an optimal task assignment: : T_1 is assigned to R_1, T_2 to R_2, and T_4 to R_3.

In Step 4, we compute the shortest paths for each robot from its initial location to item location and then from item location to destination (Fig. 4).

In Step 5, we create automata models for the robots based on the computed shortest paths. Let the automaton of R_1 be $\mathbf{G}_1$, the automaton of R_2 be $\mathbf{G}_2$, and the automaton of R_3 be $\mathbf{G}_3$, respectively.

As the control specification, in Step 6 we impose mutual exclusion on each cell of the grid so that the robots do not collide with one another (i.e. safety). One exception is the waiting area state 0: we assume that this area is large enough such that multiple robots can be at state 0 at the same time.

In Step 7, we employ the standard SCT to compute a supervisor that satisfies the safety control specification. The resulting supervisor is guaranteed to be safe and nonblocking (the latter ensures all tasks are eventually accomplished).

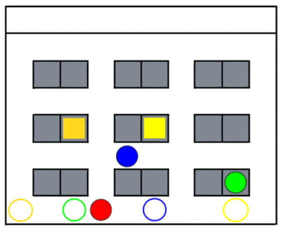

Fig. 5. Robot R_1 (red disc) finishes its assigned task and a new higher-priority task T_5 appears (darker yellow square)

The computed supervisor executes its control decisions online according to (1); accordingly the robots executes their events enabled by the supervisor. In Step 8, we decrement the timer of task T_3 in Q_2 by one after every occurrence of 3 events (ideally each robot executing one event). Then after 18 events have occurred, the timer of T_3 ticks down from 6 to 0, and we move T_3 from Q_2 to Q_0; namely

$$Q_0 = \{T_3\}, \quad Q_1 = \emptyset, \quad Q_2 = \emptyset.$$

When robot R_1 finishes delivering its assigned item to the destination, as shown in Fig. 5, it becomes available again to serve new tasks. Suppose that there have also newly appeared a higher-priority task T_5 (at location 33), so that

$$Q_0 = \{T_3\}, \quad Q_1 = \{T_5\}, \quad Q_2 = \emptyset.$$

Although the task T_3 was originally of lower-priority, its timer has become zero meaning that it has to be served next with the highest priority. Hence, the online procedure will return to Step 2 and assigns the only available robot R_1 to serve task T_3 (instead of T_5).

References

[1] Y. Tatsumoto, M. Shiraishi, and K. Cai, "Application of supervisory control theory with warehouse automation case study," *Trans. ISCIE*, vol. 62, no. 6, pp. 203–208, 2018.
[2] M. Shiraishi, Y. Tatsumoto, K. Cai, and Z. Lin, "Online supervisory control of multi-agent discrete-event systems with warehouse automation case study," in *Proceedings of the SICE Annual Conference*, 2018, pp. 1059–1062.
[3] Y. Tatsumoto, M. Shiraishi, K. Cai, and Z. Lin, "Application of online supervisory control of discrete-event systems to multi-robot warehouse automation," *Control Engineering Practice*, vol. 81, pp. 97–104, 2018.
[4] K. Cai, "Warehouse automation by logistic robotic networks – a cyber-physical control approach," *Frontiers of Information Technology & Electronic Engineering*, vol. 21, pp. 693–704, 2020.
[5] M. Kasahara and K. Cai, "Online supervisory control with optimal task assignment for efficient and adaptive warehouse automation," in *Proc. the 63rd Japan Joint Automatic Control Conf.*, 2020, pp. 90–93.
[6] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.
[7] W. M. Wonham and K. Cai, "Supervisory Control of Discrete-Event Systems," *Springer*, 2019.
[8] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–19, 2008.
[9] S.-L. Chung, S. Lafortune, and F. Lin, "Limited lookahead policies in supervisory control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1921–1935, 1992.
[10] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.