# LEARNING SCALABLE AND TRANSFERABLE MULTI-ROBOT/MACHINE SEQUENTIAL ASSIGNMENT PLANNING VIA GRAPH EMBEDDING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Can the success of reinforcement learning methods for simple combinatorial optimization problems be extended to multi-robot sequential assignment planning? In addition to the challenge of achieving near-optimal performance in large problems, transferability to an unseen number of robots and tasks is another key challenge for real-world applications. In this paper, we suggest a method that achieves the first success in both challenges for robot/machine scheduling problems.

Our method comprises of three components. First, we show a robot scheduling problem can be expressed as a random probabilistic graphical model (PGM). We develop a mean-field inference method for random PGM and use it for $Q$-function inference. Second, we show that transferability can be achieved by carefully designing two-step sequential encoding of problem state. Third, we resolve the computational scalability issue of fitted $Q$-iteration by suggesting a heuristic auction-based Q-iteration fitting method enabled by transferability we achieved.

We apply our method to discrete-time, discrete space problems (Multi-Robot Reward Collection (MRRC)) and scalably achieve 97% optimality with transferability. This optimality is maintained under stochastic contexts. By extending our method to continuous time, continuous space formulation, we claim to be the first learning-based method with scalable performance among multi-machine scheduling problems; our method scalability achieves comparable performance to popular metaheuristics in Identical parallel machine scheduling (IPMS) problems.

## 1 INTRODUCTION

Imagine we are given a set of robots and want to service a set of spatially distributed tasks. A reward is given when serving each task fast (decaying reward collection), or when finishing the whole task fast (makespan minimization). As physical control and routing of individual robots are becoming highly capable [Li (2017)], efficient *orchestration* of robots arises as the important remaining issue. Recent advances in probabilistic inference [Agha-mohammadi et al. (2014)] allows us to infer the probabilistic distribution of time required for controlling and routing a robot to complete a task. We will call the random variable with this probability distribution *task completion time*. Using task completion time information, this orchestration problem can be formulated as a *stochastic scheduling planning problem with sequential decision making* [Omidshafiei et al. (2017)]. As pointed in [Omidshafiei et al. (2017)], one of the most important bottleneck in this problem is inscalability; as the number of robots and tasks increases, existing methods suffer from either exponentially increasing computation time or significant performance loss [Rossi et al. (2018)].

Recently, there has been some seminar works in learning-based algorithms that scalably solve traveling salesman problem (TSP) [Bello et al. (2016); Dai et al. (2017); Kool et al. (2018)]. They formulate TSP as single-robot sequential assignment decision-making problems minimizing traveling cost. However, there has not yet been reported success in multi-robot scheduling problem except for special cases when the problem can be extended as a variant of single-robot traveling cost minimization method with multiple routing of a single robot [Nazari et al. (2018)].

**Near-optimal performance with scalability.** In this paper, we introduce a general type of problem called Multi-Robot Reward Collection (MRRC) problems and show that this type of problems can be scalably solved by our method with near-optimal performance. While learning-based methods are generally believed to suffer exponentially increasing training requirements as problem size (number of robots and tasks) increases, our method's training requirement is empirically shown not to scale while maintaining near-optimal performance. While MRRC was simulated under discrete state/time space in this paper, this success extends to problems with continuous state/time space; our method claims to be the first learning-based method with scalable performance among machine scheduling problems; our method achieves scalability (e.g. 10 machines, 100 tasks) with comparable performance to popular heuristics in Identical Parallel Machine Scheduling (IPMS).

**Transferability.** In addition to scalability, we claim that our method possesses transferability. Transferability means that a trained policy can be applied to new environments with a small loss of performance [Wang et al. (2018)]. In real-world applications, engineers aren't given a whole real system for training, but rather given a small testbed environment for training; even if the whole system is given for training, system design or the number of robots would be frequently changed during operation. We show that our method achieves transferability with almost no performance loss.

Our method stems from the work of [Dai et al. (2017)]. In this seminar paper, they observe that combinatorial optimization problems such as TSP can be formulated as a sequential decision making problem: at each decision step, they grow a partial solution $s$ by choosing increment $a$ with minimum future cost estimate $Q(s, a)$. Under this observation, [Dai et al. (2017)]'s solution framework can be described by the following three aspects. 1) For each combinatorial optimization problem, one can carefully find a heuristic way to induce a graph representation of $(s, a)$ (in case of TSP, the paper induces a fully connected graph for every possible next increment). 2) This induced graph representation can be considered as a probabilistic graphical model (PGM) [Koller & Friedman (2009)]). This model can be used for a graph-based mean-field inference method called structure2vec [Dai et al. (2016)] to infer $Q(s, a)$ and solve a combinatorial optimization problem. 3) Inference of $Q(s, a)$ can be learned by reinforcement framework called fitted Q-iteration.

Our method shares the overall solution framework with [Dai et al. (2017)], but designed specifically to achieve scalability and transferability in multi-robot sequential assignment decision making problems. To be specific, our method differs from [Dai et al. (2017)] in the following aspects:
**1. State representation and theory of mean-field inference with random PGM.** Instead of heuristically inducing a PGM, we analytically show that a robot scheduling problem exactly induces a type of *random* PGM. Since there exists no mean-field inference theory for random PGM, we develop the theory and corresponding new structure2vec equation with simple extra inference task.
**2. Sequential encoding of information for transferability.** To achieve transferability in terms of the number of robots and tasks, we carefully design a two-step hierarchical mean-field inference [Ranganath et al. (2015)]. Each inference step is designed to infer certain information: the first step is designed to infer 'each task's relative graphical distance from robots'; the second step is designed to infer $Q(s, a)$ ($a$ here means a joint assignment of robots). While the first step is by its nature transferable to any number of tasks and robots, the transferability in inference of the second step is achieved by scale-free characteristic of fitted Q-iteration [van Hasselt et al. (2015)]; relative magnitudes of $Q(s, a)$ values instead of their actual magnitudes is important in selecting action $a$.
**3. Auction-based assignment.** Even if we can infer $Q(s, a)$ precisely, the computation time for finding action $a$ with maximum $Q(s, a)$ exponentially increases as robots and task increases. To solve this issue, we suggest a heuristic auction that is enabled by transferability of $Q(s, a)$ inference. While this heuristic auction selects $a$ with polynomial computation complexity, it also gives a surprisingly good choice of $a$ (In fact this heuristic auction increases the performance empirically).
**4. Auction-fitted Q-iteration.** The heuristic auction-based action selection can be incorporated into learning (fitting) $Q(s, a)$ with approximation. To be specific, we use the auction-based action selection scheme, instead of typical max-operator based action selection, in Bellman equation during fitted Q-iteration. We call this new learning framework as *auction-fitted Q-iteration*.

## 2 Multi-robot/machine scheduling problem formulation

Throughout this study, we formulate problems as centralized sequential assignment planning problem: we assume an assignment decision maker who instantaneously achieves necessary information to infer the distribution of all possible task completion times (this indicates perfect communication).

### 2.1 Multi-Robot Reward Collection (MRRC)

Here we formulate MRRC in a discrete-state, discrete-time sequential assignment decision making problem (In appendix A, we discuss how MRRC can be formulated with continuous-state and continuous-time space and can be solved by using our method). The initial location and ending location of robots and tasks are arbitrary on a grid (e.g., grid world). We assume that task is serviced immediately after a robot arrives (In appendix A, we extend how our method can be easily extended to address setup time and processing time). Under these assumptions, at each time-step, we can assign every robot to every remaining task. This problem can be cast as a Markov Decision Problem (MDP) whose state, action, and reward are defined as follows:

**State.** State $s_t$ at time step $t$ is a directed graph $\mathcal{G}_t = (\mathcal{R}_t \cup \mathcal{T}_t, \mathcal{E}_t)$: $\mathcal{R}_t$ is the set of all robots and $\mathcal{T}_t$ is the set of all tasks; The set of directed edges $\mathcal{E}_t = \mathcal{E}_t^{RT} \cup \mathcal{E}_t^{TT}$ where a directed edge $\epsilon_{r_i t_j} \in \mathcal{E}_t^{RT}$ is a random variable which denotes task completion time of robot $i$ in $\mathcal{R}_t$ to service task $j$ in $\mathcal{T}_t$ and a directed edge $\epsilon_{t_i t_j} \in \mathcal{E}_t^{TT}$ denotes a task completion time of a robot which just finished serving task $i$ in $\mathcal{T}_t$ to service task $j$ in $\mathcal{T}_t$.

**Action.** The action $a_t$ at time step $t$ is the joint assignment of robots given the current state $s_t = \mathcal{G}_t$. The feasible action should satisfy the two constraints: No two robots can be assigned to a task; some robots may not be assigned when number of robots are more than remaining tasks. To best address those restrictions, we define an action $a_t$ at time $t$ as a maximal bipartite matching in bipartite subgraph $((\mathcal{R}_t, \mathcal{T}_t), \mathcal{E}_t^{RT})$ of graph $\mathcal{G}_t$. For example, robot $i$ in $\mathcal{R}_t$ is matched with task $j$ in $\mathcal{T}_t$ in an action $a_t$ if we assign robot $i$ to task $j$ at time-step $t$. We denote the set of all possible actions at time-step $t$ as $\mathcal{A}_t$.

**Reward.** In MRRC, Each task has an arbitrarily determined initial age. At each time-step, the age of each task increases by one. When a task is serviced, a reward is determined only by its age when serviced. Denote this reward rule as $R(t)$. One can easily see that whether a task is served at time-step $t$ is completely determined by $s_t$, $a_t$ and $s_{t+1}$. Therefore, we can denote the reward we get with $s_t, a_t$ and $s_{t+1}$ as $R(s_t, a_t, s_{t+1})$.

**Objective.** We can now define an assignment policy $\phi$ as a function that maps a state $s_t$ to an action $a_t$. Given $s_0$ initial state, an MRRC problem can be expressed as a problem of finding an optimal assignment policy $\phi^*$ such that

$$\phi^* = \underset{\phi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{\infty} R\left(s_t, a_t, s_{t+1}\right) | s_0\right].$$

### 2.2 Identical parallel machine scheduling (IPMS) make-span minimization

IPMS is a problem defined in continuous state/continuous time space. Machines are all identical, but processing times of tasks are all different. In this paper, we discuss IPMS with ‘sequence-dependent setup time’. In this case, a machine’s setup time required for servicing a task $i$ is determined by its previously served task $j$. In this case, the task completion time is the sum of setup time and processing time. Under this setting, we solve IPMS problem for make-span minimization objective discussed in [Kurz et al. (2001)]; That is, minimizing the total time spent from start to end to finish all tasks. IPMS formulation resembles MRRC formulation in continuous time space with continuous state space (Appendix B) and we relegate the detailed formulation to the Appendix.

## 3 Robot assignment at each time-step

As in [Dai et al. (2017)], we employ $Q$-function based action choice; at state $s_t$, we choose action $a_t$ using $Q(s_t, a_t)$ inference. However, even if we can infer $Q(s_t, a_t)$ well, $Q$-function based action choice suffers from exponentially increasing computational complexity with the number of agents. This section introduces 1) how we can express $(s_t, a_t)$ as a graph representation 2) how we can make inference about $Q(s_t, a_t)$ in a transferable way using the sequential graph embedding 3) how we can select a joint assignment of robots with polynomial computation complexity by using the heuristic auction and the transferable inference on $Q(s_t, a_t)$.

### 3.1 Random PGM-representation and a new theory for mean-field inference

This section discusses how our problems can be represented as a random probabilistic graphical model (PGM) for structure2vec inference method. While one can heuristically induce a PGM as [Dai et al. (2017)] for structure2vec, we illustrate that a robot scheduling problem induces an exact graphical model we call 'random' PGM. Since there is no mean-field inference theory for random PGM, we develop a new theory and corresponding new structure2vec in this section.

While PGM-based mean-field inference methods require prior knowledge of whole probabilistic relationships (called cliques) of PGM, structure2vec requires only the structure (*existence* of each possible cliques) of PGM with learning of neural networks (in fixed point iterations that surrogates the fixed-point iterations of mean-field inference with PGM). The observation we make is that in robot scheduling problems the *structure of PGM itself is random.* Suppose that we know the future path of all robots from a given state. For each task $t_i \in \mathcal{T}_t$, define a random variable $X_i$ as 'some characteristic of task $t_i$' (e.g. when task $i$ is served). Given the future path, the relationship among $\{X_i\}$ is a well-defined PGM (a Bayesian Network - see Appendix C). However, this future path will be random because of the randomness of task completion times. We thus define 'random' PGM: A probabilistic model of how a PGM is randomly chosen from a set of possible PGMs with $\{X_i\}$ and set of all possible probabilistic relationships $\mathfrak{C}$ (we call them 'semi-cliques'). In robot scheduling problem, a semi-clique $\mathcal{D}_{ij} \in \mathfrak{C}$ is a conditional dependence $X_i|X_j$. The semi-clique $\mathcal{D}_{ij}$ presents as an actual clique if and only if the robot which finishes task $t_i$ chooses task $t_j$ as next task.

Note that inference of probability model for all possible future path of robots with random task completion times is challenging. The following theorem for mean-field inference with random PGM is an extension of mean-field inference with PGM [Koller & Friedman (2009)] and suggests that only a simple inference task is required: inferring the presence probability of each semi-cliques.

**Theorem 1. Mean field inference with random PGM.** Suppose that a PGM is to be randomly chosen among PGMs with $\{X_k\}$ and semi-cliques $\mathfrak{C}$. Denote the presence probability of semi-clique $\mathcal{D}_m \in \mathfrak{C}$ as $p_m$. The latent variable distribution $\{q_k(x_k)\}$ in mean-field inference is locally optimal only if

$$q_k(x_k) = \frac{1}{Z_k} \exp\left\{ \sum_{m: X_k \in \mathcal{D}_m} p_m \mathbb{E}_{(\mathcal{D}_m - \{X_k\}) \sim q} \left[ \ln \phi^m (\mathcal{D}_m, x_k) \right] \right\}$$

where $Z_k$ is a normalizing constant and $\phi^m$ is the probabilistic relationship for clique $m$.

From this new theory, we can derive the structure2vec inference method with random PGM. As in [Dai et al. (2016)], we restrict our discussion to when every semi-clique is between two random variables. In this case, a semi-clique can be written as $\mathcal{D}_{ij}$ with its presence probability $p_{ij}$.

**Lemma 1. Structure2vec for random PGM** Suppose we are given a random PGM with $\{X_k\}$ and semi-cliques $\mathfrak{C} = \{\mathcal{D}_{ij}\}$ with $\{p_{ij}\}$. The fixed point iteration in Theorem 1 for posterior marginal $p(\{H_k\}|\{x_k\})$ can be embedded in a nonliner function mapping with embedding vector $\tilde{\mu_k}$ (which summarizes the latent variable distribution $q_k(x_k)$) and $X_k$ as:

$$\tilde{\mu_k} = \sigma\left( W_1 x_k + W_2 \sum_{j \neq k} p_{kj} \tilde{\mu}_j \right).$$

**Proof of Theorem 1 and lemma 1**. For brevity, proofs are relegated to the Appendix D and E (We use a neural network as a nonlinear function approximator with $W_1$ and $W_2$ trainable parameters)

**Corollary 1.** For a robot scheduling problem with set of tasks $t_i \in \mathcal{T}_t$, the random PGM representation for structure2vec in lemma 1 is $((\mathcal{T}_t, \mathcal{E}_t^{TT}), \{p_{ij}\})$ where $\{p_{ij}\}$ denotes the probability of a robot choosing task $t_i$ after serving $t_j$.

$\{p_{ij}\}$ inference procedure employed in this paper is as follows. Denote ages of task $i, j$ as $age_i$, $age_j$. Note that if we generate M samples of $\epsilon_{ij}$ as $\{e_{ij}^k\}_{k=1}^M$, then $\frac{1}{M}\sum_{k=1}^M f(e_{ij}^k, age_i, age_j)$ is an unbiased and consistent estimator of $E[f(\epsilon_{ij}, age_i, age_j)]$. For each sample $k$, for each task $i$ and task $j$, we form a vector of $u_{ij}^k = (e_{ij}^k, age_i, age_j)$ and compute $g_{ij} = \sum_{k=1}^M \frac{1}{M} W_1(relu(W_2 u_{ij}^k)$. We obtain $\{p_{ij}\}$ from $\{g_{ij}\}$ using softmax. Algorithm details are in Appendix F.

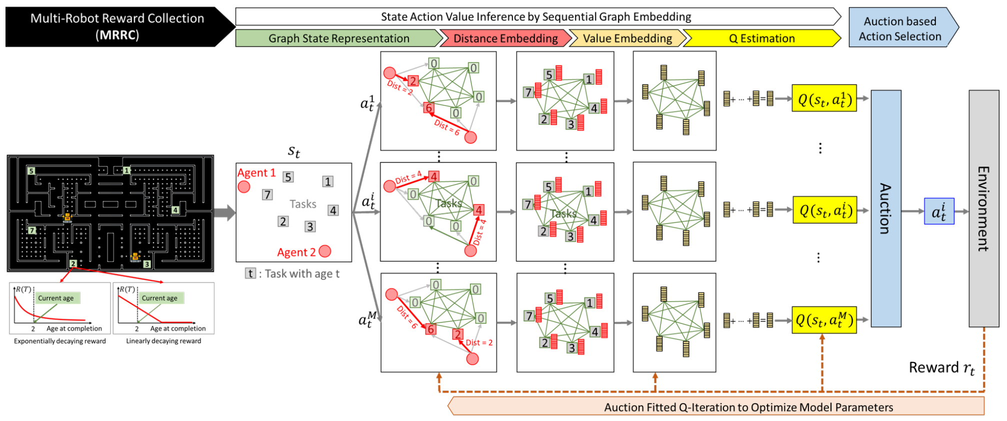

Figure 1: Illustration of overall pipeline of our method

## 3.2 TRANSFERABLE INFERENCE OF Q-FUNCTION USING STRUCTURE2VEC

The key to transferable inference for $Q(s_t, a_t)$ is the two-step sequential state embedding with hierarchical variational inference [Ranganath et al. (2015)]. We here suppose that we are given a graph representation $((\mathcal{T}_t, \mathcal{E}_t^{TT}), \{p_{ij}\})$. Then we can apply Corollary 1 of section 3.1 to infer $Q(s_t, a_t)$. For brevity, we illustrate the inference procedure for the special case when task completion time is not a random variable, but a value in $\mathbb{R}$ (Appendix G illustrates how we can combine random sampling to inference procedure to deal with task completion times as a random variable).

**Step 1. Distance Embedding.** The first step is designed to infer each task's 'relative graphical distance' from all robots. As illustrated in Dai et al. (2016), the output vectors $\{\tilde{\mu}_k^1\}$ of structure2vec *embeds a local graph structure around that vector node.* As the input of first structure2vec ($\{x_k\}$ in lemma 1), we only use robot assignment information (if $t_k$ is an assigned task, we set $x_k$ as 'task completion time of assignment'; if $t_k$ is not an assigned task:, we set $x_k = 0$). This procedure is illustrated in Figure **??**. According to [Dai et al. (2016)], the output vectors $\{\tilde{\mu}_k^1\}$ of structure2vec will include sufficient information about the relative graphical distance from all robots to each task.

**Step 2. Value Embedding.** The second step is designed to infer 'How much value is likely in the local graph around each task'. Remind that vectors $\{\tilde{\mu}_k^1\}$, output vectors of the first step, carries information about the relative graphical distance from all robots to each task. We concatenate 'age' of each tasks $\{age_k\}$ to each corresponding vector in $\{\tilde{\mu}_k^1\}$ and use the resulting graph as an input ($\{x_k\}$ in lemma 1) of second structure2vec, as illustrated in Figure **??**. Again, vectors $\{\tilde{\mu}_k^2\}$ of the output graph of second structure2vec operation embeds a local graph structure around each node. Our intuition is that $\{\tilde{\mu}_k^2\}$ includes sufficient information about 'How much value is likely in the local graph around each task'.

**Step 3. Computing $Q(s_t, a_t)$.** To infer $Q(s_t, a_t)$, we aggregate the embedding vectors for all nodes, i.e., $\tilde{\mu}^2 = \sum_k \tilde{\mu}_k^2$ to get one vector $\tilde{\mu}^2$ which embeds the 'value likeliness' of the global graph. We then use a layer of neural network to map $\tilde{\mu}^2$ into $Q(s_t, a_t)$. The detailed algorithm of above whole procedure (combined with random task completion times) is illustrated in Appendix G.

Why are each inference steps transferable? For the first step, it is trivial; the inference problem is a scale-free task. In the second step, the 'value likeliness' will be underestimated or overestimated according to the ratio of (number of robots/number of tasks) in a local graph: underestimated if ratio in training environment is smaller than the ratio in the testing environment; overestimated otherwise. The key idea solving this problem is that this over/under-estimation does not matter in Q-function based action decision [van Hasselt et al. (2015)] as long as the *order* of Q-function value among actions are the same. While analytic justification of this order invariance is beyond this paper's scope, the fact that there is no over/underestimation issue in the first step inference problem helps this justification.

### 3.3 Action Selection Using Heuristic Auction

In Q-function based action choice, at each time-step $t$, we find an action with largest $Q(s_t, a_t)$. We call this action choice operation 'max-operation'. The problem in max-operation in multi-robot setting is that the number of computation exponentially increases as the number of robots and tasks increases. In this section, we show that *transferability of Q-function inference enables designing an efficient heuristic auction* that replaces max operation. We call it auction-based policy(ADP) and denote it as $\phi_{Q_\theta}$, where $Q_\theta$ indicates that we compute $\phi_{Q_\theta}$ using current $Q_\theta$ estimator.

At time-step $t$, a state $s_t$ is a graph $\mathcal{G}_t = (\mathcal{R}_t \cup \mathcal{T}_t, \mathcal{E}_t)$ as defined in section 2.1. Our ADP, $\phi_{Q_\theta}$, finds an action $a_t$ (which is a matching in bipartite graph $((\mathcal{R}_t, \mathcal{T}_t), \mathcal{E}_t^{RT})$ of graph $\mathcal{G}_t$) through iterations between two phases: the bidding phase and the consensus phase. We start with a bidding phase. All robots initially know the matching determined in previous iterations. We denote this matching as $\mathcal{Y}$, a bipartite subgraph of $((\mathcal{R}_t, \mathcal{T}_t), \mathcal{E}_t^{RT})$. When making a bid, a robot $r_i$ ignores all other unassigned robots. For example, suppose robot $r_i$ considers $t_j$ for bidding. For $r_i$, $\mathcal{Y} \cup \epsilon_{ij}$ is a proper action (according to definition in section 2.1) in a 'unassigned robot-ignored' problem. Robot $r_i$ thus can compute $Q(s_t, \mathcal{Y} \cup \epsilon_{r_i t_j})$ of 'unassigned robot-ignored' problem for all unassigned task $t_j$. If task $t^*$ is with the highest value, robot $r_i$ bids $\{\epsilon_{r_i t^*}, Q(s_t, \mathcal{Y} \cup \epsilon_{r_i t^*})\}$ to auctioneer. Since number of robots ignored by $r_i$ is different at each iteration, transferability of Q-function inference plays key role. The consensus phase is simple. The auctioneer finds the bid with the best value, say $\{\epsilon^*$, bid value with $\epsilon^*\}$. Then auctioneer updates everyone's $\mathcal{Y}$ as $\mathcal{Y} \cup \{\epsilon^*\}$.

These bidding and consensus phases are iterated until we can't add an edge to $\mathcal{Y}$ anymore. Then the central decision maker chooses $\mathcal{Y}$ as $\phi_{Q_\theta}(s_k)$. One can easily verify that the computational complexity of computing $\phi_{Q_\theta}$ is O $(|L_R|\,|L_T|)$, which is only polynomial. While theoretical performance guarantee of this heuristic auction is out of this paper's scope, in section 5 we show that empirically this heuristic achieves near-optimal performance.

## 4 Learning algorithm

### 4.1 Auction-fitted Q-iteration framework

In fitted Q-iteration, we fit $\theta$ of $Q_\theta(s_t, a_t)$ with stored data using Bellman optimality equation. That is, chooses $\theta$ that makes $E\left[Q_\theta\left(s_k, a_k\right) - \left[r\left(s_k, a_k\right) + \gamma \max_{a'}\left(Q_\theta\left(s'_{k+1}, a'_{k+1}\right)\right)\right]\right.$ small. Note that every update of $\theta$ needs at least one max-operation.

To solve this issue, we suggest a learning framework we call auction-fitted $Q$-iteration. What we do is simple: when we update $\theta$, we use auction-based policy(ADP) defined in section 3.3 instead of max-operation. That is, we seek the parameter $\theta$ that minimizes $E\left[Q_\theta\left(s_k, a_k\right) - \left[r\left(s_k, a_k\right) + \gamma\left(Q_\theta\left(s'_{k+1}, \phi_{Q_\theta}\left(s'_{k+1}\right)\right)\right)\right]\right.$.

### 4.2 Exploration for Auction-fitted Q-iteration

How can we conduct exploration in Auction-fitted Q-iteration framework? Unfortunately, we can't use $\epsilon$-greedy method since such randomly altered assignment is very likely to cause a catastrophic result in problems with combinatorial nature.

In this paper, we suggest that parameter space exploration [Plappert et al. (2017)] can be applied. Recall that we use $Q_\theta\left(s_k, a_k\right)$ to get policy $\phi_{Q_\theta}\left(s_k\right)$. Note that $\theta$ denotes all neural network parameters used in the structure2vec iterations introduced in Section 5. Since $Q_\theta\left(s_k, a_k\right)$ is parametrized by $\theta$, exploration with $\phi_{Q_\theta}\left(s_k\right)$ can be performed by exploration with parameter $\theta$. Such exploration in parameter space has been introduced in the policy gradient RL literature. While this method was originally developed for policy gradient based methods, exploration in parameter space can be particularly useful in auction-fitted Q-iteration.

The detailed application is as follows. When conducting exploration, apply a random perturbation on the neural network parameters $\theta$ in structure2vec. The resulting a perturbation in the Q-function used for decision making via the auction-based policy $\phi_{Q_\theta}\left(s_k\right)$ throughout that problem. Similarly, when conducting exploitation, the current surrogate Q-function is used throughout the problem.

Table 1: Performance test (50 trials of training for each cases)

| **Reward** | **Environment** | **Baseline** | **Robots (R) / Tasks (T)** | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 2R/20T | 3R/20T | 3R/30T | 5R/30T | 5R/40T | 8R/40T | 8R/50T |
| Linear | Deterministic | Optimal | 98.31% ±4.23 | 97.50% ±4.71 | 97.80% ±5.14 | 95.35% ±5.28 | 96.99% ±5.42 | 96.11% ±4.56 | 96.85% ±3.40 |
| | | (%SGA) | (137.3) | (120.6) | (129.7) | (110.4) | (123.0) | (119.9) | (119.8) |
| | Stochastic | Optimal | N/A | | | | | | |
| | | (%SGA) | (130.9) | (115.7) | (122.8) | (115.6) | (122.3) | (113.3) | (115.9) |
| Nonlinear | Deterministic | Optimal | N/A | | | | | | |
| | | (%SGA) | (111.5) | (118.1) | (118.0) | (110.9) | (118.7) | (111.2) | (112.6) |
| | Stochastic | Optimal | N/A | | | | | | |
| | | (%SGA) | (110.8) | (117.4) | (119.7) | (111.9) | (120.0) | (110.4) | (112.4) |

Updates for the surrogate Q-function may only occur after each problem is complete (and typically after a group of problems).

## 5 Experiment

### 5.1 MRRC

For MRRC, we conduct a simulation experiment for a discrete time, discrete state environment. We use maze (see Figure **??**) generator of UC Berkeley CS188 Pacman project [Neller et al. (2010)] to generate large size mazes. We generated a new maze for every training and testing experiments.

Under the deterministic environment, the robot succeeds its movement 100%. Under stochastic environment, a robot succeeds its intended movement in 55% on the grid with dots and for every other direction 15% each; on the grid without dots, the rates are 70% and 10%. As described in section 2, routing problems are already solved. That is, each robot knows how to optimally (in expectation) reach a task. To find an optimal routing policy, we use Dijkstra's algorithm for deterministic environments and dynamic programming for stochastic environments. The central assignment decision maker has enough samples of task completion time for every possible route.

We consider two reward rules: Linearly decaying rewards obey $f(age) = 200 - age$ until reaching 0, where $age$ is the task age when served; For nonlinearly decaying rewards, $f(t) = \lambda^t$ for $\lambda = 0.99$. Initial age of tasks were uniformly distributed in the interval $[0, 100]$.

**Performance test.** We tested the performance under four environments: deterministic/linear rewards, deterministic/nonlinear rewards, stochastic/ linear rewards, stochastic/nonlinear rewards.

There are two baselines used for performance test: exact baseline and indirect baseline. For the experiment with deterministic with linearly decaying rewards, an exact optimal solution for mixed-integer linear program exists and can be used as a baseline. We solve this program using Gurobi with 60-min cut to get the baseline. For any other experiments with nonlinearly decaying rewards or stochastic environment, such an exact optimal solution or other heuristics methods does not exist. In these cases, we should be conservative when talking about performance. Our strategy is to construct a indirect baseline using a universally applicable algorithm called Sequential greedy algorithm (SGA) [Han-Lim Choi et al. (2009)]. SGA is a polynomial-time task allocation algorithm that shows decent scalable performance to both linear and non-linear rewards. For stochastic environments, we use mean task completion time for task allocation and re-allocate the whole tasks at every time-steps. We construct our indirect baseline as *'ratio between our method and SGA for experiments with deterministic-linearly decaying rewards'*. Showing that *this ratio is maintained* for stochastic environments in both linear/nonlinear rewards suffices our purpose.

Table 1 shows experiment results for (# of robots, # of tasks) $= (2, 20)$, $(3, 20)$, $(3, 30)$, $(5, 30)$, $(5, 40)$, $(8, 40)$, $(8, 50)$; For linear/deterministic rewards, our proposed method achieves near-optimality (all above 95% optimality). While there is no exact or comparable performance baseline for experiments under other environments, indirect baseline (%SGA) at least shows that our method does not lose %SGA for stochastic environments compared with %SGA for deterministic environments in both linear and nonlinear rewards.

Table 2: Scalability test (mean of 20 trials of training, linear & deterministic env.)

| **Linear & Deterministic** | **Testing size (Robot (R) / Task (T))** | | | | | | |
|---|---|---|---|---|---|---|---|
| | 2R/20T | 3R/20T | 3R/30T | 5R/30T | 5R/40T | 8R/40T | 8R/50T |
| Performance with full training | 98.31% | 97.50% | 97.80% | 95.35% | 96.99% | 96.11% | 96.85% |
| # Training for 93% optimality | 19261.2 | 61034.0 | 99032.7 | 48675.3 | 48217.5 | 45360.0 | 47244.2 |

Table 3: Transferability test (50 trials of training for each cases, linear & deterministic env.)

| | **Testing size (Robot (R) / Task (T))** | | | | | | |
|---|---|---|---|---|---|---|---|
| **Training size** | 2R/20T | 3R/20T | (3R/30T) | 5R/30T | 5R/40T | 8R/40T | 8R/50T |
| Same size as testing size | 98.31% | 97.50% | (97.80%) | 95.35% | 96.99% | 96.11% | 96.85% |
| Only 3R/30T | 97.72% | 94.58% | (97.80%) | 94.68% | 97.38% | 95.05% | 96.95% |

**Scalability test.** We count the training requirements for 93% optimality for seven problem sizes (# of robots $N_R$, # of tasks $N_T$) $= (2, 20), (3, 20), (5, 30), (5, 40), (8, 40), (8, 50)$ with deterministic/linearly decaying rewards (we can compare optimality only in this case). As we can see in Table 2, the training requirement shown not to scale as problem size increases.

**Transferability test**. Suppose that we trained our learning algorithm with problems of three robots and 30 Tasks. We can claim transferability of our algorithm if our algorithm achieves similar performance for testing with problems of 8 robots and 50 tasks when compared with the algorithm specifically trained with problems of 8 robots and 50 tasks, the same size as testing. Table 3 shows the experiment result trained with three robots and 30 tasks and tested with six combinations (# of robots $N_R$, # of tasks $N_T$) $= (2, 20), (3, 20), (5, 30), (5, 40), (8, 40), (8, 50)$ with deterministic/linearly decaying rewards (we can compare optimality only in this case). Values in 'Same size as testing size' are from table 1 and shown here for comparison purposes. As we can see in table 3, we achieve transferability within 3% loss (in large problems nearly 0% loss or better).

**Ablation study.** There are three components in our proposed method: 1) a careful encoding of information using two-layers of structure2vec, 2) new structure2vec equation with random PGM and 3) an auction-based assignment. Each component was removed from the full method and tested to check the necessity of the component.

We test the performance in a deterministic/linearly decaying rewards (so that there is an optimal solution available for comparison). The experimental results are shown in Figure **??**. While the full method requires more training steps, only the full method achieves near-optimal performance.

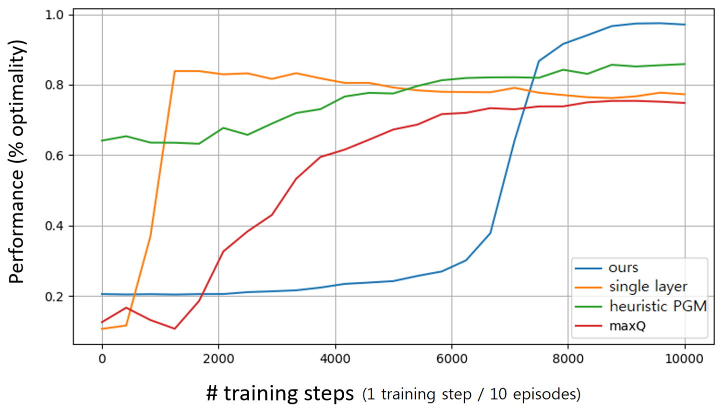

Figure 2: Tested with 1) single layer, 2) heuristic PGM 3) Max-operation

## 5.2 IPMS

For IPMS, we test it with continuous time, continuous state environment. While there have been many learning-based methods proposed for (single) robot scheduling problems, to the best our knowledge our method is the first learning method to claim scalable performance among machine-

Table 4: IPMS test results for makespan minimization (our algorithm / best Google OR tool result)

| **Makespan minimization for Deterministic environment** | | **# Machines** | | | |
|---|---|---|---|---|---|
| | | 3 | 5 | 7 | 10 |
| **# Tasks** | 50 | 106.7% | 117.0% | 119.8% | 116.7% |
| | 75 | 105.2% | 109.6% | 113.9% | 111.3% |
| | 100 | 100.7% | 111.0% | 109.1% | 109.0% |

scheduling problems. Hence, in this case, we focus on showing comparable performance for large problems, instead of attempting to show the superiority of our method compared with heuristics specifically designed for IPMS (actually no heuristic was specifically designed to solve our exact problem (makespan minimization, sequence-dependent setup with no restriction on setup times))

For each task, processing times is determined using uniform [16, 64]. For every (task $i$, task $j$) ordered pair, a unique setup time is determined using uniform [0, 32]. As illustrated in section 2, we want to minimize make-span. As a benchmark for IPMS, we use Google OR-Tools library Google (2012). This library provides metaheuristics such as Greedy Descent, Guided Local Search, Simulated Annealing, Tabu Search. We compare our algorithm's result with the heuristic with the best result for each experiment. We consider cases with $3, 5, 7, 10$ machines and $50, 75, 100$ jobs.

The results are provided in Table 4. Makespan obtained by our method divided by the makespan obtained in the baseline is provided. Although our method has limitations in problems with a small number of tasks, it shows comparable performance to a large number of tasks and shows its value as the first learning-based machine scheduling method that achieves scalable performance.

## 6 Conclusions

We presented a learning-based method that achieves the first success for multi-robot/machine scheduling problems in both challenges: scalable performance and tranferability. We identified that robot scheduling problems have an exact representation as random PGM. We developed a mean-field inference theory for random PGM and extended structure2vec method of Dai et al. (2016). To overcome the limitations of fitted Q-iteration, a heuristic auction that was enabled by transferability is suggested. Through experimental evaluation, we demonstrate our method's success for MRRC problems under deterministic/stochastic environment. Our method also claims to be the first learning-based algorithm that achieves scalable performance among machine scheduling algorithms; our method achieves a comparable performance in a scalable manner.

## References

Ali-akbar Agha-mohammadi, Suman Chakravorty, and Nancy M Amato. FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304, 2014. ISSN 0278-3649. doi: 10.1177/0278364913501564. URL `http://journals.sagepub.com/doi/10.1177/0278364913501564`.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. *Annual Review of Cell and Developmental Biology*, 15(1):81–112, nov 2016. ISSN 1081-0706. doi: 10.1146/annurev.cellbio.15.1.81.

Hanjun Dai, Bo Dai, and Le Song. Discriminative Embeddings of Latent Variable Models for Structured Data. 48:1–23, 2016. doi: 1603.05629.

Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning Combinatorial Optimization Algorithms over Graphs. (Nips), 2017.

Google. Google OR-Tools, 2012. URL `https://developers.google.com/optimization/`.

Han-Lim Choi, Luc Brunet, and J.P. How. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Transactions on Robotics*, 25(4):912–926, aug 2009. ISSN 1552-3098. doi: 10.1109/TRO.2009.2022423.

Daphne Koller and Nir Friedman. *Probabilistic graphical models : principles and techniques, page 449-453*. The MIT Press, 1st edition, 2009. ISBN 9780262013192.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! mar 2018.

M E Kurz, R G Askin, M E Kurzy, and R G Askiny. Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39(16): 3747–3769, 2001. ISSN 0020-7543. doi: 10.1080/00207540110064938.

Yuxi Li. Deep Reinforcement Learning: An Overview. jan 2017.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takáč. Reinforcement Learning for Solving the Vehicle Routing Problem. feb 2018.

Todd Neller, John DeNero, Dan Klein, Sven Koenig, William Yeoh, Xiaoming Zheng, Kenny Daniel, Alex Nash, Zachary Dodds, Giuseppe Carenini, David Poole, and Chris Brooks. Model AI Assignments. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pp. 1919–1921, 2010.

Shayegan Omidshafiei, AliAkbar AghaMohammadi, Christopher Amato, ShihYuan Liu, Jonathan P How, and John Vian. Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions. *The International Journal of Robotics Research*, 36 (2):231–258, 2017. doi: 10.1177/0278364917692864.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter Space Noise for Exploration. pp. 1–18, 2017.

Rajesh Ranganath, Dustin Tran, and David M. Blei. Hierarchical Variational Models. 2015.

Federico Rossi, Saptarshi Bandyopadhyay, Michael Wolf, and Marco Pavone. Review of Multi-Agent Algorithms for Collective Behavior: a Structural Taxonomy. *IFAC-PapersOnLine*, 51(12): 112–117, 2018. ISSN 24058963. doi: 10.1016/j.ifacol.2018.07.097.

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. 2015.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. NerveNet: Learning Structured Policy with Graph Neural Networks. In *International Conference on Learning Representations*, 2018.

## A MRRC WITH CONTINUOUS STATE/CONTINUOUS TIME SPACE FORMULATION, OR WITH SETUP TIME AND PROCESSING TIME

In continuous state/continuous time space formulation, the initial location and ending location of robots and tasks are arbitrary on $\mathbb{R}^2$. At every moment at least a robot finishes a task, we make assignment decision for a free robot(s). We call this moments as 'decision epochs' and express them as an ordered set $(t_1, t_2, \dots, t_k, \dots)$. Abusing this notation slightly, we use $(\cdot)_{t_k} = (\cdot)_k$.

Task completion time can consist of three components: travel time, setup time and processing time. While a robot in the travel phase or setup phase may be reassigned to other tasks, we can't reassign a robot in the processing phase. Under these assumptions, at each decision epoch robot $r_i$ is given a set of tasks it can assign itself: if it is in the traveling phase or setup phase, it can be assigned to any tasks or not assigned; if it is in the processing phase, it must be reassigned to its unfinished task. This problem can be cast as a Markov Decision Problem (MDP) whose state, action, and reward are defined as follows:

**State.** State $s_k$ at decision epoch $k$ is a directed graph $\mathcal{G}_k = (\mathcal{R}_k \cup \mathcal{T}_k, \mathcal{E}_k)$: $\mathcal{R}_k$ is the set of all robots and $\mathcal{T}_k$ is the set of all tasks; The set of directed edges $\mathcal{E}_k = \mathcal{E}_k^{RT} \cup \mathcal{E}_k^{TT}$ where a directed edge $\epsilon_{r_i t_j} \in \mathcal{E}_k^{RT}$ is a random variable which denotes task completion time of robot $i$ in $\mathcal{R}_k$ to service task $j$ in $\mathcal{T}_k$ and a directed edge $\epsilon_{t_i t_j} \in \mathcal{E}_k^{TT}$ denotes a task completion time of a robot which just finished serving task $i$ in $\mathcal{T}_k$ to service task $j$ in $\mathcal{T}_k$. $\mathcal{E}_k^{RT}$ contains information about each robot's possible assignments: $\mathcal{E}_k^{RT} = \cup_i \mathcal{E}_k^{r_i}$, where $\mathcal{E}_t^{r_i}$ is a singleton set if robot $i$ is in the processing phase and it must be assigned to its unfinished task, and otherwise it is the set of possible assignments from robot $r_i$ to remaining tasks that are not in the processing phase.

**Action.** The action $a_k$ at decision epoch $k$ is the joint assignment of robots given the current state $s_k = \mathcal{G}_k$. The feasible action should satisfy the two constraints: No two robots can be assigned to a task; some robots may not be assigned when number of robots are more than remaining tasks. To best address those restrictions, we define an action $a_k$ at time $t$ as a maximal bipartite matching in bipartite sub-graph $((\mathcal{R}_k, \mathcal{T}_k), \mathcal{E}_k^{RT})$ of graph $\mathcal{G}_k$. For example, robot $i$ in $\mathcal{R}_k$ is matched with task $j$ in $\mathcal{T}_k$ in an action $a_k$ if we assign robot $i$ to task $j$ at decision epoch $t$. We denote the set of all possible actions at epoch $k$ as $\mathcal{A}_k$.

**Reward.** In MRRC, Each task has an arbitrarily determined initial age. At each decision epoch, the age of each task increases by one. When a task is serviced, a reward is determined only by its age when serviced. Denote this reward rule as $R(k)$. One can easily see that whether a task is served at epoch $k$ is completely determined by $s_k$, $a_k$ and $s_{k+1}$. Therefore, we can denote the reward we get with $s_k, a_k$ and $s_{k+1}$ as $R(s_k, a_k, s_{k+1})$.

**Objective.** We can now define an assignment policy $\phi$ as a function that maps a state $s_k$ to action $a_k$. Given $s_0$ initial state, an MRRC problem can be expressed as a problem of finding an optimal assignment policy $\phi^*$ such that

$$\phi^* = \underset{\phi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{k=0}^{\infty} R\left(s_k, a_k, s_{k+1}\right) | s_0\right].$$

## B Identical parallel machine scheduling problem formulation

As written in 2.2, IPMS is a problem defined in continuous state/continuous time space. Machines are all identical, but processing times of tasks are all different. In this paper, we discuss IPMS with 'sequence-dependent setup time'. A machine's setup time required for servicing a task $i$ is determined by its previously served task $j$. In this case, the task completion time is the sum of setup time and processing time. Under this setting, we solve IPMS problem for make-span minimization objective discussed in [Kurz et al. (2001)] (The constraints are different in this problem though); That is, minimizing total time spent from start to end to finish all tasks.

Every time there is a finished task, we make assignment decision for a free machine. We call this times as 'decision epochs' and express them as an ordered set $(t_1, t_2, \ldots, t_k, \ldots)$. Abusing this notation slightly, we use $(\cdot)_{t_k} = (\cdot)_k$.

Task completion time for a machine to a task consists of two components: processing time and setup time. While a machine in setup phase may be reassigned to another task, we can't reassign a machine in the processing phase. Under these assumptions, at each epoch, a machine $r_i$ is given a set of tasks it can assign: if it is in the setup phase, it can be assigned to any tasks or not assigned; if it is in the processing phase, it must be reassigned to its unfinished task. This problem can be cast as a Markov Decision Problem (MDP) whose state, action, and reward are defined as follows:

**State.** State $s_k$ at decision epoch $k$ is a directed graph $\mathcal{G}_k = (\mathcal{R}_k \cup \mathcal{T}_k, \mathcal{E}_k)$: $\mathcal{R}_k$ is the set of all machines and $\mathcal{T}_k$ is the set of all tasks; The set of directed edges $\mathcal{E}_k = \mathcal{E}_k^{RT} \cup \mathcal{E}_k^{TT}$ where a directed edge $\epsilon_{r_i t_j} \in \mathcal{E}_k^{RT}$ is a random variable which denotes task completion time of machine $i$ in $\mathcal{R}_k$ to service task $j$ in $\mathcal{T}_k$ and a directed edge $\epsilon_{t_i t_j} \in \mathcal{E}_k^{TT}$ denotes a task completion time of a machine which just finished serving task $i$ in $\mathcal{T}_k$ to service task $j$ in $\mathcal{T}_k$. $\mathcal{E}_k^{RT}$ contains information about each robot's possible assignments: $\mathcal{E}_k^{RT} = \cup_i \mathcal{E}_k^{r_i}$, where $\mathcal{E}_k^{r_i}$ is a singleton set if machine $i$ is in the processing phase and it must be assigned to its unfinished task, and otherwise it is the set of possible assignments from machine $r_i$ to remaining tasks that are not in the processing phase.

**Action.** Defined the same as MRRC with continuous state/time space.

**Reward.** In IPMS, time passes between decision epoch $t$ and decision epoch $t+1$. Denote this time as $T_t$. One can easily see that $T_t$ is completely determined by $s_k$, $a_k$ and $s_{k+1}$. Therefore, we can denote the reward we get with $s_k, a_k$ and $s_{k+1}$ as $T(s_k, a_k, s_{k+1})$.

**Objective.** We can now define an assignment policy $\phi$ as a function that maps a state $s_k$ to action $a_k$. Given $s_0$ initial state, an MRRC problem can be expressed as a problem of finding an optimal assignment policy $\phi^*$ such that

$$\phi^* = \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}\left[\sum_{k=0}^{\infty} T\left(s_k, a_k, s_{k+1}\right) |s_0\right].$$

## C Bayesian Network representation

Here we analytically show that robot scheduling problem randomly induces a random Bayesian Network from state $s_t$. To do this, we first define a concept 'scenario at $s_t, a_t$ with policy $\phi$'. Given starting state $s_t$ and action $a_t$, a person can may repeat a random experiment of sequential decision making from $s_t, a_t$ using $\phi$. In this random experiment, we can define events 'How robots serve all remaining tasks in which sequence'. We call such an event a 'scenario'. For example, suppose that at time-step $t$ we are given robots $\{A, B\}$, tasks $\{1, 2, 3, 4, 5\}$, and policy $\phi$. One possible scenario $S^*$ can be {robot A serves task $3 \to 1 \to 2$ and robot B serves task $5 \to 4$}. Define random variable $X_k$ a task characteristic, e.g. 'The time when task $k$ is serviced'. The question is, 'Given a scenario $S^*$, what is the relationship among random variables $\{X_k\}$'? Recall that in our sequential decision making formulation we are given all the 'task completion time' information in the $s_t$ description. Note that, task completion time is only dependent on the previous task and assigned task. In our example above, under scenario $S^*$ 'when task 2 is served' is only dependent on 'when task 1 is served'. That is, $P(X_2|X_1, X_3, S^*) = P(X_2|X_1, S^*)$. This relationship is called 'conditional independence'. Given a scenario $S^*$, every relationship among $\{X_i|S^*\}$ can be expressed using this kind of relationship among random variables. A graph with this special relationship is called 'Bayesian Network' [Koller & Friedman (2009)], a probabilistic graphical model.

## D Proof of Theorem 1.

We start from formally stating what a probabilistic graphical model is. Given a set of random variables $\{X_k\}$ and set of semi-cliques $\mathfrak{C}$, suppose that we can express the joint distribution of $\{X_k\}$ as

$$P\left(X_1, \ldots, X_n\right) = \frac{1}{Z}\prod_i \phi^i\left(\mathcal{D}^i\right), \tag{1}$$

where $\phi^i$ clique potential for clique $\mathcal{D}^i \in \mathfrak{C}$ and $Z$ is a normalizing constant. Sometimes, we will suppress the explicit dependence on $\mathcal{D}^i$ and write simply $\phi^i$. These functions $\phi^i$ are often referred to as clique potentials. The collection $\mathcal{D}^i$ is referred to as the scope of clique potential $\phi^i$. The set of cliques in a probabilistic graphical model is called a factorization and expressed as notation $\mathcal{S} \subseteq \mathfrak{C}$.

In a random graphical model, a PGM is chosen among PGMs with $\{X_k\}$ and semi-cliques $\mathfrak{C}$. Denote the set of all possible factorization as $\mathcal{F} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_N\}$ where a factorization with index $k$ is denoted as $\mathcal{S}_k \subseteq \mathfrak{C}$. Suppose we are given $P(\mathcal{S}_m)$.

For each semi-clique $\mathcal{D}^i$ in $\mathfrak{C}$, define a binary random variable $V^i$: $\mathcal{F} \mapsto \{0, 1\}$ with value 0 for the factorization that does not include semi-clique $\mathcal{D}^i$ and value 1 for the factorization that include semi-clique $\mathcal{D}^i$. Let $V$ be a random vector $V = \left(V^1, V^2, \ldots, V^{|\mathfrak{C}|}\right)$. Then we can express $P(X_1, \ldots, X_n|V) \propto \prod_{i=1}^{|\mathfrak{C}|}\left[\phi^i\left(\mathcal{D}^i\right)\right]^{V^i}$. In other words, $\{\left[\phi^i\left(\mathcal{D}^i\right)\right]^{V^i}\}_{i=1}^{|\mathfrak{C}|}$ factors $P(X_1, \ldots, X_n|V)$. We denote $\left[\phi^i\left(\mathcal{D}^i\right)\right]^{V^i}$ as $\psi(\mathcal{D}^i)$.

Given $\{P(S_k)\}$, each semi-clique $\mathcal{D}^i$'s presence probability $p_i$ can be simply calculated; clique $i$'s presence probability $p_i$ is simply the sum of probabilities of all factorizations which include clique $i$, that is, $p_i = \sum_{m:\mathcal{D}^i\in\mathcal{S}_m} P\left(\mathcal{S}_m\right)$.

Now we prove Theorem 1.

In mean-field inference, we want to find a distribution $Q\left(X_1, \ldots, X_n\right) = \prod_{i=1}^n Q_i(X_i)$ such that the cross-entropy between it and a target distribution is minimized. Following the notation in Koller

& Friedman (2009), the mean field inference problem can written as the following optimization problem.

$$\begin{aligned}\min_{Q} \quad & \mathbb{D}\left(\prod_i Q_i \,|P\left(X_1,\ldots,X_n|V\right)\right)\\ \text{s.t.} \quad & \sum_{x_i} Q_i\left(x_i\right)=1 \quad \forall i\end{aligned}$$

Here $\mathbb{D}\left(\prod_i Q_i \mid P\left(X_1,\ldots,X_n|V\right)\right)$ can be expressed as $\mathbb{D}\left(\prod_i Q_i \mid P\left(X_1,\ldots,X_n|V\right)\right) = \mathbb{E}_Q\left[\ln\left(\prod_i Q_i\right)\right] - \mathbb{E}_Q\left[\ln\left(P\left(X_1,\ldots,X_n|V\right)\right)\right]$.
Note that

$$\begin{aligned}
\mathbb{E}_Q\left[\ln\left(P\left(X_1,\ldots,X_n|V\right)\right)\right] &= \mathbb{E}_Q\left[\ln\left(\frac{1}{z}\Pi_{i=1}^{|\mathfrak{C}|}\psi^i\left(\mathcal{D}^i,V\right)\right)\right]\\
&= \mathbb{E}_Q\left[\ln\left(\frac{1}{z}\prod_{i=1}^{|\mathfrak{C}|}\psi^i\left(\mathcal{D}^i,V\right)\right)\right]\\
&= \mathbb{E}_Q\left[\sum_{i=1}^{|\mathfrak{C}|}V^i\ln\left(\phi^i\left(\mathcal{D}^i\right)\right)\right]-\mathbb{E}_Q[\ln(Z)]\\
&= \sum_{i=1}^{|\mathfrak{C}|}\mathbb{E}_Q\left[V^i\ln\left(\phi^i\left(\mathcal{D}^i\right)\right)\right]-\mathbb{E}_Q[\ln(Z)]\\
&= \sum_{i=1}^{|\mathfrak{C}|}\mathbb{E}_{V^i}\left[\mathbb{E}_Q\left[V^i\ln\left(\phi^i\left(\mathcal{D}^i\right)\right)|V^i\right]\right]-\mathbb{E}_Q[\ln(Z)]\\
&= \sum_{i=1}^{|\mathfrak{C}|}P\left(V^i=1\right)\left[\mathbb{E}_Q\left[\ln\left(\phi^i\left(\mathcal{D}^i\right)\right)\right]\right]-\mathbb{E}_Q[\ln(Z)]\\
&= \sum_{i=1}^{|\mathfrak{C}|}p_i\left[\mathbb{E}_Q\left[\ln\left(\phi^i\left(\mathcal{D}^i\right)\right)\right]\right]-\mathbb{E}_Q[\ln(Z)].
\end{aligned}$$

Hence, the above optimization problem can be written as

$$\begin{aligned}\max_{Q} \quad & \mathbb{E}_Q\left[\sum_{i=1}^{|\mathfrak{C}|}p_i\ln\left(\phi^i\left(\mathcal{D}^i\right)\right)\right]+\mathbb{E}_Q\sum_{i=1}^{n}\left(\ln Q_i\right)\\ \text{s.t.} \quad & \sum_{x_i}Q_i\left(x_i\right)=1 \quad \forall i\end{aligned} \tag{2}$$

In Koller & Friedman (2009), the fixed point equation is derived by solving an analogous equation to (2) without the presence of the $p_i$. Theorem 1 follows by proceeding as in Koller & Friedman (2009) with straightforward accounting for $p_i$.

## E PROOF OF LEMMA 1.

Since we assume semi-cliques are only between two random variables, we can denote $\mathfrak{C} = \mathcal{D}^{ij}$ and presence probabilities as $\{p_{ij}\}$ where $i, j$ are node indexes. Denote the set of nodes as $\mathcal{V}$.

From here, we follow the approach of Dai et al. (2016) and assume that the joint distribution of random variables can be written as

$$p\left(\{H_k\},\{X_k\}\right) \propto \prod_{k\in\mathcal{V}} \psi^i\left(H_k|X_k\right) \prod_{k,i\in\mathcal{V}} \psi^i\left(H_k|H_i\right).$$

Expanding the fixed-point equation for the mean field inference from Theorem 1, we obtain:

$$\begin{aligned} Q_k\left(h_k\right) &= \frac{1}{Z_k}\exp\left\{\sum_{\psi^i:H_k\in\mathcal{D}^i} \mathbb{E}_{(\mathcal{D}^i-\{H_k\})\sim Q}\left[\ln\psi^i\left(H_k=h_k|\mathcal{D}^i\right)\right]\right\} \\ &= \frac{1}{Z_k}\exp\left\{\ln\phi\left(H_k=h_k|x_k\right)+\sum_{i\in\mathcal{V}}\int_{\mathcal{H}} p_{ki}Q_i\left(h_i\right)\ln\phi\left(H_k=h_k|H_i\right)dh_i\right\}. \end{aligned}$$

This fixed-point equation for $Q_k\left(h_k\right)$ is a function of $\left\{Q_j\left(h_j\right)\right\}_{j\neq k}$ such that

$$Q_k\left(h_k\right) = f\left(h_k, x_k, \left\{p_{kj}Q_j\left(h_j\right)\right\}_{j\neq k}\right).$$

As in Dai et al. (2016), this equation can be expressed as a Hilbert space embedding of the form

$$\tilde{\mu}_k = \tilde{\mathcal{T}}\circ\left(x_k, \{p_{kj}\tilde{\mu}_j\}_{j\neq i}\right),$$

where $\tilde{\mu}_k$ indicates a vector that encodes $Q_k\left(h_k\right)$. In this paper, we use the nonlinear mapping $\tilde{\mathcal{T}}$ (based on a neural network form ) suggested in Dai et al. (2016):

$$\tilde{\mu_k} = \sigma\left(W_1 x_k + W_2\sum_{j\neq k} p_{kj}\tilde{\mu}_j\right)$$

## F Presence probability inference

Let $\mathcal{V}$ denote the set of nodes. In lines 1 and 2, the likelihood of the existence of a directed edge from each node $m$ to node $n$ is computed by calculating $W_1\left(relu\left(W_2 u^k_{mn}\right)\right)$ and averaging over the $M$ samples. In lines 3 and 4, we use the soft-max function to obtain $p_{m,n}$.

```
1  For $m, n \in \mathcal{V}$ do
2      $g_{mn} = \frac{1}{M}\sum_{k=1}^{M} W_1\left(relu\left(W_2 u^k_{mn}\right)\right)$
3  For $m, n \in \mathcal{V}$ do
4      $p_{m,n} = \frac{e^{g_{mn}/\tau}}{\sum_{j\in v} e^{g_{mn}/\tau}}$.
```

## G Task completion time as a random variable

We combine random sampling and inference procedure suggested in section 3.2 and Figure **??**. Denote the set of task with a robot assigned to it as $\mathcal{T}^A$. Denote a task in $\mathcal{T}^A$ as $t_i$ and the robot assigned to $t_i$ as $r_{t_i}$. The corresponding edge in $\mathcal{E}^{RT}$ for this assignment is $\epsilon_{r_{t_i}t_i}$. The key idea is to use samples of $\epsilon_{r_{t_i}t_i}$ to generate $N$ number of sampled $Q(s,a)$ value and average them to get the estimate of $E(Q(s,a))$. First, for $l = 1\ldots N$ we conduct the following procedure. For each task $t_i$ in $\mathcal{T}^A$, we sample one data $e^l_{r_{t_i}t_i}$. Using those samples and $\{p_{ij}\}$, we follow the whole procedure illustrated in section 3.2 to get $Q(s,a)^l$. Second, we get the average of $\{Q(s,a)^l\}_{l=1}^{l=N}$ to get the estimate of $E(Q(s,a))$, $\frac{1}{N}\sum_{l=1}^{l=N} Q(s,a)^l$.

The complete algorithm of section 3.2 with task completion time as a random variable is given as below.

```
1  $age_i$ = age of node $i$
2  The set of nodes for assigned tasks $\equiv \mathcal{T}_A$
3  Initialize $\{\mu_i^{(0)}\}, \{\gamma_i^{(0)}\}$
```

```
4   for $l = 1$ to $N$:
5     for $t_i \in \mathcal{T}$:
5       if $t_i \in \mathcal{T}^{\mathcal{A}}$ do:
6         sample $e^l_{r_{t_i} t_i}$ from $\epsilon_{r_{t_i} t_i}$
7         $x_i = e^l_{r_{t_i} t_i}$
9       else: $x_i = 0$
10    for $t = 1$ to $T_1$ do
11      for $i \in \mathcal{V}$ do
12        $l_i = \sum_{j \in \mathcal{V}} p_{ji} \mu_j^{(t-1)}$
13        $\mu_i^{(t)} = relu\left(W_3 l_i + W_4 x_i\right)$
14    $\widetilde{\mu}_l =$ Concatenate $\left(\mu_i^{(T_1)}, age_i\right)$
15    for $t = 1$ to $T_2$ do
16      for $i \in \mathcal{V}$ do
17        $l_i = \sum_{j \in \mathcal{V}} p_{ji} \gamma_j^{(t-1)}$
18        $\gamma_j^{(t)} = relu\left(W_5 l_i + W_6 \tilde{\mu}_i\right)$
19    $Q_l = W_7 \sum_{i \in \mathcal{V}} \gamma_i^{(T)}$
20  $Q_{avg} = \frac{1}{N} \sum_{l=1}^{N} Q_l$
```