Learning Multiple Coordinated Agents under Directed Acyclic Graph Constraints

Jaeyeon Jang
The Catholic University of Korea

Diego Klabjan
Northwestern University

Han Liu
Northwestern University

Nital S. Patel
Intel, Corporation

Xiuqi Li
Intel, Corporation

Balakrishnan Ananthanarayanan
Intel, Corporation

Husam Dauod
Intel, Corporation

Tzung-Han Juang
Northwestern University

A Proof of Theorem 1

For simplicity, let $g_{ik} = \sum_{t=0}^{\infty} \gamma^t f_{ik}((s_t^j, a_t^j)|j \in \Delta(k)) r^k(s_t^k, \{a_t^j|j \in \Delta(k)\})$. We have

$$\begin{aligned}
&\sum_{i\in\mathcal{V}} \tilde{V}_i^{\{\pi^j|j\in\Omega(i)\}}(s_0^i) \\
&= \sum_{i\in\mathcal{V}} \mathbb{E}_{\{\pi^j|j\in\Omega(i)\}}\left[\sum_{t=0}^{\infty} \gamma^t s r^i((s_t^j, a_t^j)|j \in \Omega(i))\right] \\
&= \sum_{i\in\mathcal{V}} \mathbb{E}_{\{\pi^j|j\in\Omega(i)\}}\left[\sum_{t=0}^{\infty} \gamma^t \sum_{k\in\mathcal{L}\cap\Upsilon(i)} f_{ik}((s_t^j, a_t^j)|j \in \Delta(k)) r^k(s_t^k, \{a_t^j|j \in \Delta(k)\})\right] \\
&= \sum_{i\in\mathcal{V}} \mathbb{E}_{\{\pi^j|j\in\Omega(i)\}}\left[\sum_{k\in\mathcal{L}\cap\Upsilon(i)} g_{ik}\right]. \qquad (1)
\end{aligned}$$

Since g_{ik} has dependency only on $\{\pi^j|j \in \Delta(k)\}$ and $f_{ik}((s_t^j, a_t^j)|j \in \Delta(k))$, we further have

$$\begin{aligned}
&\sum_{i\in\mathcal{V}} \mathbb{E}_{\{\pi^j|j\in\Omega(i)\}}\left[\sum_{k\in\mathcal{L}\cap\Upsilon(i)} g_{ik}\right] \\
&= \sum_{i\in\mathcal{V}} \sum_{k\in\mathcal{L}\cap\Upsilon(i)} \mathbb{E}_{\{\pi^j|j\in\Delta(k)\}}[g_{ik}] \\
&= \sum_{k\in\mathcal{L}} \mathbb{E}_{\{\pi^j|j\in\Delta(k)\}}\left[\sum_{i\in\Delta(k)} g_{ik}\right] \\
&= \sum_{k\in\mathcal{L}} \mathbb{E}_{\{\pi^j|j\in\Delta(k)\}}\left[\sum_{t=0}^{\infty} \gamma^t \sum_{i\in\Delta(k)} f_{ik}((s_t^j, a_t^j)|j \in \Delta(k)) r^k(s_t^k, \{a_t^j|j \in \Delta(k)\})\right] \\
&\leq \sum_{k\in\mathcal{L}} \mathbb{E}_{\{\pi^j|j\in\Delta(k)\}}\left[\sum_{t=0}^{\infty} \gamma^t r^k(s_t^k, \{a_t^j|j \in \Delta(k)\})\right] \\
&= \sum_{k\in\mathcal{L}} V_k^{\{\pi^j|j\in\Delta(k)\}}(s_0^k). \qquad (2)
\end{aligned}$$

Preprint. Under review.

Finally, by combining (1) and (2), we derive

$$\sum_{i\in\mathcal{V}} \tilde{V}_i^{\{\pi^j | j\in\Omega(i)\}}(s_0^i) \leq \sum_{i\in\mathcal{L}} V_i^{\{\pi^j | j\in\Delta(i)\}}(s_0^i). \tag{3}$$

B Overview of the algorithm

In this section, an overview of the proposed training algorithm for MARLM-SR is described. The algorithm consists of the outer and inner settings as shown in Fig. 1. In Fig. 1(a), in the inner setting, the followers perform their subtasks in a DAG every time step. In Fig. 1(b), in the outer setting, two different types of agents are trained to guide the followers to achieve a high team reward. First, the leader provides goals to followers in each goal period based on the global state at the beginning of a goal period. After each goal period, reward generator and distributor (RGD) provides synthetic rewards to guide the followers to have a better exploration based on their achievements within the goal period. Here, a single agent plays both the roles of the reward generator and the reward distributor. Specifically, the reward generator generates a synthetic reward and the reward distributor distributes the produced synthetic reward from the highest level followers (sinks) to the lowest level followers following the DAG based on the achievement of the followers within the goal period. The team reward based on the follower's achievements in the episode is given to all outer agents. In other words, these three outer agents must learn the policy to produce and distribute the goals and the synthetic reward that coordinates the followers. On the other hand, in the inner setting, the followers learn policies for high team and synthetic rewards by following the given goals well. If the followers are guided well based on the policies of the outer agents and a high team reward is achieved, this high team reward is given not only to the followers but also to the outer agents.

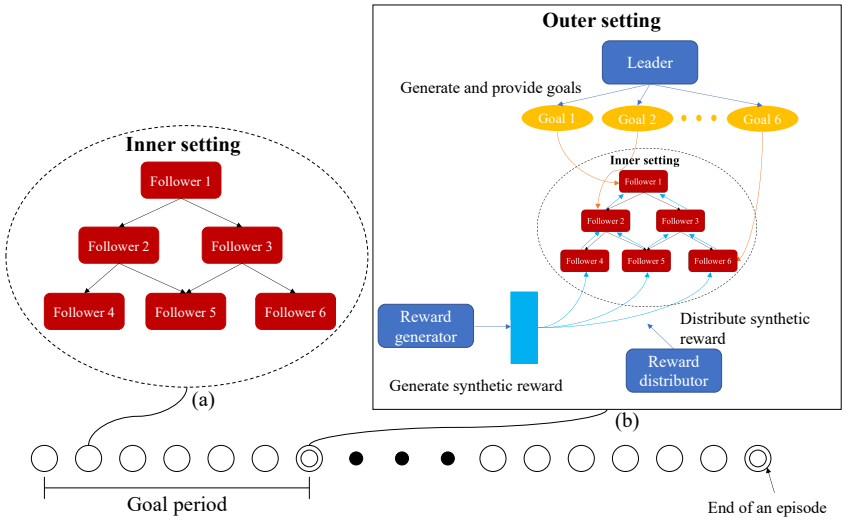
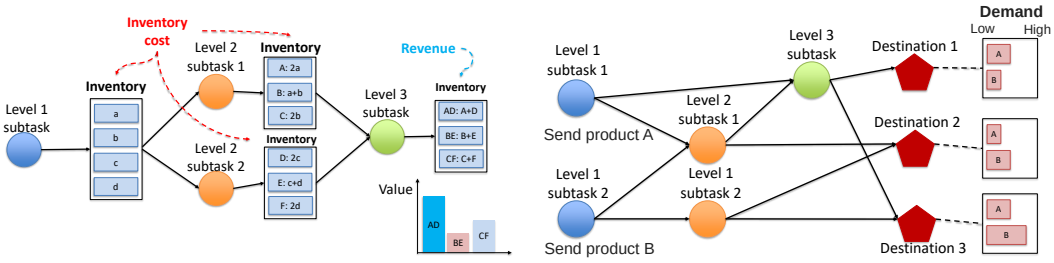


Figure 1: An overview of the proposed training algorithm.

C Experimental Environments

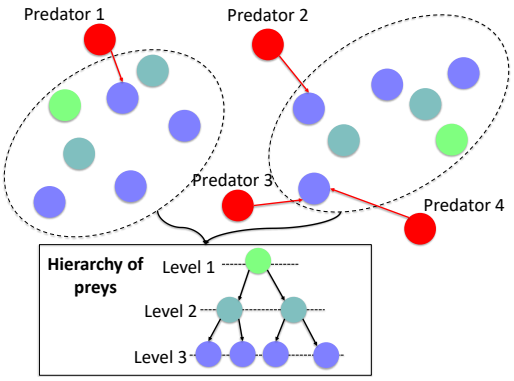
We evaluated our model in diverse and challenging environments with DAG constraints including a real-world scheduling task. First, we created three environments to simulate DAG systems as shown in Fig. 2.

Factory production planning case. As shown in Fig. 2(a), we construct a DAG with four nodes and a depth of three in this case. In each episode, across ten goal periods with each lasting 40 steps, the objective of the four agents is to maximize revenue by producing final products in accordance with both the product's value and the demand. The agents must cooperate to achieve this objective, given that the values of the final products change every goal period. Additionally, costs are incurred from both overproduction and parts that remain unused in the final products. In every goal period, each final product is assigned a value of 2, 3, or 4 uniformly at random with each final product having a



(a) Factory production planning case

(b) Logistics case



(c) Hierarchical predator-prey case

Figure 2: Illustrations of the use cases. Cooperation is necessary (a) to achieve a high profit, (b) to satisfy demands while minimizing costs, and (c) to avoid being caught by predators for as long as possible.

distinct value. We have set these values to create a significant difference in the value of each product. For instance, the most valuable product holds a value twice that of the least valuable one. The agents should cooperate to prioritize the production of the most valuable product first. In addition, the three products collectively share a total demand of 10 in each goal period, distributed randomly. Ten is the maximum number of final products that can be produced from scratch within a single goal period. We randomly set the product values and demand to reflect a dynamic production and market environment.

In each subtask, a machine may or may not produce one type of a product. If a machine chooses to produce a product, it consumes required parts, which are denoted by the blue boxes in the inventories, and stores the produced product in its own inventory. For example, if the machine for level 2 subtask 1 produces 'B' and stores it, it consumes 1 part 'a' and 1 part 'b.' A machine cannot produce a product without the necessary parts. They are subject to bill of material (except the machine in level 1). Final products are made after processing all three levels. Revenue is the cumulative value of the produced final products after an episode, and the factory is rewarded according to this revenue. Meanwhile, the factory is penalized by inventory holding costs of stored parts. Inventory holding costs of 0.3 and 0.8 are assigned to each part in the inventory of level 1 and level 2. Regarding overproduction, a penalty of 1 is imposed for each final product that is overproduced. We set two holding costs and the overproduction cost randomly, but the cost imposed at a higher level is set to be greater than that at a lower level.

Logistics case. In this case, the subtasks are to send a product either 'A' or 'B' at a time step. Similar to the factory case, each node must have a product in the inventory given from lower level nodes in order to send the product. Only the nodes in level 1 can send a product without constraint, but they can provide only one type of a product. Each node can choose not to send any product as well. If a product passes through all intermediate nodes in the graph, it arrives at one of the three destinations that have specific demands for both 'A' and 'B.' In summary, as shown in Fig. 2(b), we have a three-level DAG with five nodes, excluding the three destinations. This is because no tasks are required at these destinations.

Each episode consists of 300 time steps with 30 goal periods. In this case, we introduce randomness into both the demand for the two products at the destinations and the shipping cost on each arc to simulate a dynamic logistics scenario. The demand and the shipping cost vary in every epoch. Specifically, we assign a shipping cost, drawn from a uniform distribution [0, 0.3], to each arc in order to create subtle differences between them. Additionally, the demand follows the uniform distribution with the lower and upper bounds in Table 1. We categorize destinations into high, medium, and low demand to diversify demand based on the difficulty of reaching each one. For instance, to reach destination 1, a product needs to traverse 1.67 arcs, while destination 2 requires an average of 2.00 arcs, and destination 3 requires an average of 2.50 arcs. At each destination, benefits are granted if the demands are met. Specifically, benefits of 100, 300, and 200 are set for destinations 1, 2, and 3, respectively. However, additional inventory holding cost or inventory shortage cost is given if the demand is not met at each destination, and inventory holding cost is imposed at each node in the graph as well. The inventory holding cost at levels 1, 2, and 3 is 0.3. We set the maximum shipping cost and the inventory holding cost at each node to 0.3. This allows them to impact the total benefit, but not significantly. Meanwhile, we set the additional inventory holding cost at the destinations to 3, and the inventory shortage cost at the destinations to 8, so that they significantly influence the total benefits. The inventory shortage cost is set higher than the inventory holding cost, which encourages agents to prioritize shipping over holding inventory. All costs are deducted from the total benefits to calculate the team rewards. Consequently, the maximum reward amounts to 600.

Table 1: The lower and upper bounds of the uniform distribution for each product at the three destinations in the logistics case

Destination	Product	Lower bound	Upper bound
Destination 1	A	5	10
(Low demand)	B	3	7
Destination 2	A	110	130
(High demand)	B	70	90
Destination 3	A	35	45
(Medium demand)	B	80	100

Hierarchical predator-prey case. In this variant of the classic predator-prey game, the preys have a hierarchy in which a higher level prey follows the parent prey (refer to the structure of the DAG in Fig. 2(c)). The objective of this game is to guide the prey at the sink nodes to evade predators and survive for as long as possible. Each time predators move one step to chase the preys in the highest level. The preys also can move one step to move away from the predators. A prey can choose to move in any direction, but cannot go beyond the boundary set by the parent prey. The boundary is 5 steps away from the parent prey. The total survival time of the preys at the sink nodes is given as the team reward after each episode. In other words, the later the preys in the highest level are caught, the higher rewards are given to the set of the preys. Thus, the parent preys must guide their children well to provide safe areas to the preys in the highest level. An episode concludes when all the prey at the sink nodes are caught, or when the maximum duration of 200 steps is reached. We set the length of the goal period to 10. In each episode, the predators and the preys start at fixed positions, but the predators randomly select their direction.

Real-world production planning case. We also investigated the performance of the proposed algorithm in a real-world scheduling environment. For this environment, we developed a simulator that sets a reasonable demand goal for each product type within a given time period based on the actual production information of one of Intel's high volume packaging and test factory (AT factory). The purpose of this task is to schedule jobs so that each job goes through a specific sequence of operations defined by product type. In this scheduling environment, the precedence constraints of operations are not different according to product type, even though each product type requires a different set of operations. For example, if operation 'b' comes after operation 'a' for a product 'A,' 'b' cannot come earlier than 'a' for any product. Thus, the unique DAG is built based on these precedence constraints. Several stations are assigned for an operation and one station can only process one operation. Thus, we group stations by target operation, and each group of stations is called a station family. Even members in the same station family have a different set of products available. In addition, the processing time of a product significantly varies depending on the type of operation. In summary, jobs must be scheduled to meet demand goals while considering all these operational

constraints and relationships between operations. The AT factory is a large-scale line that contains more than 75 stations, 10 operations, and 35 product types. This scheduling task is very challenging.

In this study, we simulated an environment where jobs need to be scheduled for five shifts, with each shift lasting 12 hours (refer to Fig. 3). In the figure, only nine product types are present (right vertical legend). The left vertical axis corresponds to stations. It shows that not all products are processed in the same shift, but the AT factory has to fabricate a different set of products each shift to meet ever-changing demands. In addition, the AT factory has a strict constraint that for most station families only one product conversion is allowed in a shift. In other words, with the exception of a few families that can execute multiple conversions, the other families have only one opportunity to select a station and change product type for the selected station in a shift. Thus, it is catastrophic if an agent makes a wrong conversion decision because the waiting jobs that are not current setup will not have a chance to be processed for a long time. Therefore, an agent makes a conversion decision for the assigned station family, including which station will perform the conversion and what the next product type will be.

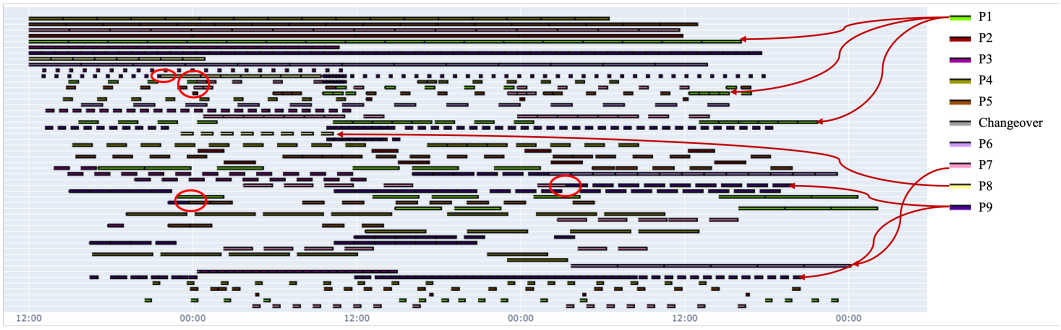


Figure 3: An example Gantt chart for a five shift schedule. We use arrows to distinguish a few products, and we highlight some changeovers with a red circle.

D Hyperparameter settings

We implemented the proposed algorithm and the comparison algorithms by using the proximal policy optimization (PPO) algorithm as the baseline learning algorithm for each agent in all MARL algorithms. A fully-connected neural network with 2 layers of size 256 with ReLU activations is applied to both actor and critic of each agent. Entropy regularization [1] is applied with a coefficient of 0.01. We summarize the other hyperparameters for RL in Table 2. For the three artificial benchmark cases, the step size k used to obtain the global state flow (GSF) gsf_l at the l-th goal period is set to three. In the real-world production planning case, we include only the initial and final global state of each goal period in the GSF.

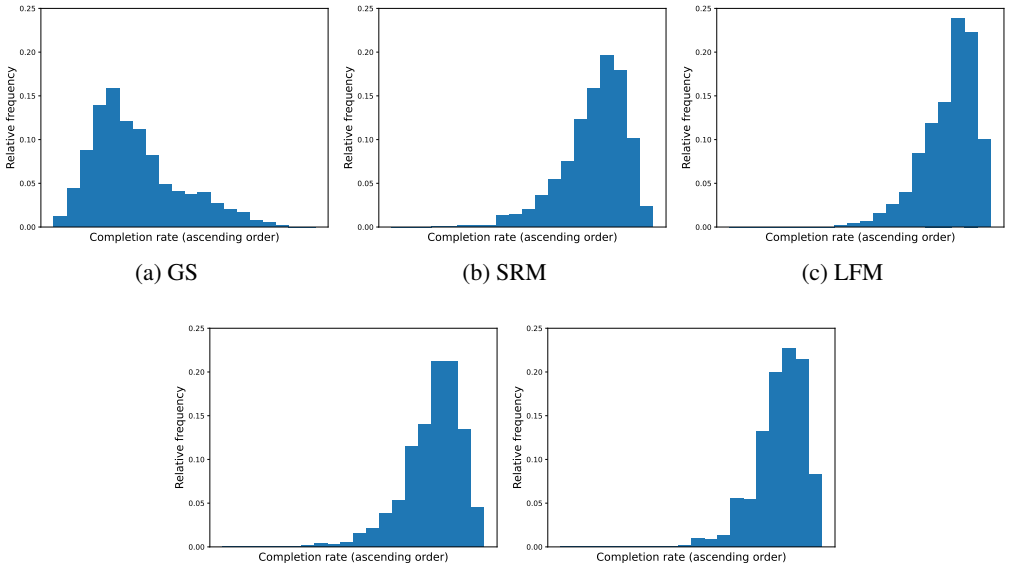
Table 2: Hyperparameters for our model and the baselines.

Hyperparameter	Value
Episode length	1,200
Batch size	256
Learning rate	0.0001
Discount factor	0.9900
Clipping value	0.2000
Generalized advantage estimation parameter lambda	0.9500

E Ablation analysis in scheduling scenarios

We compared the five baselines for the ablation study in diverse scheduling scenarios. Specifically, we trained the agents in the DAG using the five baseline algorithm: GS, SRM, LFM, RFM, and the proposed algorithm, and simulate 1,000 scheduling episodes using the trained models. Fig. 4 shows the histogram of the completion rate on 1,000 episodes for each baseline, where each baseline

has the same x-axis values. Here, the completion rate is the ratio of the lots that pass all required operations to demanded lots. We do not report the average value of the completion rate and the range of the histogram for confidentiality. First, by introducing the concept of MARL, we were able to improve the mean completion rate by 148.7%. This component contributes most significantly to the performance improvement. A comparison between SRM and LFM/RFM reveals that the leader and the RGD also contribute to performance improvement. Specifically, the RGD improves the mean completion rate by 3.9%. Furthermore, by introducing both outer agents, we are able to improve the mean completion rate by 8.5%. As a result, the proposed algorithm demonstrates a higher overall completion rate compared to other baselines. It achieves an improvement of 169.6% in the mean completion rate compared to GS.



(a) GS (b) SRM (c) LFM

(d) RFM (e) Proposed

Figure 4: The histogram of completion rate over 1,000 scheduling scenarios (episodes) for the ablation study.

References

[1] J. Schulman, P. Abbeel, and X. Chen. Equivalence between policy gradients and soft q-learning. *arXiv:1704.06440*, 2017.