GiPH: Generalizable Placement Learning for Adaptive Heterogeneous Computing

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose **GiPH**, a reinforcement learning (RL) approach to learning efficient and fully generalizable placement search policies for heterogeneous computing. The placement of a computational graph on a target device cluster is critical for achieving low latency for distributed computing (e.g., cloud computing, distributed neural network training). The problem is challenging due to its NP-hardness and combinatorial nature. In recent years, learning-based approaches have been proposed to learn a general placement policy that can be applied to unseen graphs. While in practice, a wide range of application graphs need to run constantly changing networks of devices, the learned policies based on existing formulations cannot quickly adapt to changes in the device cluster because those methods only take a fixed number of devices into account. GiPH overcomes this limitation through the use of 1) a novel graph representation **gpNet** that efficiently encodes the information needed for choosing a good placement, and 2) a scalable graph neural network (GNN) that directly takes edge features with topological information into account. GiPH turns the placement problem into a problem of finding a sequence of placement improvements to learn a policy that scales to placement problems of arbitrary sizes. The learned policy can efficiently search for good placement given a heterogeneous task graph and a cluster of heterogeneous devices. We evaluate GiPH with a wide range of randomly generated task graphs and device clusters and show that our learned policy can be applied to new problem instances to rapidly find good placements. Our learned policy achieves placement speedup by up to 30.5%, and searches up to $3\times$ faster than other search-based placement policies. GiPH policy also adapts to network changes and produces better placement results than baselines.

1 Introduction

The placement of computation tasks of distributed applications has key importance on performance of distributed platforms. When running a compute application across a network of computing devices, careful choice of which parts of the application to run on which device can significantly affect application performance. This is particularly true when devices are heterogeneous: e.g., compute-intensive tasks should be run on devices with more computation resources, but those devices may not be the best choice if they have limited communication resources. Solving this problem is particularly difficult. Such placement problems on heterogeneous networks are particularly common in the emerging Internet-of-Things, where we have a mix of edge devices, edge servers, and clond servers. The resulting device placement problem is typically modeled by representing applications as

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

directed acyclic graphs (DAGs), in which each node represents a unit of computation (task) and edges represent data links between tasks which determine the sequence of computations. By using DAGs to encode the precedence order of different computation units within an application, we can measure the end-to-end completion time of the application as a function of the computing and communication resources available at each device, given an application placement.

Due to the NP-hardness and combinatorial nature of the placement problem, heuristic methods that rely on simple strategies and hand-crafted features have been proposed. However, they can only achieve sub-optimality and forego potential performance optimization. Many of the heuristics also assume an inaccurate or simplified performance model (e.g., computation and communication times) about the system to enable a closed-form formulation. This paper, instead, follows another recent line of work that has focused on machine-learning techniques to automatically learn highly efficient placement policies using reinforcement learning (RL). Unlike traditional static heuristic methods, RL allows learning from simulated or real completion times without heavily relying on inaccurate assumptions.

RL also has the potential to generalize across different problem instances, which cannot be done by static heuristics. The training overhead is reduced when the policy learned from previous experience is applicable to a wide range of cases. Although RL learns problem-specific policies, in practice a wide range of applications may run on constantly changing networks of devices. It is thus imperative to design RL representations and learning algorithms that can generalize well. Existing approaches fail to be fully generalizable as they either do not take features of the network of devices into account or only consider a fixed number of devices. As a result, whenever the device network changes, the learned policies will perform poorly and require significant amount of re-training. In heterogeneous computing environment, placement feasibility constraints may also exist due to hardware availability, which in general are not well handled by RL.

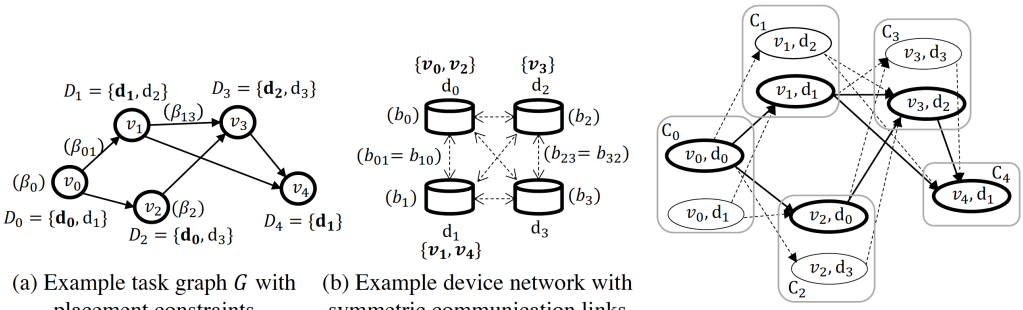
In this work, we propose **GiPH**, an RL-based approach to learning efficient and fully Generalizable Placement with the ability to adapt to dynamic Heterogeneous networks. To the best of author's knowledge, GiPH is the very first RL approach to learn a placement policy that not only generalizes to new task graphs that are not in the training set, but also adapts to changing device networks. The key contributions and findings of this work can be summarized as follows:

- We develop a *scalable* and *generalizable* neural network design that can process *general* placement problems given arbitrary task graphs and device networks.
- We formulate the learning problem as a *search problem* where the placement is done through applying a sequence of iterative placement improvements.
- We propose **GiPH**, a RL-based framework for learning *generalizable* placement policy. Given a heterogeneous device network and a heterogeneous task graph, the learned policy efficiently searches for good placement and produces results comparable to HEFT (Topcuoglu et al. [2002]).
- We devise **gpNet**, a novel and *universal* graph representation of the placement problem that takes both the task graph and the device network into account. It is flexible for representing the placement of any task graph on any device network with optional placement constraints.
- We evaluate GiPH in terms of the placement quality and generalizability. For completion time minimization, GiPH policy achieves placement speedup by up to 30.5% with higher search efficiency.

2 Placement Problem

This section defines a general placement problem that takes into account 1) placement constraints, 2) heterogeneous compute features, and 3) heterogeneous communication features.

Given a distributed application and a device network, a placement maps each computation task in the application to a device in the network to optimize a performance criteria. In this work, we consider



(a) Example task graph G with placement constraints

(b) Example device network with symmetric communication links

(c) gpNet representation. The pivots and the edges between pivots are identified in bold.

Figure 1: (a) and (b) show the task graph and the device network, respectively, of an example placement problem. The set of feasible devices for each task (placement constraints) is shown in (a). Node and edge features are in parenthesis. (c) shows the gpNet representation of a feasible placement indicated in (a) and (b).

the minimization of the completion time of an application, which is the time duration from the start of the first task's execution to the end of the last task's execution. An application is defined by a directed acyclic graph $G = (V, E)$, as shown in Fig. 1(a), where nodes $V = \{v_0, ..., v_{n-1}\}$ represent computation tasks of the application and edges $E \subset V \times V$ represent inter-task data dependencies and communication. Each edge also represents the precedence constraint such that a task should complete its execution before any of its child tasks starts. The raw features of each node $v_i \in V$ and each edge $(v_i, v_j) \in E$, represented by β_i^n and β_{ij}^e, respectively, should be defined according to the performance criteria to optimize, e.g., node features and edge features may include the amount of compute and data transfer each task and data link requires for completion time minimization.

A target computing network (e.g., Fig. 1(b)) consists of a set $D = \{d_0, ..., d_{m-1}\}$ of m devices. Each device d_i has device feature b_i (i.e., device compute speed, hardware type) and each pair of devices (d_i, d_j) has communication link features b_{ij} (e.g., bandwidths, delay). For the purpose of this paper, we assume the devices are fully connected and the communication links are symmetric $b_{ij} = b_{ji}$, and we only consider single-link paths (represented by edges) between devices. It is easy to generalize to more complex device topology by attaching extremely high communication losses to links that do not exist.

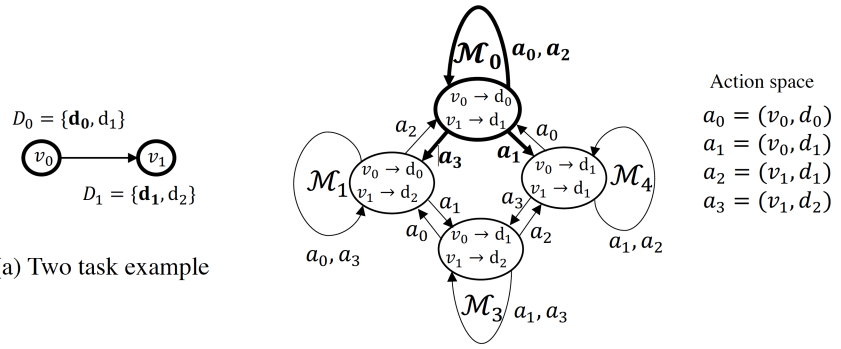
A placement of an application G on a network N is a mapping $\mathcal{M} : V \to D$. For heterogeneous network, we consider placement constraints resulted from hardware availability and feasibility, where each computation task v_i can only be mapped to a subset of devices $D_i \subseteq D$. See Fig. 1(a) and (b) for an example. The goal of the placement is to optimize a performance criteria $\rho(\mathcal{M}|G, N)$ while satisfying $\mathcal{M}(v_i) \in D_i$ for all $v_i \in V$. (G, N) defines a specific problem instance, and we denote a general placement as a triple $\mathcal{P} = (G, N, \mathcal{M}^{G \to N})$.

3 GiPH

This section introduces GiPH. Each subsection describes one of the key ideas: the formulation of the search problem and the associated Markov decision process (MDP) (§3.1), gpNet representation of the placement (§3.2), the scalable neural network design (§3.3), and RL training (§3.4).

3.1 MDP Formalism

We formulate the learning problem as a search problem, where given an initial placement, a learned policy iteratively applies some small changes to the current placement. Through making incremental changes, the policy is able to search through the solution space and find better placement solutions. Instead of trying to learn a policy that places the whole graph at once, our search approach makes the learning simpler by only considering a small local search space at a time.



(a) Two task example

(b) State action diagram of the MDP for the two-task example in (a)

Figure 2: MDP of the placement search problem.

Consider a single problem instance (G, N). For the search problem, we define the *state space* as the set of all feasible placement $S_{G,N} = \{\mathcal{M}|\mathcal{M}(v_i) \in D_i, \forall v_i \in V\}$. (*state* and *placement* terms are interchangeably used in the paper.) The size of the state space $|S_{G,N}| = \prod_{i=0}^{|V|-1} |D_i|$, since each task can be placed on any of the feasible devices for it. When there is no placement constraint, $|S_{G,N}| = |D|^{|V|}$. For the two-task example shown in Fig.2(a), there are a total of 4 feasible placements, all shown as states in the transition diagram of Fig.2(b).

To search through the state space, we define an *action* to be a task and device pair (v_i, d_j) that sets the placement of v_i to device d_j. Suppose at time t, the current state $s_t = \mathcal{M}$. If $\mathcal{M}(v_i) = d_j$, the action does not change the current placement and $s_{t+1} = s_t = \mathcal{M}$. If $\mathcal{M}(v_i) \neq d_j$, the action moves the placement of v_i from $\mathcal{M}(v_i)$ to d_j, and $s_{t+1} = \mathcal{M}' \neq \mathcal{M}$ such that $\mathcal{M}'(v_i) = d_j$ and $\mathcal{M}'(v_k) = \mathcal{M}(v_k)$ for all $k \in [n]/i$. We only consider feasible actions (v_i, d_j) such that $d_j \in D_i$. The size of the action space is thus $|A_{G,N}| = \sum_{i=0}^{|V|-1} |D_i|$. In the case there is no placement constraint, $|A_{G,N}| = |V||D|$. Fig. 2(b) lists all four actions for the simple two-task example and shows the deterministic state transition given an action taken at each state.

Note that the diameter, i.e., the longest shortest path between state of the state transition diagram is $|V|$ because you can always change from one placement to any other placement by moving each task node for at most once. Therefore, even though the state space grows exponentially with $|V|$, it is always possible to go from any state to any other state in $|V|$ steps.

We consider the MDP for the placement of a task graph G on a device network D as follows. At first, the episode is initialized at a random placement $s_0 \sim |S_{G,N}|$, where each task node is randomly assigned to one of the feasible devices. At each step t, an action $a_t \in A_{G,N}$ is taken to update the current placement. The episode length is set to $2|V|$ steps, during which the agent is free to explore the search landscape through "trial and error".

The objective function $\rho(\mathcal{M}|G, N)$ reflects how good a state $s = \mathcal{M}$ is. We assign intermediate reward $r_t = \rho(s_{t+1}|G, N) - \rho(s_t|G, N)$ that indicates the performance improvement after taking an action a_t at state s_t for $t = 0, 1, ..., 2|V| - 1$. The goal of RL is to learn to take actions in order to maximize the expected discounted return $\sum_{t=0}^{2|V|-1} \gamma^t r_t$. When $\gamma = 1$, the expected return is the expected performance improvement between the final state $s_{2|V|}$ and a randomly initialized state s_0, i.e., the policy tries to maximize $\mathbb{E}[\rho(s_{2|V|}|G, N)] - \mathbb{E}[\rho(s_0|G, N)]$. Since the latter term is constant for a (G, N) pair with random initialization, RL is effectively improving the expected performance of the final placement through maximizing the expected return. When $\gamma < 1$, the policy seeks more immediate reward as future rewards are discounted. In this case, the policy also learns to search more efficiently at the beginning of the episodes.

3.2 gpNet Representation

We have formulated a discrete MDP for the placement search problem given (G, N). However, for the learned policy to be fully generalizable without depending on (G, N), a representation of a general placement $\mathcal{P} = (G, N, \mathcal{M}^{G\to N})$ is required to work across different problem instances with an arbitrary pair of task graph and device network. The representation should capture the heterogeneous compute and communication requirements of the task graph G and the heterogeneous compute and communication capabilities of the device network N, in order to enable the learning of the relation between ρ and $\mathcal{P}$.

To this end, we present gpNet, a novel and universal graph representation of the placement that encapsulates features of both the task graphs and the device networks with placement constraints. gpNet generates a unique graph $H = (V_H, E_H)$ given a general placement $\mathcal{P} = (G, N, \mathcal{M}^{G\to N})$, where $G = (V, E)$ is an arbitrary task graph with node features $\boldsymbol{\beta}^n$ and edge feature $\boldsymbol{\beta}^e$, N is an arbitrary device network with a set of devices D that has device compute features $\boldsymbol{b}^n$ and communication link features $\boldsymbol{b}^e$. The pseudocode is shown in Algorithm below.

Algorithm gpNet

function GPNET($G = (V, E, \boldsymbol{\beta}^n, \boldsymbol{\beta}^e), N = (D, \boldsymbol{b}^n, \boldsymbol{b}^e), \mathcal{M}^{G\to N}$)
  Initialize an empty graph $H = (V_H, E_H, \boldsymbol{x}^n, \boldsymbol{x}^e)$
  $V_{H,\mathcal{P}} = \{\}$ ▷ The set of pivots
  for $v_i \in V$ **do**
    $C_i = \{\}$ ▷ The node cluster of v_i
    for $d_j \in D_i$ **do** ▷ D_i: the placement constraint of v_i
      add node $u = (v_i, d_j)$ to V_H and C_i
      node feature of u: $x_u^n = f_n(\beta_i^n, b_i^n)$
      if $\mathcal{M}^{G\to N}(v_i) = d_j$ **then**
        add $u = (v_i, d_j)$ to $V_{H,\mathcal{P}}$
  for $(v_i, v_j) \in E$ **do**
    for $u_1 = (v_i, d_k) \in C_i, u_2 = (v_j, d_l) \in C_j$ **do**
      if $u_1 \in V_{H,\mathcal{P}}$ or $u_2 \in V_{H,\mathcal{P}}$ **then**
        add edge $c = (u_1, u_2)$ to E_H
        edge feature of c: $x_c^e = f_e(\beta_{ij}^e, b_{kl}^e)$
  return H ▷ gpNet representation of the placement

Each node in H represents a feasible placement of $v_i \in V$ on device $d_j \in D_i$, and is labeled (v_i, d_j). The node feature of (v_i, d_j) is a function, f_n, of the task feature β_i^n and the device feature b_j^n. The set of nodes for all possible placements of a task v_i forms a cluster $C_i = \{(v_i, \cdot)\} \subseteq V_H$. Those nodes, whose labels are in the current placement $\mathcal{M}^{G\to N}$, are called *pivots* and form a set $V_{H,\mathcal{P}} \subseteq V_H$. The subgraph induced by $V_{H,\mathcal{P}}$ contains all information about the current placement $\mathcal{M}^{G\to N}$. Non-pivot nodes, on the other hand, represents a potential task re-placement. Each node in H also corresponds to one action defined in the search problem (§3.1). This correspondence is exploited later in our neural network design (§3.3).

Edge $((v_i, d_k), (v_j, d_l))$ exists in H if and only if $(v_i, v_j) \in E$ and at least one of (v_i, d_k) and (v_j, d_l) is in $V_{H,\mathcal{P}}$. In this way, each non-pivot node (v_i, d_j) only has edges pointing to or from pivots that contain the current placement information of the parent tasks and child tasks , and thus, has a local graph structure corresponding to the one if a replacement of v_i to d_j happens. This design decision is in support of the local search idea that at most one task is moved each time. The edge feature of $((v_i, d_k), (v_j, d_l))$ is a function, f_e, of the data link feature β_{ij}^e and the communication link feature b_{kl}^e. The resulting graph H has $|V_H| = \sum_{i=0}^{|V|-1} |D_i|$ nodes and $|E_H| = \sum_{i=0}^{|V|-1} (|D_i||E_i|) - |E|$ edges, where $|E_i|$ is the degree of v_i in G. Fig.1(c) shows the gpNet of the example placement in (a) and (b). f_n and f_e can be any sensible functions that combine the raw features of the task graph and the device network (e.g., concatenation function).

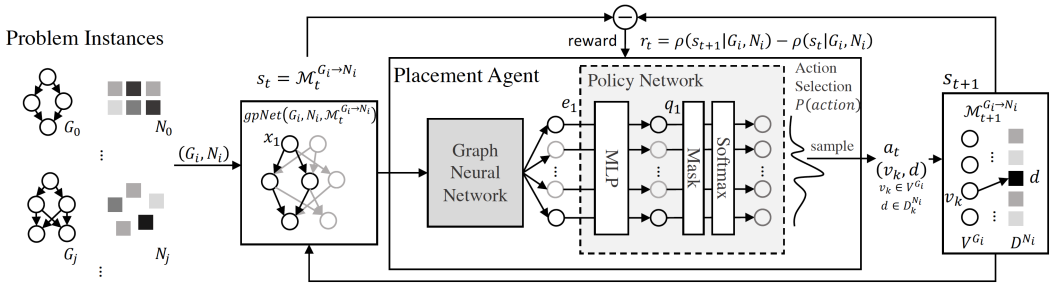


Figure 3: GiPH neural network design

Our proposed gpNet captures all network- and task-related features for making a placement update decision given a current placement. The original task dependencies in G and placement constraints are implicitly present in the output gpNet H through the way it is constructed. gpNet also generalizes to different problem instances and can represent any feasible placement of an arbitrary problem pair (G, N).

3.3 Neural Network Design

The learning framework of GiPH is visualized in Fig. 3. Given a placement problem of an arbitrary task graph G_i and target network N_i, the placement agent starts a search from a randomly initialized placement $\mathcal{M}_0^{G_i \to N_i}$. The agent then takes as the input the current state of the search, $\mathcal{P} = (G_i, N_i, \mathcal{M}_t^{G_i \to N_i})$, decides a placement update (an action defined in §3.1) that modifies the current placement, and observes the improvement of the performance objective ρ as the reward. It consists of a graph neural network and a policy network, which are jointly trained.

Scalable and generalizable graph embedding GiPH must first convert the state information into features to pass to the policy network. We use gpNet (§3.2) to generate a graph representation $gpNet(G_i, N_i, \mathcal{M}_t^{G_i \to N_i})$ of the current state. However, we still need to encode the graph-structured state information as vectors. Creating a flat vector representation is not scalable because it cannot handle graphs of arbitrary sizes and shapes (which depend on the specific task graph, target network and constraints).

GiPH achieves scalability using a graph neural network (GNN) that embeds the state information in a set of embedding vectors. Taking a gpNet as input with node features $\boldsymbol{x}^n$ and edge features $\boldsymbol{x}^e$ composed as described in §3.2, GiPH propagates information in a sequence of message passing step in the form of

$$e_u = h\left(\sum_{v \in \xi(u)} g([e_v \parallel x_{vu}^e])\right) + x_u^n, \tag{1}$$

where $\xi(u)$ is the set of parents of u, who have aggregated messages from all of their parents. $g(\dot{})$ and $h(\dot{})$ are non-linear transformations over vector inputs with trainable parameters. The message passing is done in both forward and backward directions with separate parameters, each summarizes information about the subgraph of nodes that can be reached and is reachable from u. GiPH concatenates the two summaries along each direction as the embedding of a node in gpNet. For a node with label (v, d), with embedding captures the local placement information if v is placed on d (i.e., if an action (v, d) is taken).

Adopting a GNN also helps generalizability because it automatically learns high-level features that are statistically important through end-to-end training, and the model learned can generalize (and scale) to unseen graphs.

Policy network and actions The policy network consists of a multi-layer perceptron (MLP), an optional mask layer, and a softmax layer (Fig. 3). We use the per-node embedding from GNN to

compute a score $q_a = g(e_a)$ for each action a in the action space A_{G_i,N_i} (represented as nodes in the gpNet). $g(\cdot)$ is a score function implemented as a MLP that computes a scalar value from an embedding vector. The score q_a quantifies how good an action is given the current state s. GiPH then uses a softmax layer to output a probability of selecting each action based on the score $P(a|s) = \exp(q_a) \sum_{b \in A_{G_i,N_i}} \exp(q_b)$. An optional mask layer can be placed before the softmax to mask out undesired actions. The final output is a probability distribution over all feasible actions

3.4 RL Training

GiPH uses a policy gradient method REINFORCE for training (Williams [1992]). During each episode, a placement problem (G, N) is sampled from a training set $\mathcal{G}_T \times \mathcal{N}_T$. Starting from a random placement s_0, the agent collects observations (s_t, a_t, r_t) at each step $t = 0, ..., 2|V| - 1$ following the current policy π_θ. It updates its policy parameters at the end of each episode

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{2|V|-1} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{2|V|-1} \gamma^{t'-t} r_{t'} - b_t \right), \tag{2}$$

where α is the learning rate, γ is the discounting factor, and b_t is a baseline for reducing the variance of the policy gradient (Weaver and Tao [2001]).

To improve the sample efficiency and force exploration, we mask out actions that do not change the current placement (e.g., a_0, a_1 at state $\mathcal{M}_0$ in the example Fig. 2(b)) because no new information will be acquired by taking those actions. We also mask out actions that will move the same task node as the immediate previous action did as moving the same node consecutively is inefficient.

4 Experiments

In this section, we evaluate the performance of our proposed GiPH for makespan minimization. We assume a heterogeneous computing environment where the computation time and communication time can be estimated from compute (task) and communication (data link) features of the device network (task graph). We consider randomly generated task graphs and device networks and compare GiPH with the following baseline algorithms:

- **Random placement sampling**: generate random placements of the task graph by sampling a feasible placement for each task from a uniform random distribution. This random baseline is representative of the average placement "quality" without GiPH's intelligent search.
- **Random task selection + earliest finish time device selection**: a heuristic for placement search, where at each step a task in the graph is randomly selected and placed to a feasible device that is expected to finish the task the earliest (Topcuoglu et al. [2002]).
- **GiPH task selection + EFT device selection**: the version of GiPH without using gpNet. At each step, instead of deciding both a task and a device, the RL agent only selects a task. The task is then placed to a feasible device that can finish the task the earliest.
- **Placeto** (Addanki et al. [2019]):
- **HEFT** (Topcuoglu et al. [2002]): one of the most well-known heuristic offline scheduling algorithm for heterogeneous computing, which assumes the perfect knowledge of compute and communication times.

Evaluation Metrics We compare 1) the *placement quality* of these placement algorithms and 2) the *generalizability*.

For the placement quality, we consider an objective of minimizing the makespan, which is the duration from the start to the end of the excution of the task graph. Since the makespan can vary significantly across different problem instances, it is necessary to normalize the makespan to a problem-dependent

lower bound. We use the Schedule Length Ratio (SLR) defined by

$$SLR = \frac{makespan}{\sum_{v_i \in CP_{MIN}} \min_{d_j \in D_i} w_{i,j}}, \tag{3}$$

where $w_{i,j}$ is the expected time of running task v_i on device d_j. This metric is first introduced in(Topcuoglu et al. [2002]). The denominator is the sum of the minimum computation cost of tasks along the critical path CP_{MIN}. The placement algorithm that gives the lowest SLR is the best with respect to the placement performance. Average SLR of different problem instances is used in the experiments for testing.

We measure the generalizability of the algorithms by testing how well they can adapt to network changes. The test is done by gradually removing devices and then putting new devices back in one by one. We record the placement quality of each algorithm as the device network become more different.

Dataset We implement a random task graph generator and a random device network generator with various characteristics depending on input parameters. For task graphs, those parameters include: the number of tasks, the depth/width ratio of the DAG, the edge connection probability, the average compute requirements of tasks, the average amount of data transmission, and the heterogeneity across tasks in a graph. For device networks, the input parameters include: the number of devices, the average communication bandwidth and delay, the average compute speed of devices, and the heterogeneity across devices in the network. Each task graph and device network has compute and communication properties that are draw from a uniform distribution centered around the average value with a range defined by the heterogeneity parameters. Our simulation-based framework allows assigning sets of values to the parameters used by the generators.

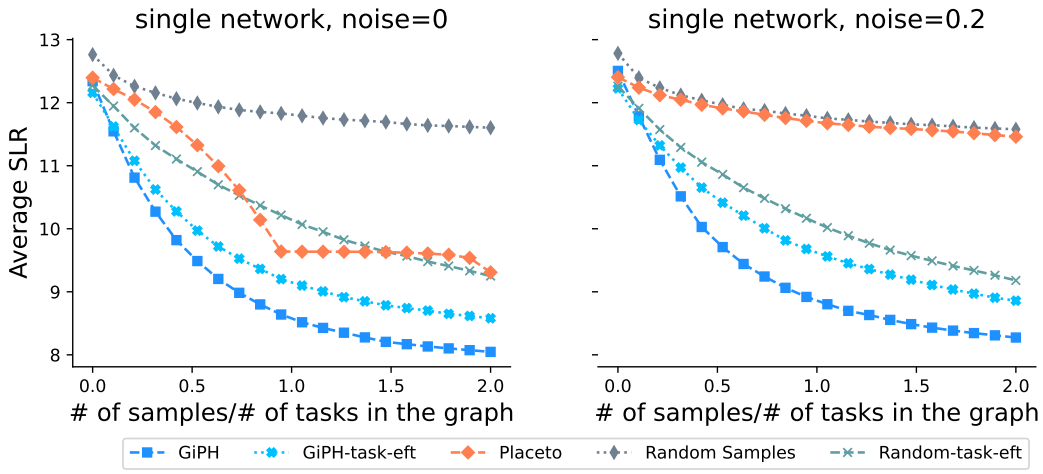
4.1 Placement Performance

We train GiPH and other learning-based methods using the same training dataset $\mathcal{G}_{train} \times \mathcal{N}_{train}$ and evaluate them every five episodes with a separate set of evaluation cases $\mathcal{G}_E \times \mathcal{N}_E$. The training stops when there is no further performance improvement on the evaluation set. Then, a larger set of test cases $\mathcal{G}_{test} \times \mathcal{N}_{test}$ is used to test the performance of each learned policy. To demonstrate the generalizability, the training set and test set are generated separately using different sets of random seeds to make sure the learned policy is evaluated on task graphs and device networks that are not seen during training (except for the single-device-network case).

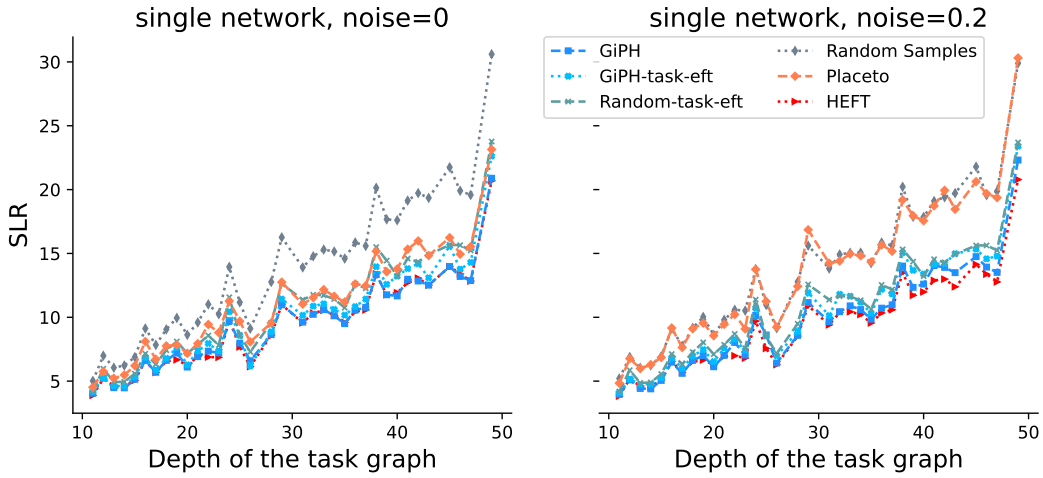
For a given test case (composed of an arbitrary task graph and device network), all search-based policies start from the same initial placement for fair comparison. The episode length is set to twice the number of computational tasks in the $2|V|$. Since Placeto fixes the number of search steps to $|V|$, we start a new search episode for Placeto after $|V|$ steps. Each policy outputs the SLR of the best placement found so far within a the episode. The average SLR across test cases is record as a function of the number of samples as each policy searches through the solution space.

Single-device-network case (task-graph generalization) We first evaluate GiPH's performance when there is only one single device network with and without 20% noise added to the computation and communication times. The noise is intended to model the randomness (unpredictability) of the communication and computation times in real systems. The training set contains 300 task graphs, and the learned policies are tested on a separate set of 300 graphs.

The average SLR across test cases as a function of the number of the search step is shown in Fig.4a. In all cases, our GiPH policy outperforms other search policies and more rapidly finds better placements within fewer number of search steps. It achieves up to 30.4% speadup compared to the random baseline, which represents the average placement "quality". GiPH also exhibits resistance to variance in the communication and computation times (noise). In contrast, the Placeto policy degrades considerably probably because the agent cannot de-couple the noise sources without a proper presentation of the device network. Fig.4b shows the SLR of the final placements found by different algorithms with respect to the depth of the task graph. GiPH outperforms other search-based methods



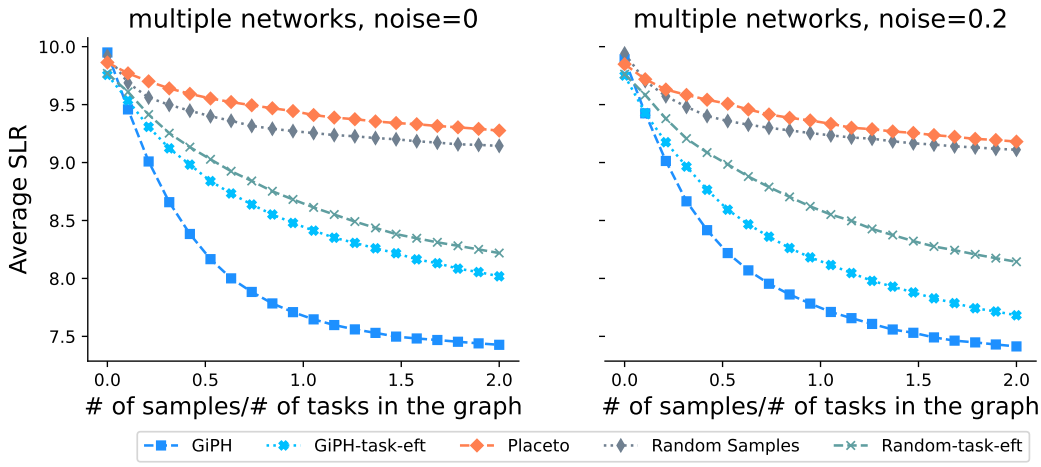
(a) Placement search efficiency when trained and tested on a single network



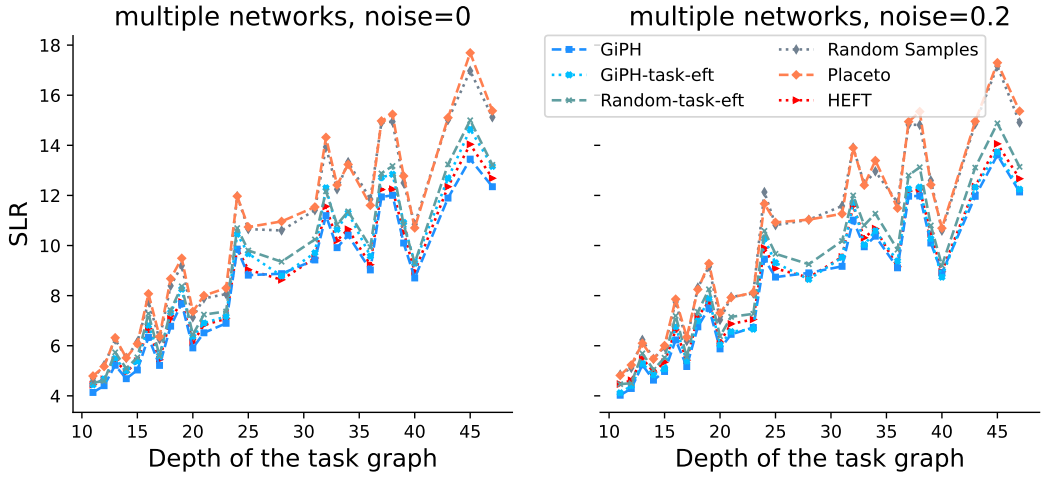
(b) Average SLR with respect to the depth of the task graph

in most of the cases and are comparable to the state-of-the-art HEFT. The results demonstrate the generalizability of the GiPH policy to unseen task graphs.

Multiple-device-network case (device-network generalization) We also look at the case where multiple device networks are used for training and testing. Since Placeto only works for a fixed number of devices, we generate fixed-sized device networks but with varying compute and communication capacities per device across generated networks instances. 10 device networks and 120 task graphs (a total of 1200 combinations for train cases), and the learned policies are tested on 500 sampled placement problems from 10 test device networks and 120 test graphs. In terms of average SLR across test cases (Fig. 5a), our GiPH policy again outperforms other search policies and find good placements within fewer steps. Placeto, on the other hand, is not able to distinguish between different device networks without encoding any device-level information. The policy could be even biased during the training process towards some false local optima that no longer exist in a new device network. Thus, with a biased sampling strategy, Placeto performs even worse than the random policy. The results demonstrate the generalizability of the GiPH policy across device networks.



(a) Placement search efficiency when trained and tested on multiple networks



(b) Average SLR with respect to the depth of the task graph

4.2 Generalizability

5 Conclusion

We presented GiPH, an RL-based framework for learning generalizable placement policies. We formulate the learning problem as a search problem such that the policy outputs incremental placement improvement steps. To generalize across different problem instances, we devise gpNet, a graph representation of placement for any task graphs and device networks. This novel graph representation together with a scalable and generalizable neural network design enables learning from an arbitrary placement problem. We demonstrate that GiPH is able to learn generalizable policies that produce better placement results than other baseline algorithms on unseen task graphs and device networks.

References

Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, and Mohammad Alizadeh. Placeto: Learning generalizable device placement algorithms for distributed machine learning, 2019.

H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):

260–274, 2002. doi: 10.1109/71.993206.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, page 538–545, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1558608001.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.