

RESEARCH ARTICLE

PHH: Policy-Based Hyper-Heuristic With Reinforcement Learning

ORACHUN UDOMKASEMSUB¹, BOONCHAROEN SIRINAOVAKUL¹,
AND TIRANEE ACHALAKUL^{1,2}

¹Department of Computer Engineering, King Mongkut's University of Technology Thonburi, Bangkok 10140, Thailand

²Government Big Data Institute, Bangkok 10900, Thailand

Corresponding author: Orachun Udomkasemsub (orachun.udo@kmutt.ac.th)


This work was supported by the Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program under Grant PHD/0065/2554.

ABSTRACT Hyper-heuristics have a high level of generality and adaptability, allowing them to effectively solve a wide range of complex optimization problems. With reinforcement learning, hyper-heuristics can use experience and knowledge gained to tackle unforeseen problems, allowing hyper-heuristics to adapt and improve over time. Our paper proposes a framework for using policy-based reinforcement learning to improve the performance of hyper-heuristics. The framework trains hyper-heuristic agents to select the best generalized constructive low-level heuristics to solve combinatorial optimization problems. The framework evaluation was performed using three benchmarking problems: traveling salesman, capacitated vehicle routing, and bin packing problems. The results showed that the proposed framework can outperform existing meta-heuristic and hyper-heuristic-based algorithms for all large problem instances in all problem domains. The proposed framework was also evaluated by applying it to a cost optimization problem for workflow scheduling on a hybrid cloud with a deadline constraint. Eight agents were trained on medium-sized workflows with two deadlines and tested against traditional meta-heuristic and hyper-heuristic methods to solve smaller and larger workflows with unforeseen deadlines. Four workflow applications, three workflow sizes, and three deadlines were used in the evaluation. The results showed that our proposed framework provided significantly better solutions of up to 98% for benchmarking problems, and up to 22% for cost optimization in workflow scheduling. Moreover, trained with small problem instances, the framework performed well for unforeseen larger problem instances implying its generalization. The proposed framework, thus, has the potential to improve both the generality and performance of solving large combinatorial optimization problems.

INDEX TERMS Combinatorial optimization, hyper-heuristics, policy optimization, reinforcement learning, workflow scheduling.

I. INTRODUCTION

Combinatorial optimization is an active research area for the last few decades across broad disciplines including operational research, computer science, finance, bioinformatics, industrial informatics, engineering, and data science. Some well-known combinatorial optimization problems are scheduling, timetabling, planning, resource and space allocation, cutting and packing, and engineering design. Workflow

The associate editor coordinating the review of this manuscript and approving it for publication was Daniel Augusto Ribeiro Chaves .

scheduling is another NP-complete combinatorial optimization problem that attracted many researchers in the past. Many works propose algorithms to solve this problem [1].

Meta-heuristics are flexible and can be adapted to different problem types. In recent years, many meta-heuristic methods have been proposed to solve combinatorial optimization [2], [3], [4]. However, heuristics are designed for specific problems and may require fine-tuning parameters. Additionally, solving large complex problems with heuristic methods may take a long time and may not output the global optimum within a reasonable time frame [5]. The performance

of heuristic methods also varies across different problem instances.

Hyper-heuristics have recently gained attraction from the research community to address this challenge. It is an alternative generality of computational search algorithms with an expectation for acceptable solution quality across problems or problem instances. Hyper-heuristics are high-level automated methodologies for selecting or generating a set of low-level heuristics (LLH) [6]. That is, hyper-heuristics automatically generate a combination of supplied heuristics to effectively solve the given problem. This can be efficient for problems where the appropriate heuristic is not known in advance.

Previous research on reinforcement learning shows promising performance for combinatorial problems as it has led to optimal solutions or solutions that are at least as optimal as solutions obtained by heuristic methods [7]. Reinforcement learning enables hyper-heuristics to capture knowledge and experiences used to solve problem instances and utilize them to solve unforeseen problems. Since the motivation for hyper-heuristics is to raise the level of generality but not to be competitive with state-of-the-art approaches [8], integration of hyper-heuristics with reinforcement learning must preserve the generality of hyper-heuristic search algorithms and their capability to obtain the optimal solution.

This study developed a framework for generalizing hyper-heuristics using policy-based reinforcement learning to solve combinatorial optimization problems by learning how to select constructive low-level heuristics to construct a solution.

One of the most widely used benchmark frameworks for hyper-heuristic is HyFlex [9], which provides a platform for evaluating hyper-heuristic approaches across various problem domains. However, HyFlex currently supports perturbative low-level heuristics. The evaluation of constructive low-level heuristics is thus limited. We then implemented some standard benchmarking problems existing in HyFlex to evaluate the proposed framework. We conducted the experiments on three well-known problem domains, namely traveling salesman, capacitated vehicle routing, and bin packing problems.

In addition, to test the efficiency of the proposed framework in solving real-world problems, we also applied it to the cost optimization workflow scheduling on a hybrid cloud with a deadline constraint. Large workflows mimicking real-world problems were used in the experiments.

These results from the evaluation show that hyper-heuristics can be trained to construct combinations of generalized constructive LLHs, which solve both benchmarking problems and the workflow scheduling more optimally when compared to traditional meta-heuristics and existing hyper-heuristics.

The rest of this paper is organized as follows. Section II discusses the literature on hyper-heuristics and reinforcement learning. Section III describes the proposed policy optimization-based hyper-heuristic framework to solve a

combinatorial optimization problem. Section IV shows an evaluation of the proposed framework with benchmarking problems. In addition, an application of the proposed framework to real-world cost optimization for a workflow scheduling problem was also evaluated. Experiment design and results are discussed, and the performance of the framework is evaluated. Section V finally summarizes this research.

II. RELATED WORKS

A. HYPER-HEURISTICS

Cowling and Soubeiga coined the term “hyperheuristics” in 2000 to represent the idea of “heuristics to choose heuristics” [10]. They also applied a choice function to determine which heuristic to choose for each decision in the employee scheduling problem [11]. More recent hyper-heuristics research involves using genetic programming to generate new heuristics that are suited to a specific problem or class of problems by combining components or building blocks of human-designed heuristics [12]. In 2019, Burke, E. K. et al. proposed a more general definition of the term hyper-heuristic as “an automated methodology for selecting or generating heuristics to solve computational search problems” [13]. A heuristic to be chosen by hyper-heuristics is a low-level heuristic or LLH. There are two types of LLHs: constructive LLHs, which are used to construct a solution, and perturbative LLHs, which are used to modify a solution in order to improve it.

Most of the existing literature on hyper-heuristics focuses on selecting perturbative LLHs [14]. Choong et al. [15] used a modified choice function to select perturbative LLHs for the neighborhood search mechanism in the Artificial Bee Colony algorithm to solve combinatorial discrete optimization problems. They evaluated their method on the traveling salesman problem. Alkhanak and Lee [16] proposed a hyper-heuristic for cloud cost optimization on scientific workflow scheduling. They used perturbative LLHs to repeatedly update solutions. The hyper-heuristic used a scoreboard, and the LLHs were selected based on the performance score from previous iterations. Lin et al. [17] also proposed a genetic programming hyper-heuristic algorithm to generate perturbative heuristics to solve multi-skill resource-constrained project scheduling problems. Kenari and Shamsi [18] proposed a framework to solve workflow scheduling problems on cloud environment. The framework classified workflows based on their features using a decision tree. The meta-heuristics used as perturbative LLHs include Genetic Algorithm (GA), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO). However, only one LLH was selected to generate a solution, and therefore several meta-heuristic LLHs cannot be utilized to solve a single problem instance.

According to the reviewed literature [15], [16], [17], [18], high-level heuristics are simple rule-based or meta-heuristics. While they can generalize to different problem instances, sizes, or types, they do not have the ability to capture

experiences from previously solved problem instances and use them to solve unforeseen ones. The goal of this research is to use the captured experiences of constructive LLH selection to enhance the capabilities of hyper-heuristics.

B. REINFORCEMENT LEARNING

Reinforcement learning [19] has been widely applied in various fields including robotics, communication and networking, natural language processing, games, self-organized system, scheduling management, resource management, and computer vision [20]. Mao et al. [21] designed a way to apply reinforcement learning to a multi-resource cluster scheduling problem. Several recent studies have demonstrated the use of reinforcement learning in job-shop scheduling problems [22], [23]. In addition, Melnik and Nasonov [24] proposed neural network scheduling, a workflow scheduling algorithm using reinforcement learning and an artificial neural network.

Reinforcement learning is a type of machine learning that allows a system to learn how to select the best action in a given situation by learning from trial and error. Unlike supervised learning methods, which require manually labeled training data, reinforcement learning generates training samples automatically during the training process. It can be used to solve very complex problems which might not be able to be solved by traditional techniques. Furthermore, it can capture knowledge from previously solved problem instances and use it to make decisions in new situations.

Q-learning [25] is a popular model-free reinforcement learning algorithm used for solving complex decision-making problems. In Q-learning, an agent learns an optimal policy by iteratively estimating the Q-value of each state-action pair using the Bellman equation. The Q-value represents the expected cumulative reward an agent will receive by taking a particular action in a given state. By continuously updating its Q-value estimates, the agent learns the optimal policy that maximizes its expected reward over time. Q-learning is attractive because it can handle continuous state spaces, is easy to implement, and can learn optimal policies from scratch. However, Q-learning is limited by its requirement for an accurate reward function, its sensitivity to large state spaces (the curse of dimensionality), and its inability to handle stochastic environments. Q-learning has been successfully applied to various domains, including gaming, robotics, and recommendation systems. For example, Q-learning has been used for robotic control tasks, such as grasping and manipulation, and for optimizing recommendations in online shopping platforms. In the context of hyper-heuristics, Q-learning can be used to optimize the heuristic selection and parameter tuning in combinatorial optimization problems.

While Q-learning uses the Bellman equation, DQN [26] is a variation of Q-learning that uses deep neural networks to approximate the Q-values. By using neural networks, DQN is able to handle high-dimensional state spaces and can learn to generalize across similar states. In addition, DQN uses

experience replay, a technique that stores past experiences in a buffer and randomly samples from them during training to reduce the correlation between successive samples. DQN is more complex and requires more computational resources than Q-learning. The algorithm can also be more sensitive to hyperparameter tuning. However, despite these limitations, DQN has been successfully applied to a variety of challenging problems, including Atari games and robotic control tasks.

In addition, there is another category of algorithm that finds the optimal Q-value using policy-based methods instead of state-action pairs. This type of algorithm learns the optimal policy directly, without explicitly computing the Q-values. Policy-based methods such as Actor-Critic methods and Trust Region methods are more efficient in problems with high-dimensional state space [27]. Actor-Critic method consists of two neural networks: an actor network that learns the policy and a critic network that estimates the value function. Trust Region method optimizes the policy by constraining the maximum change in the policy during each update.

Proximal Policy Optimization (PPO) [28] is also another popular policy-based method that combines ideas from both actor-critic and trust region methods. PPO optimizes the policy iteratively using a clipped surrogate objective. This objective will ensure that the policy change is not too large. In Comparison to value-based methods, policy-based methods are better suited for continuous action spaces and can handle stochastic environments more effectively. However, policy-based methods are generally more compute-intensive and can be more sensitive to the choice of hyperparameters.

Reinforcement learning algorithms, in particular policy-based methods, offer promising approaches for hyper-heuristics that can optimize selection in combinatorial problems. These algorithms do not rely on handcrafted heuristics and domain-specific knowledge. It can identify effective heuristics through trial and error. Moreover, these algorithms are suitable for continuous and stochastic environments, which are common in combinatorial optimization problems. However, generalizations must also be included in the design of reinforcement learning components, such as actions, state representation, and reward function.

C. REINFORCEMENT LEARNING FOR HYPER-HEURISTICS

The potential benefits of reinforcement learning have motivated research on its use in hyper-heuristics for automated low-level heuristic selection.

Falcão et al. [29] applied Q-learning-based hyper-heuristics to solve a scheduling problem in manufacturing systems. The proposed hyper-heuristic uses Q-learning to learn how to select perturbative meta-heuristics to gradually improve the schedule. It was evaluated against AutoDynAgents [30], a system that finds scheduling solutions using meta-heuristics with coordination and self-parameterization. The results showed that the Q-learning-based hyper-heuristic significantly outperformed the AutoDynAgents system.

Lin et al. [31] proposed a Q-learning-based hyper-heuristic to solve the semiconductor final testing scheduling problem. Eight easy-to-implement heuristics, wrapped by Simulated Annealing (SA), were used as LLHs. The Q-learning method was used to select perturbative SA-based LLHs to perturb the solution for a defined number of iterations. The proposed method was evaluated with 10 problem instances, and the average makespan was the lowest among the evaluated algorithms. However, the proposed method used online learning, so the Q-learning network parameters were re-initialized every run, and the algorithm could not utilize any experience across the runs.

Gölcük and Ozsoydan [32] proposed an algorithm recommender to select the most suitable bio-inspired algorithm for dynamic multidimensional knapsack problems (DMKP) using Q-learning and hyper-heuristic method. The LLHs used in this study include Artificial Bee Colony (ABC), Mantra Ray Foraging Optimization (MRFO), Salp Swarm Algorithm (SSA), and Whale Optimization Algorithm (WOA). The proposed algorithm solved four problem instances that have been dynamically changed with different change frequencies. The results from the recommender were compared with ones from the standalone bio-inspired algorithms. The results show that Q-learning is the best recommender algorithm. In this work, all DMKP instances are of the same sizes. The recommender requires a modification to generalize across different instance sizes. In addition, the LLHs in this work are meta-heuristics, which start from random solutions and iteratively perturb them until the stopping criterion is met.

Dantas et al. [33] evaluated a hyper-heuristic with a deep Q-network (DQN) selection strategy to select perturbative LLHs to iteratively improve solutions for the vehicle routing problem (VRP) and the traveling salesman problem (TSP). The results were compared to those obtained using Multi-Armed Bandit-based algorithms. The experiment results showed that the DQN selection strategy outperforms the other algorithms for 9 out of 10 VRP instances. While reinforcement learning can utilize knowledge from previously solved problem instances to solve new ones, this prior work did not demonstrate the use of the trained model to solve new problem instances.

Qin et al. [34] proposed a novel reinforcement learning-based hyper-heuristic using evolutionary meta-heuristics as perturbative LLH selection to solve heterogeneous vehicle routing problems. The proposed algorithm was trained on randomly generated problem instances and used to solve new ones. It outperformed six state-of-the-art meta-heuristics in 36 out of 38 instances, with average results up to 40% lower distance.

D. HyFLEX: A BENCHMARK FRAMEWORK FOR CROSS-DOMAIN HEURISTIC SEARCH

HyFlex [9] provides a generic architecture that allows researchers to create new hyper-heuristics by designing an LLH selection strategy and move acceptance mechanism.

It includes a set of ready-to-use components, such as benchmark problems and their corresponding low-level heuristics, and provides tools for performance comparison and visualization. The framework has been used in various research studies and competitions, demonstrating the effectiveness and versatility of hyper-heuristics in solving different optimization problems.

In HyFlex, the available LLHs are perturbative, meaning that they generate new solutions by making changes to the current solution. The perturbative heuristics consist of four types: mutation, cross-over, local search, and ruin-recreate. Hyper-heuristics in the HyFlex framework generally start with initializing a population of random solutions. A user defines a high-level search strategy that selects and combines these low-level heuristics to create new solutions. The high-level search strategy can be implemented using different mechanisms, such as a genetic algorithm, a tabu search, or a random search. The resulting solutions are evaluated using a move acceptance mechanism that determines whether the new solution is accepted or rejected. From the perturbative nature of HyFlex, the framework is not suitable for constructive low-level heuristics evaluation as detailed in the next section.

E. PERTURBATIVE VS CONSTRUCTIVE LOW-LEVEL HEURISTICS

Perturbative heuristics refer to a class of optimization algorithms that involve making small perturbations or changes to an existing solution in order to find a better solution. These algorithms are often used in situations where it is difficult or impractical to find an optimal solution using traditional optimization techniques. The basic idea behind perturbative heuristics is to start with a good solution and then make small, random changes to it in order to explore the search space for other potentially better solutions. This process is repeated multiple times, with the expectation that eventually the algorithm will converge on an optimal or near-optimal solution. Many research papers [15], [16], [17], [18], [29], [31], [33], [34] including the HyFlex framework use perturbative heuristics as low-level heuristics.

However, perturbative heuristics have some limitations. First, the initial solution or a starting point can have a significant impact on the final solution obtained. In some cases, even small changes to the initial solution can lead to significantly different results. This can be problematic in situations where there is a high degree of uncertainty in the problem. Second, the order in which perturbations are applied can have a significant impact on the resulting solution. Thus, in many cases, the approach requires additional experimentation and analysis to identify the optimal sequence of perturbations. Third, selecting stopping criteria or an appropriate number of iterations can be challenging. If the number of iterations is too small, the algorithm may not be able to explore the search space adequately for good solutions, resulting in underfitting. On the other hand, if the number of iterations is too large,

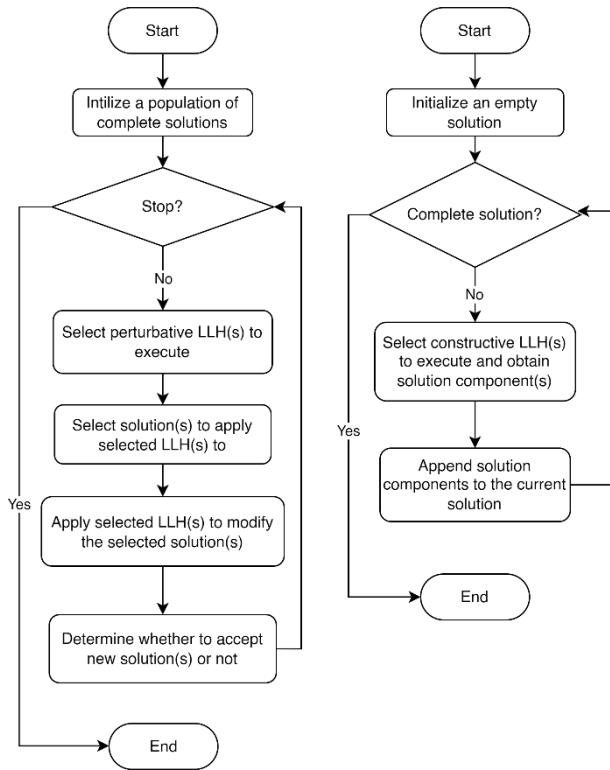


FIGURE 1. Comparison between hyper-heuristic approaches with perturbative (left) and constructive (right) low-level heuristics.

it can lead to overfitting or increased computational overhead unnecessarily.

Constructive heuristics, in contrast, build a solution iteratively by adding or selecting components one at a time until a complete solution is obtained. Hyper-heuristics with constructive LLHs start with an empty or partially complete solution and gradually add components to it in a way that satisfies the constraints of the problem. The constructive LLHs can refer to some predefined rules or strategies to select the next component to add, based on the current state of the solution. These rules can be based on local or global information about the problem, such as the cost or feasibility of different components or their impact on the objective function. Figure 1 compares the process to obtain a solution from perturbative LLHs (left) and constructive LLHs (right).

In many real-world problem domains, constructive LLHs will have advantages over perturbative LLHs because the approach does not depend so much on the quality of the initial solution. Constructive LLHs can start with an empty solution. Moreover, constructive LLHs are generally more efficient in terms of computational resources as the approach requires fewer evaluations of the objective function. Constructive LLHs add components to the solution iteratively, rather than generating and evaluating many perturbations. Thus, searching for the optimal solution will generally be simpler and faster. The number of iterations is limited to the

number of components in the complete solution and there is no need to fine-tune the stopping criterion.

Based on the discussion above, We proposed a policy-based hyper-heuristic framework (PHH) with benefits from reinforcement learning and constructive LLHs. Moreover, once trained, PHH can create solutions quickly as it does not require online training as in the previous works. In addition, the proposed framework has the potential to handle cross-problem domains, such as the semiconductor final testing scheduling problem in [31]. Furthermore, our proposed framework has the potential to solve optimization problems in dynamic environments, as described [32], by evaluating the changed state and selecting LLHs at each decision step. In this research, the proposed PHH framework was applied to solve a real-world problem: workflow scheduling in a hybrid cloud. In this problem domain, the execution speed and computation resources matter. The proposed framework aims to address the gaps in the existing literature with policy-based reinforcement learning. The learning process can utilize previous experiences on constructive LLHs selection at each decision step. The framework is thus expected to construct an optimal complete solution for different problem instances, sizes, or types. In addition to the proposed real-world problem, the framework was also evaluated with three problem domains in HyFlex for its generality.

III. MOTIVATION AND THE PROPOSED PHH FRAMEWORK

The existing hyper-heuristics are limited in their ability to capture experiences from previously solved problem instances and use them to solve unseen ones. To address this challenge, we propose a novel policy-based hyper-heuristic framework (PHH) with constructive LLHs and Proximal Policy Optimization (PPO). The proposed framework augments traditional hyper-heuristics by leveraging the strengths of reinforcement learning. Trained with small problem instances, the framework can offer solutions for unseen larger problems. Additionally, with constructive LLHs, the framework potentially provides more efficient algorithms than perturbative LLHs in solving combinatorial optimization problems. The use of PPO also allows our framework to obtain the optimal solution for different problem instances, sizes, or types.

The components and the training process of the framework are illustrated in Figure 2. The components of the framework include an environment, an agent that interacts with the environment, a policy, and an experience buffer. The policy represents a probability distribution over the available LLHs for each given environment state. The experience buffer is a storage of experience samples used for training and improving the policy. The inputs to the training process include a set of problems to be used for training, a set of available constructive LLHs, a solution quality function, a generalized reward function, and stopping criteria to determine when to end the training process. A set of available constructive LLHs for a traveling salesman problem, for example, might include

selecting the nearest city, the furthest city, or a random city. It is important to note that the given LLHs must be generalized enough to be compatible with any instances in the given problem sets. The stopping criteria of the training process are based on a fixed number of iterations or a threshold value to detect convergence. Once training is complete, the trained policy can be used to obtain solutions for new problem instances.

Figure 3 illustrates the components and process for obtaining a solution for a given problem instance using the trained policy. This process does not involve data sampling or policy updates, so there is no need for the experience buffer or rewards. It simply iteratively asks the trained policy for the best constructive LLHs to obtain a set of solution components until the complete solution is constructed.

The process of learning is described in Algorithm 1, while Algorithm 2 describes the process of obtaining a solution from the trained policy.

In the proposed PHH framework, an environment is a system or world in which the agent operates. The environment provides all the information to the agent as well as the appropriate reward signal. An agent is a learning entity that takes actions in the environment to maximize its reward or minimize its penalty. It makes observations from the environment and uses these observations to determine the current state of the environment as well as to decide which action to take next. A state is a representation of the current conditions or situation of the environment. A policy is a function that maps states of the environment to actions taken by the agent. It determines the behavior of the agent in a given state and is typically learned by the agent through interaction with the environment. The state transition is the change in the environment that occurs as a result of the agent taking an action. It includes the transition from one state to another and the resulting reward or penalty.

The agent begins by observing the initial state of the problem environment and applying the selected LLH from the initialized policy. The LLH returns a solution component to be integrated into the current solution, which is incomplete until it includes all necessary components to form a complete solution. For instance, in the traveling salesman problem, a solution component might be a city and the order it is visited. The solution is considered complete when all cities have been visited.

After integrating the obtained solution component into the current solution, the solution quality function evaluates the solution and returns a quality value that the agent uses as a reward for the selected action. The agent then observes changes in the environmental state and the obtained reward value, and uses this information to train the policy.

A. POLICY-BASED REINFORCEMENT ALGORITHM

This section outlines the reinforcement learning methods used in the proposed PHH framework. The learning algorithm in the PHH framework employs a policy-based learning method rather than a value-based method. Value-based

Algorithm 1 PHH Training Algorithm

Given

- $\mathbf{P} = \{p_i\}$: A set of training instances of the problem class
 - $\mathbf{L} = \{l_i\}$: Available constructive LLHs for the given problem class
 - $Q(X)$: Solution quality function
 - $R(s, l)$: Generalized reward function for the selected l on the given state s
 - STOP(): Stopping criteria
1. Initialize the policy π_θ
 2. Generate an environment of problem instance p from \mathbf{P}
 3. Let $n =$ number of components for a complete solution
 4. Let $X = \{\}$, an empty solution as an empty set of solution components
 5. **While** the stopping criteria STOP() is false, **do**
 6. Observe the current environment state s
 7. Obtain the next LLH l from the policy π_θ for the given state s
 8. Apply the selected LLH l to get a solution component Δx
 9. Append Δx to X
 10. Evaluate the quality of the (incomplete) solution $Q(X)$
 11. Observe the reward $r = R(s, l)$ and the updated state s'
 12. Store the experience (s, l, r, s') in the experience buffer
 13. **If** the experience buffer is big enough, **then**
 14. Train the policy π_θ with samples from the experience buffer
 15. **End if**
 16. **If** the solution is complete, $\text{count}(X) = n$, **then**
 17. Regenerate an environment of problem instance p from \mathbf{P}
 18. Reset the solution, $X = \{\}$
 19. **End if**
 20. **End while**
 21. Return the trained policy $\pi^* = \pi_\theta$

methods, such as Q-Learning and DQN, improve estimates of the quality value of state-action pairs and indirectly improve the policy. In contrast, policy-based methods directly evaluate and improve the policy itself.

In reinforcement learning, the policy defines the action that an agent should take for a given state. The policy can be either deterministic or stochastic. Figure 4 shows an example of a deterministic policy for 3×2 grid state space. That is, if the agent is on the state A1, B1, A2, or B2, it should go right. If the agent is on state C1, it should go down. In contrast, a stochastic policy specifies actions as a probability distribution over actions. For example, if the agent is in state B1, the probabilities of going right and going down may both be 0.5, since both actions will take two steps toward the goal. If the agent is in state C1, the probability of going right could be 0 and the probability of going down could be 1.

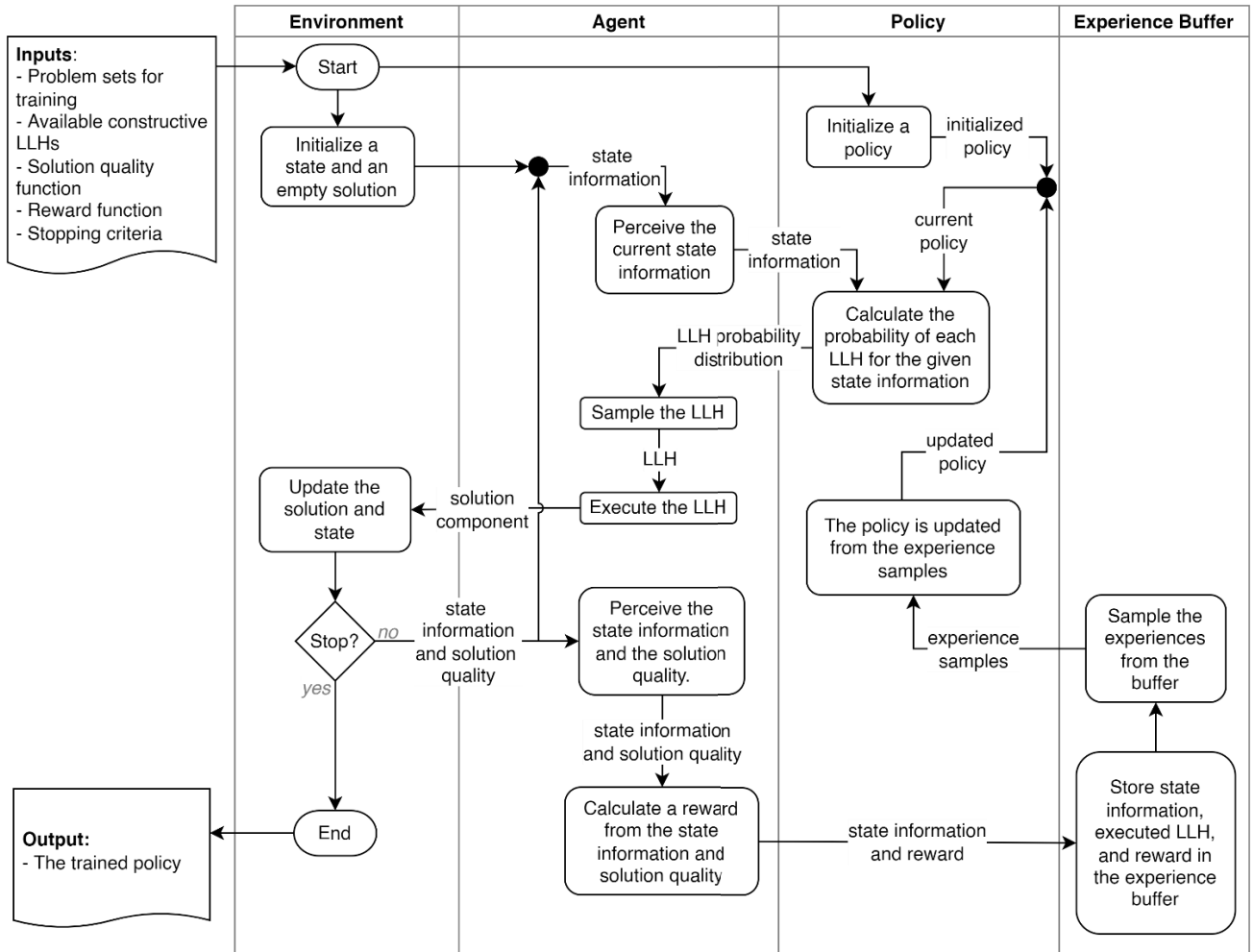


FIGURE 2. Components and training process of the PHH framework, which formulates the hyper-heuristic process as a reinforcement learning problem for selecting constructive LLHs to solve given problem sets. Each lane represents a component in the proposed framework.

This research employs an artificial neural network to represent the policy in the proposed PHH framework. This allows the policy to handle optimization problems with continuous actions and stochastic policies. The network is trained to optimize a performance objective function using policy gradient ascent to find an optimal policy. This function defines the goal or objective of the learning process, helping the agent learn which actions are more likely to lead to higher rewards and which actions should be avoided.

The objective of the PHH framework is to maximize the expected value of the cumulative discounted reward $J(\theta)$ obtained from the policy π_θ . The objective function is expressed in (1), where θ is weight parameters of the policy neural network, t is a time step, and r_t is the discounted reward of taking an action a_t at state s_t .

$$J(\theta) = \mathbb{E} \sum_{t \geq 0} [r_t | \pi_\theta] \quad (1)$$

The gradient ascent of the performance objective function can be represented by (2) [35], where $A_{\pi_\theta}(s_t, a_t)$ is an

advantage value function for taking an action a_t at state s_t , $Q_{\pi_\theta}(s_t, a_t)$ is a cumulative reward function for taking action a_t at state s_t , and $V_{\pi_\theta}(s_t)$ is a state value function for state s_t . The value returned by the Q_{π_θ} function is also known as the action value. The advantage value is the difference between the action value and the state value. It is used instead of the pure action value to distinguish between better and worse actions. It also reduces the variance of the policy gradient [36]. The proposed PHH framework uses another neural network to estimate the state value.

$$\begin{aligned} \nabla_\theta J(\theta) &\approx \sum_{t \geq 0} A_{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t | s_t) \\ A_{\pi_\theta}(s_t, a_t) &= Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s_t) \end{aligned} \quad (2)$$

The policy update in the proposed PHH framework is restricted to prevent the policy from worsening due to too large changes. The clipping objective as used in the PPO algorithm [28] is employed to achieve this. The gradient function in (3) is used instead, where $ratio_t$ is a probability ratio

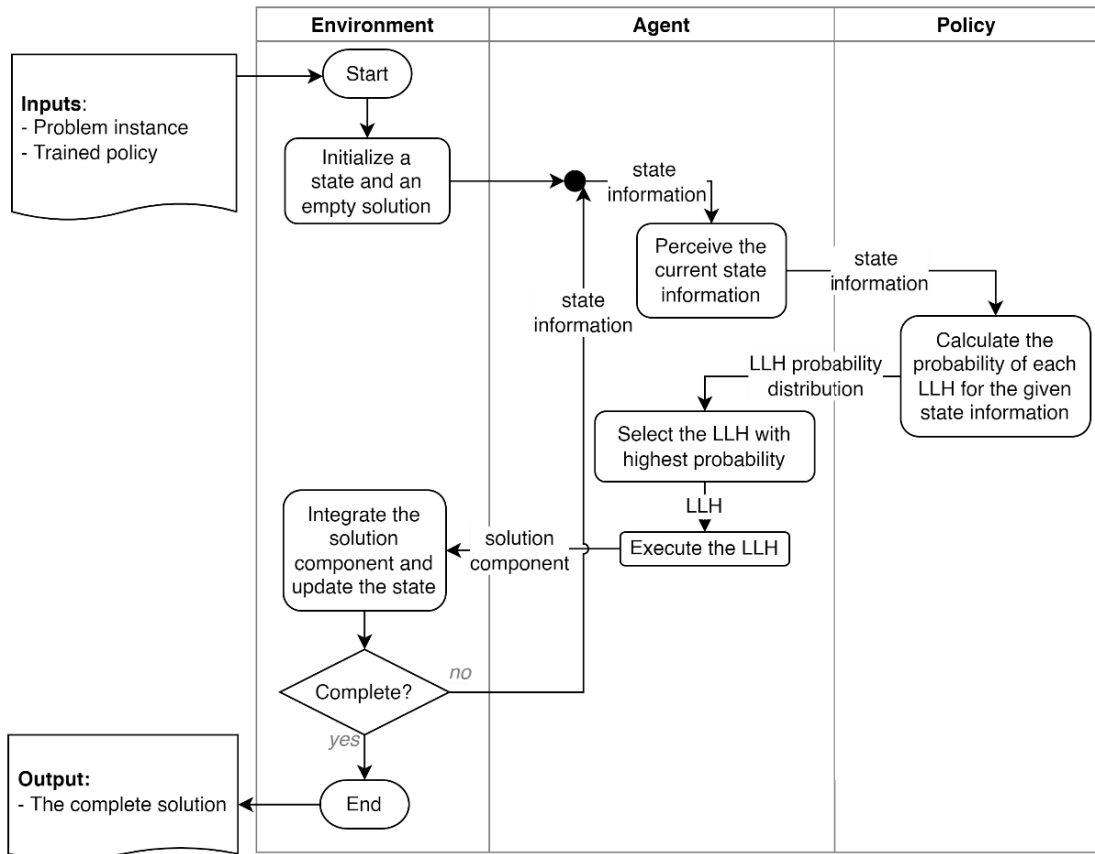


FIGURE 3. Components and process for obtaining a solution with the PHH framework, which formulates the hyper-heuristic process as a reinforcement learning problem to select constructive LLHs to solve a problem instance.

Algorithm 2 PHH Algorithm to Obtain the Solution

Given

- p : Problem instance
- π^* : Trained policy
- 1. Let n = number of components for a complete solution
- 2. Let $X = \{\}$, an empty solution as an empty set of solution components
- 3. **While** the solution is not complete, count $(X) < n$, **do**
- 4. Observe the current environment state s
- 5. Obtain the next LLH l from the policy π^* for the given state s
- 6. Apply the selected LLH l to get a solution component Δx
- 7. Append Δx to X
- 8. **End while**
- 9. Return the complete solution X

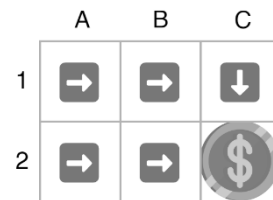


FIGURE 4. An example of a deterministic policy for 3 × 2 grid state space.

size of the policy update is restricted.

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[\min \left(\text{ratio}_t(\theta) A_{\pi_{\theta}}(s_t, a_t), \text{ratio}_t^{\text{clip}}(\theta, \epsilon) A_{\pi_{\theta}}(s_t, a_t) \right) \right]$$

$$\text{ratio}_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$$

$$\text{ratio}_t^{\text{clip}}(\theta, \epsilon) = \text{clip}(\text{ratio}_t(\theta), 1 - \epsilon, 1 + \epsilon) \quad (3)$$

between the current and the old policy, ϵ is a hyperparameter for clipping, and the clip function clips the value of ratio_t to be within the interval $[1 - \epsilon, 1 + \epsilon]$. This ratio is used to measure the relative improvement of the current policy compared to the previous policy. By clipping this value, the

In reinforcement learning, there are two methods to obtain samples during the training process: on-policy and off-policy [37]. On-policy methods involve learning the optimal policy while following it during the training process. The agent takes actions based on the current policy and updates the

policy based on the rewards it receives. These methods tend to converge faster because the agent is learning and improving the policy at the same time. Off-policy methods, on the other hand, involve learning the optimal policy using data gathered from a different policy. These methods can learn from a larger variety of experiences and can potentially learn a better policy, but they may require more data and may converge more slowly. In the proposed PHH framework, an on-policy approach is used. That is, the policy in steps 7 and 14 of Algorithm 1 is the same policy. Although the on-policy approach allows for faster convergence, it has a higher chance of getting stuck in suboptimal policies. However, the stochastic policy and clipping objective of the PPO algorithm address this challenge by allowing for exploration of the action space, while constraining the policy update to prevent it from being too greedy.

In the proposed PHH framework, the experience replay technique [38] is used to store the sampled data in a memory buffer, allowing the agent to learn from the samples multiple times and increasing sample efficiency. In reinforcement learning, rare events are important because they may have a significant impact on the learning process, but may occur infrequently. For example, in a game of chess, a rare event could be a game-winning move that is only possible in a very specific set of circumstances. If the agent does not encounter this event often, it may not learn how to respond to it effectively. By storing rare events in the replay buffer and sampling from them during training, experience replay ensures that the agent has multiple opportunities to learn from them. Thus, rare events are crucial in the learning process, and experience replay helps the agent to learn from them efficiently.

B. DESIGN FOR GENERALIZATION

One of the main goals of hyper-heuristics is to perform well on a wide range of problem domains, with little problem-specific knowledge or parameter tuning. To achieve this, the LLHs, a state representation, and a reward function must be designed for generalization. In other words, LLHs developed and tested on one problem domain can be used on other similar problems without significant modifications or adaptations.

The key challenges in hyper-heuristic RL are state representation and reward functions. The state representation determines how the hyper-heuristic will perceive the problem instance and what features it will use to make decisions. The reward functions enable the agent to effectively learn a policy that can solve a variety of problem instances. Functions that are too specific to a particular problem instance or type may result in a sub-optimal policy that fails to generalize to unseen problem instances.

In the proposed PHH framework, the generalized constructive LLHs, state representation, and reward function were designed so that the framework can be applied across problem instances, sizes, or domains. Details are provided next.

1) GENERALIZED CONSTRUCTIVE LLHS

In most reviewed literature, a different set of LLHs was provided for each different problem. However, an optimization problem in the real world does not come with a set of LLHs. This section explained an approach to provide generalized LLHs that can be used across optimization problems without adjustment.

The proposed PHH framework uses the selection of generalized constructive LLHs as available actions in the reinforcement learning process. These LLHs must be generalized across problem instances, sizes, or domains, and must provide a solution component that can be constructed into a feasible solution. The LLHs used in this framework include:

- Selecting the best solution component based on a minimum (or maximum) action quality indicator.
- Selecting the n^{th} best solution component to avoid local optima.
- Selecting the same solution component as in the previous time step.

2) GENERALIZED STATE REPRESENTATION

For generalization, the state representation should capture the essential characteristics of the problem instance that are relevant to the heuristics' performance. The state representation should also be robust to variations in the problem instance, such as changes in the size or structure of the problem. Some techniques that can be used to generalize the state representation include value normalization, moving window, or state encoding [39].

In the proposed PHH framework, the state representation in reinforcement learning is generalized to support different problem instances, sizes, or types. This means the number of dimensions and values of the state representation must be generalized. The state representation in the proposed framework contains the following information:

- Quality indicator value for the current solution constructed so far
- Values related to problem constraints
- Properties of the problem instance
- Properties and current state of the solution
- Quality indicator value after applying each constructive LLHs

These values are normalized and filtered by the moving window. These techniques can help make the state representation more adaptable to different problem instances, sizes, and types.

3) GENERALIZED REWARD FUNCTION

The reward function is the mechanism by which the agent learns to perform tasks that lead to desirable outcomes. The challenge is to configure the agent to perform well on the problem instances that it has not seen before.

The reward function in the proposed PHH framework can provide feedback for the taken action in a specific state across problem instances and types. The function may or may not

return the exact quality indicator value, but it will eventually reflect the change in the quality indicator value of the constructed solution. The reward value will thus be normalized. A reward function providing a positive generalized reward will be used for the maximization problem, whereas the negative one will be used as a penalty for the minimization problem. The reward function used in the proposed framework is in the form of (4) where R is a reward function, s_t is the state representation at the time t , a_t is the action taken at the time t , $Q(s_t)$ is the quality indicator for the state representation s_t , $C(s_t)$ is whether the state s_t satisfies problem constraints, and $P(s_t)$ is the penalty function for the state s_t .

$$R(s_{t-1}, a_t, s_t) = \begin{cases} Q(s_t) & ; C(s_t) \text{ is satisfied} \\ P(s_t) & ; \text{Otherwise} \end{cases} \quad (4)$$

While the constraint is satisfied, the reward value reflects the increment and decrement of a quality indicator value. When the constraint is not satisfied, the reward value represents a penalty. This mechanism ensures that the reward function can be used effectively for a range of different problem instances.

IV. EVALUATION OF THE PROPOSED PHH FRAMEWORK

A. ALGORITHM BENCHMARKING

This section evaluates the performance and generalization of the proposed framework against other optimization algorithms on multiple problem domains including the traveling salesman (TSP), capacitated vehicle routing (CVRP), and bin packing (BPP). These three problems have diverse characteristics and can be applied to many real-world applications. The comparison was made based on standard benchmark instances and different problem sizes are used to test for algorithms' generalization. The reason why the proposed PHH cannot be evaluated with HyFlex is that HyFlex supports only perturbative LLHs, whereas our framework uses constructive LLHs.

We implemented widely used optimization algorithms, namely, simulated annealing, genetic algorithm, and the DQN-based hyper-heuristic for comparison instead of using HyFlex. The DQN-based hyper-heuristic in this experiment used online training with DQN to mimic the Q-network-based and DQN-based hyper-heuristic from the previous works.

In this section, the results were presented and the generalization evaluation for practical use in real-world problems was discussed.

1) PROBLEM DOMAINS AND BENCHMARK INSTANCES

The Traveling Salesman Problem (TSP) seeks to determine the most efficient route for a salesman to visit a given set of cities and return to the starting city. The goal is to minimize the total distance traveled by visiting each city exactly once. TSP is generally used in logistics, transportation, and network routing.

The Capacitated Vehicle Routing Problem (CVRP) finds an optimal set of routes for a fleet of vehicles to deliver

TABLE 1. Traveling salesman problem instances.

Size	Instance	Cities
small	ch150	150
small	d198	198
small	pr144	144
small	rat195	195
medium	p654	654
medium	rat575	575
medium	d657	657
medium	rd400	400
large	pr2392	2,392
large	fnl4461	4,461
large	pcb3038	3,038
large	rl5915	5,915

goods or services to a set of customers, subject to capacity constraints. In CVRP, each vehicle has a limited carrying capacity, and during a route, the remaining load of the vehicle must be sufficient to accommodate the demand of the next customer to be visited. The problem also includes depots, which serve as replenishment points for the vehicles to refill their loads during the routes. The objective is to minimize the total distance traveled by vehicles while ensuring that each customer is visited exactly once, and the total demand of all customers served by a vehicle does not exceed its capacity. CVRP is used in logistics, delivery, and transportation planning.

The Bin Packing Problem (BPP) packs a set of items into a minimum number of bins, each with a limited capacity. In BPP, the objective is to minimize the total number of bins required while ensuring that each item is packed into exactly one bin and that the total size of the items packed into a bin does not exceed its capacity. BPP is used in logistics, manufacturing, and transportation, where it is necessary to minimize the number of containers, trucks, or storage units required to accommodate a set of items.

For each problem, we used standard benchmark instances widely used in the literature. For TSP, we selected 12 instances from TSPLIB [40], ranging from 150 to 5,915 cities. For CVRP, we selected 18 instances from TSPLIB as well, with varying numbers of vehicles and customers. For BPP, we selected 12 instances from BPPLIB [41] with different bin capacities and item sizes. A summary of problem instances for each problem domain is shown in Tables 1 to 3.

2) PARAMETER SETTINGS AND CONFIGURATION

In this experiment, four optimization algorithms: Genetic Algorithm (GA), Simulated Annealing (SA), Deep Q-Network (DQN), and the proposed PHH framework were implemented and evaluated. Parameters for each algorithm were specified as shown in Table 4. These parameters were empirically selected.

The proposed PHH-based algorithm was implemented in Python 3.10 with the Stable-Baseline3 Reinforcement Learning Library [42]. This library includes the PPO

TABLE 2. Capacitated vehicle routing problem instances.

Size	Instance	Customers	Vehicles	Avg demand	Capacity
small	eil23_v5	23	5	443.00	4,500
small	eil30_v5	30	5	425.00	4,500
small	eil33_v5	33	5	890.00	8,000
small	eil23_v20	23	20	443.00	4,500
small	eil30_v20	30	20	425.00	4,500
small	eil33_v20	33	20	890.00	8,000
medium	eil51_v5	51	5	15.24	160
medium	eila76_v5	76	5	17.95	140
medium	eilb76_v5	76	5	17.95	100
medium	eil51_v20	51	20	15.24	160
medium	eila76_v20	76	20	17.95	140
medium	eilb76_v20	76	20	17.95	100
large	eila101_v5	101	5	14.44	200
large	eilb101_v5	101	5	14.44	112
large	gil262_v5	262	5	46.21	500
large	eila101_v20	101	20	14.44	200
large	eilb101_v20	101	20	14.44	112
large	gil262_v20	262	20	46.21	500

TABLE 3. Bin packing problem instances.

Size	Instance	Items	Capacity
small	u250_00	250	150
small	u250_01	250	150
small	u250_02	250	150
small	u250_03	250	150
medium	u500_00	500	150
medium	u500_01	500	150
medium	u500_02	500	150
medium	u500_03	500	150
large	u1000_00	1,000	150
large	u1000_01	1,000	150
large	u1000_02	1,000	150
large	u1000_03	1,000	150

algorithm and provides the capability to implement a custom environment.

Each algorithm was run three times for each problem instance and the quality of the solutions was recorded. The solution quality for TSP and CVRP is represented by the traveled distance and for BPP by the number of used bins. The lower value represents the better solution.

3) RESULT AND DISCUSSION

In this section, we presented the results. Friedman’s [43] test and Nemenyi’s [44] post-hoc test were used to verify the significance of the differences and identify the performance ranks. The statistical testing was carried out using the Python package Scipy.Stats [45] and Scikit-Posthocs [46].

Tables 5 to 7 showed the average solution quality from each problem instance of TSP, CVRP, and BPP respectively. The best solution quality among all algorithms was marked in bold font. The P-values from Friedman’s test indicated that the solution quality from each algorithm was significantly different. All pairwise P-values from Nemenyi’s test were also less than 0.05, indicating that any pairs of algorithms produce significantly different results.

Tables 8 to 10 showed the average performance ranks of each algorithm for each instance size. The lowest rank among

TABLE 4. Parameter settings for each algorithm.

Parameters	Values
GA	
Population size	300
Mutation probability	0.04
Crossover probability	0.5
Number of generations	4,000
SA	
Alpha	0.996
Batches	7
Stopping Temperature	5.86e-06
DQN	
Training frequency	Every 5,120 steps
Learning rate	1e-4
Batch size	1024
Discount factor	0.99
Network	3 dense layers with ReLU: 128x64x32
Training timesteps	1000 episodes
PHH	
Training frequency	Every 5,120 steps
Number of epochs	5
Learning rate	0.01x
Batch size	1024
Clip range	0.2
Discount factor	0.99
Policy network	3 dense layers with ReLU: 128x64x32
Value network	2 dense layers with ReLU: 64x32
Training instances	TSP: rat195 VRP: eil51_v5 BPP: Falkenauer_u250_00
Training timesteps	2e7 steps

TABLE 5. The average distances of solution for TSP instances obtained from each algorithm.

Size	Instance	SA	GA	DQN	PHH
small	ch150	1.41E+4	3.78E+4	2.60E+4	1.73E+4
small	d198	4.01E+4	1.19E+5	6.46E+4	6.04E+4
small	pr144	1.62E+5	5.26E+5	3.51E+5	2.14E+5
small	rat195	5.86E+3	1.50E+4	9.40E+3	6.86E+3
medium	d657	3.86E+5	6.75E+5	1.74E+5	5.38E+4
medium	p654	6.57E+5	1.37E+6	1.37E+5	1.65E+5
medium	rat575	4.55E+4	8.52E+4	1.81E+4	7.63E+3
medium	rd400	7.82E+4	1.60E+5	4.43E+4	1.69E+4
large	fnl4461	6.55E+6	7.33E+6	2.02E+6	2.04E+5
large	pcb3038	3.93E+6	4.40E+6	1.23E+6	1.58E+5
large	pr2392	1.04E+7	1.24E+7	4.91E+6	4.32E+5
large	r15915	3.53E+7	3.74E+7	2.74E+6	6.92E+5

all algorithms, marked in bold font represents the algorithm providing the best solution quality.

For the TSP problem, while SA obtained the best results in small instances, our PHH algorithm performed from 75-98% better in most medium and all large instances. From Table 8, the result also showed that PHH produced the best average performance for medium and large instances.

For the CVRP problem, GA obtained the best results in most small instances, while PHH outperformed the other algorithms by 24-80% in all medium and large instances.

In the case of BPP, PHH obtained the best results with a 7-13% improvement in all instances of all sizes.

PHH performance was also compared to a DQN-based hyper-heuristic, where PHH outperformed DQN in all three problem domains.

These results demonstrated PHH’s effectiveness in solving larger and more complex problems, making it more practical

TABLE 6. The average distance of solutions for CVRP instances obtained from each algorithm.

Size	Instance	SA	GA	DQN	PHH
small	eil23_v5	8.14E+2	7.88E+2	1.41E+3	8.01E+2
small	eil30_v5	8.00E+2	7.45E+2	1.65E+3	7.21E+2
small	eil33_v5	1.03E+3	9.21E+2	1.60E+3	1.00E+3
small	eil23_v20	8.66E+2	8.09E+2	1.54E+3	1.26E+3
small	eil30_v20	8.29E+2	8.08E+2	1.76E+3	1.00E+3
small	eil33_v20	1.08E+3	1.15E+3	2.20E+3	7.85E+2
medium	eil51_v5	9.81E+2	1.02E+3	1.51E+3	5.72E+2
medium	eilA76_v5	1.62E+3	1.78E+3	2.26E+3	8.23E+2
medium	eilB76_v5	1.75E+3	1.84E+3	2.31E+3	1.02E+3
medium	eil51_v20	8.82E+2	1.06E+3	1.79E+3	9.96E+2
medium	eilA76_v20	1.31E+3	1.83E+3	2.59E+3	1.62E+3
medium	eilB76_v20	1.46E+3	1.81E+3	2.58E+3	1.82E+3
large	eilA101_v5	2.03E+3	2.23E+3	2.93E+3	1.11E+3
large	eilB101_v5	2.15E+3	2.41E+3	3.05E+3	1.05E+3
large	gil262_v5	1.87E+4	2.21E+4	2.36E+4	5.53E+3
large	eilA101_v20	1.67E+3	2.39E+3	3.36E+3	1.20E+3
large	eilB101_v20	1.75E+3	2.48E+3	3.47E+3	1.33E+3
large	gil262_v20	1.50E+4	2.33E+4	2.20E+4	4.74E+3

TABLE 7. The average number of bins of solutions for BPP instances obtained from each algorithm.

Size	Instance	SA	GA	DQN	PHH
small	u250_00	111.33	108.00	116.17	105.50
small	u250_01	112.67	108.50	119.89	105.67
small	u250_02	115.33	111.50	122.44	107.33
small	u250_03	112.33	110.00	119.11	106.67
medium	u500_00	231.00	219.50	243.83	211.33
medium	u500_01	234.33	226.00	255.00	212.33
medium	u500_02	235.67	234.00	285.33	213.67
medium	u500_03	240.33	226.00	250.67	215.33
large	u1000_00	478.00	455.00	452.00	421.67
large	u1000_01	488.33	463.20	469.33	428.00
large	u1000_02	498.33	464.20	499.00	432.00
large	u1000_03	499.33	469.20	489.00	432.67

TABLE 8. Average performance ranks of each algorithm for each instance size of TSP.

Size	SA	GA	DQN	PHH
small	1	4	3	2
medium	3	4	1.75	1.25
large	3	4	2	1

TABLE 9. Average performance ranks of each algorithm for each instance size of CVRP.

Size	SA	GA	DQN	PHH
small	2.5	1.5	4	2
medium	1.5	2.83	4	1.67
large	2	3.17	3.83	1

TABLE 10. Average performance ranks of each algorithm for each instance size of BPP.

Size	SA	GA	DQN	PHH
small	3	2	4	1
medium	3	2	4	1
large	3.75	2.25	3	1

for real-world problems that need generalizations. Moreover, PHH was able to solve large problem instances, even though it had been trained only with smaller instances.

PHH, however, may not be suitable for small problem instances where the overhead of offline learning is not worth the potential efficiency gains. PHH works well in scenarios where large problem instances need to be frequently solved.

From the results, the PHH algorithm shows the potential to become a valuable tool in solving complex optimization problems in the future.

B. APPLICATION TO COST OPTIMIZATION WORKFLOW SCHEDULING WITH A DEADLINE CONSTRAINT IN HYBRID CLOUD

Cloud computing offers virtually unlimited resources, with the added benefits of elasticity and auto-scaling to eliminate the costs of over-provisioning and the risks of under-provisioning. Organizations can use a hybrid cloud configuration that combines public and private clouds to achieve greater cost-effectiveness with the added benefit of elasticity. Hybrid clouds provide flexibility in managing workloads and data placement. Thus, scientific applications can be designed as a workflow and executed periodically on a hybrid cloud, which can provide both quality of service and cost-effectiveness.

However, the scheduling optimization problem for workflow applications is NP-complete [47], and state-of-the-art algorithms are meta-heuristics that require long computation time for larger problem sizes. In this section, we explained how PHH can be applied to address cost optimization for a workflow scheduling problem with a deadline constraint. The experiment was designed to evaluate the performance, generalization, and learning of the proposed method as discussed below.

1) PROBLEM FORMULATION

This section explains how to formulate the cost optimization problem for scientific workflow scheduling on a hybrid cloud with a deadline constraint.

A workflow application is represented by a directed acyclic graph $W = (T, E)$, where nodes represent a set of tasks $T = \{t_i\}$, and edges represent their dependencies $E = \{e_{ij}\}$. The dependency e_{ij} indicates that the output of the parent task t_i will be used as an input by the child task t_j . A task can only be executed after all its parent tasks, represented by $\text{succ}(t_i)$, have been completed and all input data is available. The workload of the task t_i is represented by $\text{len}(t_i)$ in millions of CPU instructions (MIs). The amount of data for task dependency e_{ij} is represented by $\text{len}(e_{ij})$ in megabytes (MB).

Figure 5 illustrates an example of a workflow application with five tasks. This workflow can be represented by the graph $W = (T, E)$ with a set of tasks $T = \{t_1, t_2, t_3, t_4, t_5\}$ and a set of dependencies $E = \{e_{12}, e_{13}, e_{24}, e_{34}, e_{45}\}$. For the task t_4 , its workload is $\text{len}(t_4) = 4$ MIs, its parent tasks are $\text{succ}(t_4) = \{t_2, t_3\}$, and its input dependencies are $\{e_{24}, e_{34}\}$. The amounts of data for these input dependencies are $\text{len}(e_{24}) = 3$ MB and $\text{len}(e_{34}) = 4$ MB. The task t_4 can only be started once the task t_2 and t_3 finish and output files 2-4.txt and 3-4.txt are available.

The hybrid cloud in this research consists of virtual machine (VM) instances. For the public cloud, the VM instances can be created on demand, whereas for the private

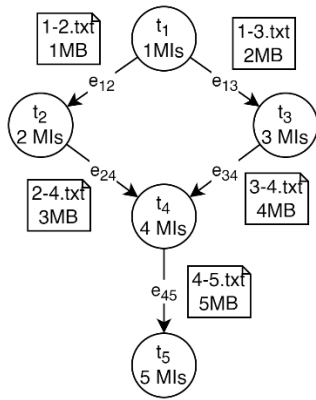


FIGURE 5. An example of a workflow application with five tasks.

cloud, VMs are already provisioned. The available VMs can be represented by $\mathbf{VM} = \{vm_i\}$; $vm_i = (cu_i, bw_i, up_i)$, where cu_i represents the computing capacity of the vm_i in millions of instructions per second (MIPs), bw_i represents the bandwidth of the vm_i in megabytes per second (MBps), and up_i represents a unit price of using the vm_i in USD per second. This research assumes that the execution time of executing a task t_i on the VM vm_j can be calculated using (5). The communication time to transfer intermediate data between tasks i and j can be calculated using (6), where $bw_{vm(t_i)}$ is the bandwidth of the VM that is assigned to execute the task i .

$$\text{exec}(t_i, vm_j) = \text{len}(t_i) / cu_j \quad (5)$$

$$\text{comm}(i, j) = \begin{cases} \text{len}(e_{ij}) / b(i, j) & ; vm(t_i) \neq vm(t_j) \\ 0 & ; \text{otherwise} \end{cases} \quad (6)$$

$$b(i, j) = \min(bw_{vm(t_i)}, bw_{vm(t_j)})$$

A workflow schedule can be represented by $\text{sch} = (\mathbf{TS}, \mathbf{M})$, where \mathbf{TS} is a sequence of tasks to be executed, and \mathbf{M} is a set of mappings between a task and the assigned VM.

Given that any task can be executed after all of its parent tasks are completed and all its input data are transferred to the assigned VM, the start time of the task t_i can be calculated by (7), where $vm(t_i)$ is the VM that is assigned to execute the task t_i , $f(t_i)$ is the finish time of the task t_i , and $\text{avail}(vm(t_i))$ is the time that the assigned VM is ready to execute the task. The finish time of the task t_i can be calculated by (8).

$$s(t_i) = \max \left(\begin{array}{l} \max_{t_p \in \text{parent}(t_i)} [f(t_p) + \text{comm}(p, i)], \\ \text{avail}(vm(t_i)) \end{array} \right) \quad (7)$$

$$f(t_i) = s(t_i) + \text{exec}(t_i, vm(t_i)) \quad (8)$$

After scheduling all tasks, the makespan and cost of the schedule can be calculated by (9) and (10).

$$\text{makespan} = \max_{t_i \in T} f(t_i) \quad (9)$$

$$\text{cost} = \sum_{t_i \in T} ([f(t_i) - s(t_i)] \times up_{vm(t_i)}) \quad (10)$$

The goal of this experiment is to find the optimal schedule for a workflow on a hybrid cloud, which minimizes

the cost of execution while meeting the deadline constraint. To achieve this, the proposed method uses a combination of hyper-heuristic and reinforcement learning techniques to find a schedule that uses the most cost-effective VMs while completing the workflow within the specified deadline.

2) CONFIGURATION OF PHH ALGORITHM FOR WORKFLOW SCHEDULING OPTIMIZATION

To implement a workflow scheduling algorithm from the proposed PHH framework, the problem is formulated as a reinforcement learning problem. In this formulation, the process of scheduling tasks to VMs is viewed as a sequence of actions taken by an agent. At each time step, the agent chooses an action by selecting a task to schedule to a VM. The schedule is updated as a result of this action, along with the makespan and cost of the schedule. The agent then observes the cost as a penalty and stores it, along with the corresponding state and action, in an experience buffer that will be used to train the policy. The training process can be illustrated in Figure 6. Once the agent has been trained, it can use the trained policy to obtain a schedule for a given workflow, a set of available VMs, and a deadline constraint, as illustrated in Figure 7. To implement the algorithm from this framework, it is necessary to define the available LLH actions, the generalized state representation, and the generalized reward function.

a: LLH ACTIONS FOR WORKFLOW SCHEDULING PROBLEM

An LLHs action represents a heuristic of how to select a task to schedule next and a VM to execute the next selected task. LLHs actions are formulated as available actions. In our application to the workflow scheduling problem, a sequence of tasks to schedule is generated using the task ranking process in the HEFT [48] algorithm. The possible LLHs actions for VM selection are listed in Table 11. These heuristics are designed to be generalized across different workflow instances, deadlines, and sets of VMs, rather than being specific to a particular problem instance or size. There are three criteria for selecting the VM, unit cost, capacity, or worthiness. Worthiness measures the capacity per 1 USD unit cost. Four ranks of each criterion are used as available actions as they are expected to increase the chance to avoid local optima. For example, action numbers 5 and 6 will select the VM with the 1st and 2nd highest capacity, whereas action numbers 10 and 11 will select the VM with the 2nd and 3rd highest worthiness value, respectively.

b: STATE REPRESENTATION FOR WORKFLOW SCHEDULING PROBLEM

The state representation in the proposed PHH framework captures the current state of the scheduling environment at a given time step. This includes the information on the problem instances (workflow, available resources (VMs) information, and the deadline constraint) and scheduling state (ready tasks, remaining tasks, VM states, cost, and makespan). To enable generalization across different problem sizes, a moving

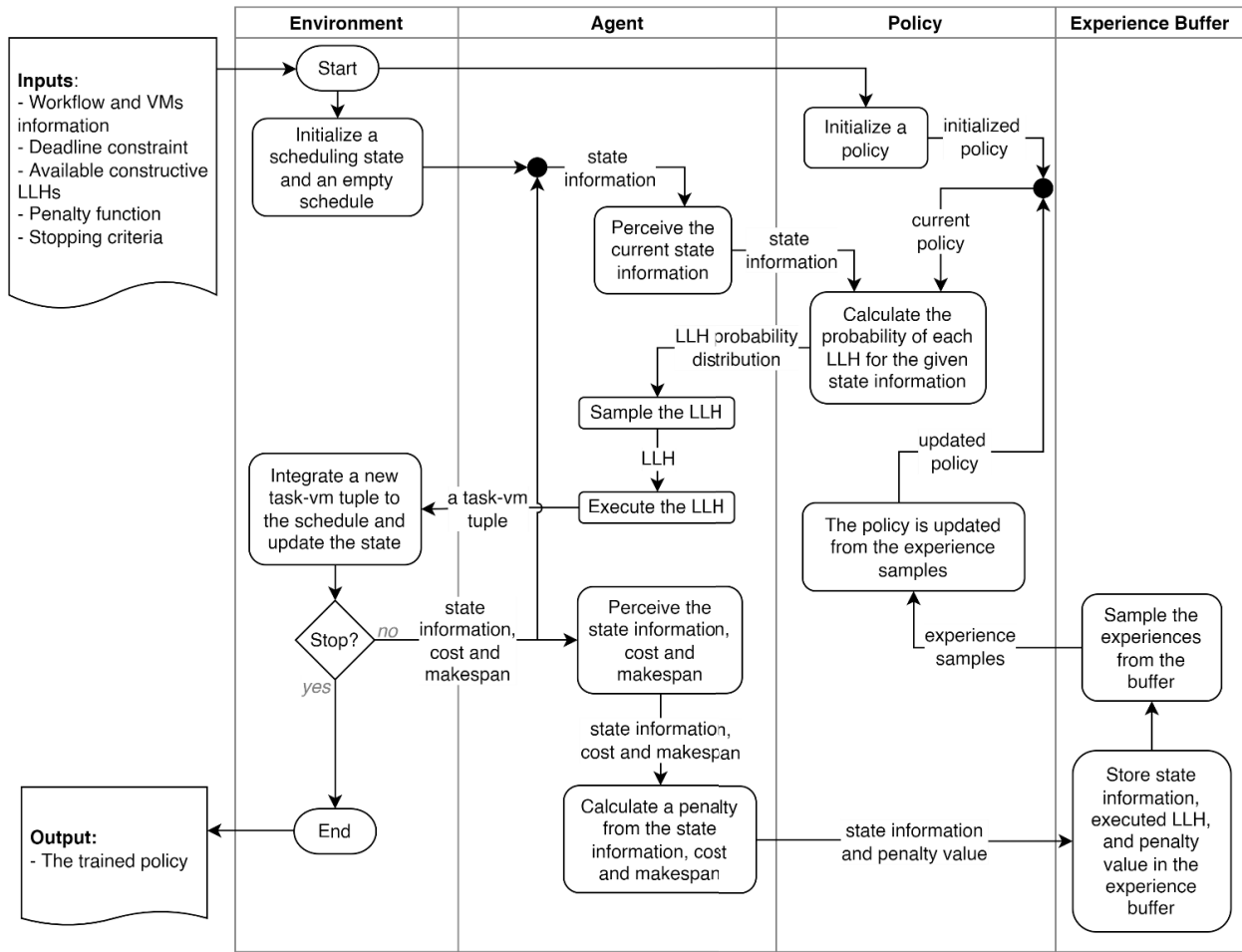


FIGURE 6. The training process of the PHH algorithm to obtain a schedule for workflow scheduling optimization problems.

TABLE 11. Available Low-Level heuristic actions.

Action Number	Low-Level Heuristic
1	Select the VM with the 1 st lowest unit cost
2	Select the VM with the 2 nd lowest unit cost
3	Select the VM with the 3 rd lowest unit cost
4	Select the VM with the 4 th lowest unit cost
5	Select the VM with the 1 st highest capacity
6	Select the VM with the 2 nd highest capacity
7	Select the VM with the 3 rd highest capacity
8	Select the VM with the 4 th highest capacity
9	Select the VM with the 1 st highest worthiness
10	Select the VM with the 2 nd highest worthiness
11	Select the VM with the 3 rd highest worthiness
12	Select the VM with the 4 th highest worthiness

window method is used to maintain constant dimensions for the state representation regardless of the size of the workflow or the number of VMs. The state representation includes the following dimensions, with μ_t representing the task moving window size and μ_{vm} representing the VM moving window size. All values in the state representation are normalized.

- 1: Makespan so far
- 2: Cost so far
- 3: Deadline

- Next μ_t dimensions: Workload of each remaining task (moving window)
- Next μ_t dimensions: Critical path length from each remaining task (moving window)
- Next μ_t dimensions: Number of child tasks for each remaining task (moving window)
- Next μ_{vm} dimensions: A finish time of the selected task on each VM (moving window)
- Next μ_{vm} dimensions: Whether each VM can complete the selected task within its sub-deadline (moving window)
- Next μ_{vm} dimensions: Computing capacity (MIPs) of each VM (moving window)
- Next μ_{vm} dimensions: Unit cost of each VM (moving window)

The task moving window and VM moving window can be obtained by using Algorithms 3 and 4 respectively.

c: REWARD FUNCTION FOR WORKFLOW SCHEDULING PROBLEM

The reward function in the proposed PHH framework for the workflow scheduling optimization problem is designed

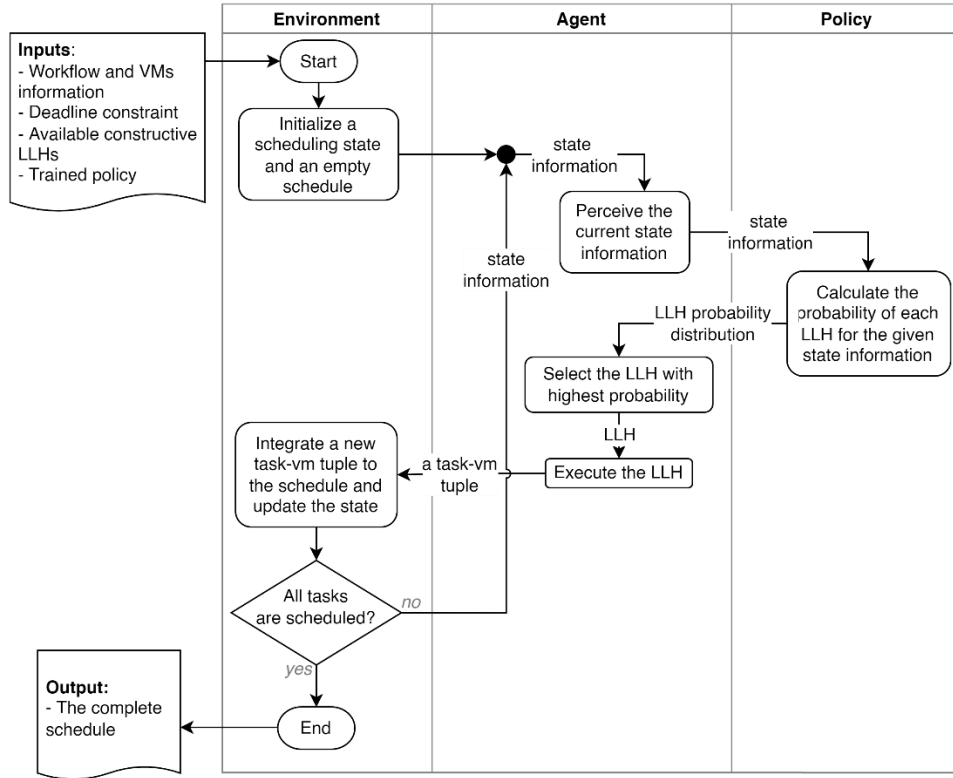


FIGURE 7. Process of PHH algorithm to obtain a schedule for workflow scheduling optimization problem from the trained PHH.

Algorithm 3 Create a Task Moving Window

1. Let μ_t = a task window width
2. Let T = a set of workflow tasks
3. Remove scheduled tasks from T
4. Sort tasks in T from highest to lowest critical path length
5. T_μ = Select the first μ_t tasks from the sorted T
6. **If** the number OF tasks in T_μ less than μ_t **then**
7. Add padding task information with 0 until the number of tasks in T_μ is μ_t
8. Return T_μ as the task moving window

Algorithm 4 Create a VM Moving Window

1. Let μ_{vm} = a VM window width
2. Let V = a set of all available VMs from private and public cloud
3. Let t = the next task to be scheduled
4. Remove any VMs from V that cannot finish the task t within its sub-deadline
5. Sort VMs in V from lowest to highest execution time for the selected tasks t
6. V_μ = Select the first μ_{vm} VMs from the sorted V
7. Return V_μ as the VM moving window

to incentivize the agent to select VMs and tasks that will minimize the overall cost and meet the deadline constraint. In this case, the cost that occurs from executing a task on a VM is treated as a penalty to the reinforcement learning agent, and the model aims to minimize this penalty. The deadline constraint is also taken into account by adding a higher magnitude penalty for schedules that exceed the deadline. The reward function can be described in (11), where $time$ is the current time step, and d is the deadline. The $penalty$ for exceeding the deadline is calculated using (12) by adding the exceeding execution time with the total executing time of all remaining tasks on the slowest VM. This amount of time is then multiplied by the most expensive unit

cost among all VMs.

$$reward(time, t_i, vm_j) = \begin{cases} cost_{time-1} - cost_{time}; & f(t_i) \leq d \\ -cost_{time} - penalty; & \text{Otherwise} \end{cases} \quad (11)$$

$$penalty = \left(\max_{i \in VM} up_i \right) \times \left[f(t_i) - d + \sum_{t_i \notin \text{scheduled}(T)} \left(\frac{\text{len}(t_i)}{\min_{i \in \text{len}(VM)} cu_i} \right) \right] \quad (12)$$

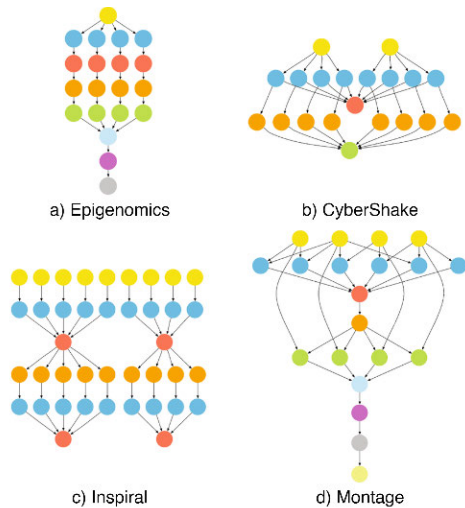


FIGURE 8. Scientific workflow applications for benchmarking [49].

3) EXPERIMENT DESIGN

This section describes the experimental design to evaluate the performance of the proposed method. The objective of the experiment is to evaluate the performance of the proposed method across the different training environments and testing problem instances, along with the following hypotheses.

- 1) The PHH-based method can learn to optimize workflow scheduling in terms of cost, resulting in lower-cost schedules after learning.
- 2) The PHH-based method can provide solutions to unforeseen problem instances of different workflow applications and sizes that are of equivalent or better quality than those produced by meta-heuristic algorithms.

a: EXPERIMENTAL WORKFLOWS

The workflow applications used in this experiment are well-known workflows [49] obtained from [50]. Their structures are based on real-world workflows: CyberShake (C), Epigenomics (E), Inspiral (I), and Montage (M). Figure 8 illustrates the structure of workflows in this experiment. Table 12 provides information on the characteristic of each workflow. Different structures of workflow applications were used to evaluate the generalization ability of the proposed framework. The objective is to see whether PHH can provide optimal solutions for the different workflow applications.

b: SIMULATION SETUP

To evaluate the first hypothesis that the PHH-based method can learn to schedule workflows, environments were created to train reinforcement learning agents. Table 13 and Table 11 provide information on available VMs including their capacity, bandwidth, unit cost, and number of VMs. The public cloud has no limit on the number of VMs to use, whereas the private cloud has no cost of execution on its VMs. Table 15 lists the deadlines used in the experiment. The 1x deadlines were calculated from makespan using the HEFT scheduling

TABLE 12. Workflow characteristics including the number of tasks, number of task dependencies, average data size in megabytes (MB), and average task size in millions of instructions (MIs).

Workflow / Size	Number of Tasks	Number of Task Dependencies	Average Data Size (MB)	Average Task Size (MIs)
C/S	30	52	117.08	27.62
C/M	100	192	187.92	24.49
C/L	1000	1,984	149.62	22.80
E/S	24	27	34.11	738.34
E/M	100	122	10.65	4,034.00
E/L	997	1,218	9.45	1,909.61
I/S	30	35	0.32	220.57
I/M	100	118	0.30	221.71
I/L	1000	1,249	0.31	225.82
M/S	25	45	3.76	9.11
M/M	100	232	3.08	10.81
M/L	1000	2,482	2.89	11.44

TABLE 13. Public cloud resources with capacity in million instructions per second (MIPs), bandwidth in MB per second, and unit cost in USD per second.

VM Size	Capacity (MIPs)	Bandwidth (MBps)	Unit Cost (USD/second)
Small	1.70	37.50	0.00060
Medium	3.75	81.25	0.00113
Large	7.50	81.25	0.00225
XLarge	15.00	125.00	0.00450
2XLarge	30.00	125.00	0.00900

TABLE 14. Private cloud resources.

VM Size	Capacity (MIPs)	Bandwidth (MBps)	Number of VMs
Small	2.00	40.00	3
Medium	3.00	60.00	5
Large	5.00	80.00	2

algorithm on each workflow in the hybrid cloud. During the training process, the task workload in the training workflow was randomized using (13) to increase the diversity of the training environments for the agents. In this experiment, the parameter λ was set to 1.5.

$$\text{len}'(t) = \text{uniform}(\text{len}(t), \lambda \text{len}(t)) \quad (13)$$

In this experiment, eight reinforcement learning agents were trained using different workflows and deadlines. The workflows with 100 tasks were used to create training environments with 1x and 16x deadlines, as summarized in Table 16. The suffix of each agent's name is the first letter of the training workflow followed by the deadline used in training. For example, the agent named PHH-C1X was trained with CyberShake workflow with 100 tasks and 1x deadline (13.74 seconds).

c: COMPARATIVE ALGORITHMS

To evaluate the second hypothesis that the trained reinforcement learning models can provide a solution to unforeseen problem instances, each trained agent was evaluated with all

TABLE 15. Deadlines for each workflow used in the experiment (seconds). 1x deadline was derived from a makespan of schedule obtained by the HEFT scheduling algorithm.

Workflow /Size	Deadline (second)				
	1x	2x	4x	8x	16x
C/S	13.81	27.62	55.24	110.48	220.96
E/S	207.09	414.18	828.36	1,656.72	3,313.44
I/S	45.20	90.40	180.80	361.60	723.20
M/S	2.79	5.58	11.16	22.32	44.64
C/M	13.74	27.48	54.96	109.92	219.84
E/M	1,015.75	2,031.50	4,063.00	8,126.00	16,252.00
I/M	46.90	93.80	187.60	375.20	750.40
M/M	4.94	9.88	19.76	39.52	79.04
C/L	13.20	26.40	52.80	105.60	211.20
E/L	617.39	1,234.78	2,469.56	4,939.12	9,878.24
I/L	47.58	95.16	190.32	380.64	761.28
M/L	31.29	62.58	125.16	250.32	500.64

TABLE 16. Environment setup for training each agent.

Agent Name	Workflow/ Number of Tasks	Deadline (second)
PHH-C1X	CyberShake/100	1x (13.74)
PHH-E1X	Epigenomics/100	1x (1,015.75)
PHH-I1X	Inspiral/100	1x (46.90)
PHH-M1X	Montage/100	1x (4.94)
PHH-C16X	CyberShake/100	16x (219.84)
PHH-E16X	Epigenomics/100	16x (16,252.00)
PHH-I16X	Inspiral/100	16x (750.40)
PHH-M16X	Montage/100	16x (79.04)

workflows and deadlines. To determine if the trained agents can produce results of comparable quality to state-of-the-art algorithms, they were compared to three meta-heuristics algorithms: GA-Based DCOH [51], Best-so-far ABC [52], and NeSS [53]; and one hyper-heuristic-based algorithm CTDHH [16].

Zhou et al. [51] proposed the GA-based DCOH (deadline-constrained cost optimization for hybrid clouds) algorithm to schedule a workflow application in the hybrid cloud to minimize monetary costs subject to a deadline constraint. The performance of DCOH was reported to reduce the monetary cost by up to 100% compared to competing algorithms.

The Best-so-far ABC (BSFABC) algorithm [52] improves the performance of the original ABC [54] by introducing a modified method to update the solution in the onlooker bee phase, which creates a bias towards the best-so-far solution. It has been applied to a job shop scheduling problem, which is similar to workflow scheduling.

The NeSS algorithm [53] is a new version of the original ABC based on the natural nest site selection behavior of honeybee swarms. The algorithm demonstrated promising performance in solving combinatorial problems.

Hyper-heuristic based CTDHH (Completion Time Driven Hyper-Heuristic) algorithm [16], proposed by Alkhanak and Lee, is a hyper-heuristic-based approach that uses four perturbative metaheuristics (GA [55], PSO [56], IWO [57], and HIWO [58]) as low-level heuristics to solve cost optimization for workflow scheduling problems in cloud computing. The

algorithm repeatedly selects and applies the selected LLH to perturb the solution until the stopping criterion is met. In each iteration, the LLHs are selected based on the performance from previous iterations.

d: PARAMETER SETTINGS

Table 17 summarizes the parameter settings for all algorithms, which were determined through empirical tuning. Table 18 provides a summary of all factors and levels used in the experiment. For each run, the cost and makespan of the resulting schedules were recorded. The proposed PHH-based algorithm was implemented in Python 3.10 with the Stable-Baseline3 Reinforcement Learning Library. This library includes the PPO algorithm and provides the capability to implement a custom environment.

4) RESULTS AND DISCUSSION

After conducting the above-described experiment, the collected data was analyzed. This section discusses the performance of the algorithms to determine whether the PHH can significantly outperform the meta-heuristic algorithms in various conditions. Friedman's test and Nemenyi's post-hoc test were used to verify the significance of the differences and identify the performance ranks. The statistical testing was carried out using the Python package Scipy.Stats and Scikit-Posthocs.

Table 19 shows the average costs from each scenario. Friedman's test was performed to evaluate the statistical significance of the differences in the non-parametric data. The p-value from Friedman's test is $1.1E-14$, indicating that some of the differences in the mean costs are statistically significant. It should be noted that N/A values were replaced with $1E+10$ for significant testing.

Nemenyi's pairwise post-hoc test was then conducted on the costs from all scenarios to identify which pairs of algorithms produced statistically significant costs. Figure 9 shows the plot of p-values for each pair of algorithm settings. Red cells indicate that the difference in costs for the corresponding pair is not significant, while green cells indicate that the difference is significant. The darker the green, the higher the significance.

The pairwise significant plot shows that the costs from most pairs are significantly different. There are two groups of algorithms with insignificant differences within the group, but these algorithms still provided significantly different costs compared to other algorithms outside their corresponding groups. The first group includes DCOH, PHH-C16X, and PHH-M16X. The second group includes PHH-C1X, PHH-E1X, PHH-E16X, PHH-I1X, PHH-I16X, and PHH-M1X. The fact that the first group contains both the PHH agents and the DCOH algorithm supports the hypothesis that, with the appropriate configurations, the proposed PHH algorithm (PHH-C16X, PHH-M16X) can produce solutions of equivalent quality to the meta-heuristic algorithm (DCOH).

According to the average costs shown in Table 19, the BSFABC algorithm produced the lowest costs in most

TABLE 17. Algorithm parameter settings.

Algorithm	Parameter	Value
Policy-Based Hyper-Heuristic (PHH)	Reinforcement learning algorithm	PPO
	Learning rate	0.01x ; x is the training progress (0.0 - 1.0)
	Number of training epochs	5
	Clip range	0.2
	Value coefficient	0.5
	Number of steps to collect in the experience before updating the policy	10,000
	Discount factor	0.99
	The activation function in the policy network and value network	ReLU
	Hidden layers of the policy network	Dense layers with 128, 64, 32 nodes
	Hidden layers of the value network	Dense layers with 64, 32 nodes
	Total training timestep	20,000,000
	Task moving window width (μ_t)	50
	VM moving window width (μ_{vm})	5
]GA-Based DCOH	Population size	500
	Crossover probability	0.6
	Mutation probability	0.25
	Stopping criterion	- Minimum 100 iterations - Stop after no improvement for 10 consecutive iterations
Best-so-far ABC (BSFABC)	Population size	200
	w_{min}, w_{max}	0.2, 1.0
	Stopping criterion	Maximum 100 iterations
NeSS	Population size	100
	Fitness threshold (E_t)	0.2
	Probability of explorer bees (P_m)	0.25
	Initial dance strength	150
	Dance strength decay rate	15
	Stopping criterion	Maximum 300 iterations
CTDHH	Low-level heuristics (LLH)	GA, PSO, IWO, HIWO
	Stopping criterion	100 iterations
	LLH iterations	2
	LLH population size	200
	GA crossover rate	0.3
	GA mutation rate	0.018
	PSO inertia weight	0.8
	PSO acceleration weight	1.0
	IWO maximum population	500
	HIWO maximum population	500
	HIWO mutation rate	0.018

scenarios for small and medium workflows, while the agents from the proposed PHH algorithm provided the lowest costs in most scenarios for large workflows and the tight deadline (2x) in small and medium workflows.

For the sake of simplicity, further analysis was limited to significantly different algorithm settings using p-value from the Nemenyi pairwise post-hoc test, as shown in Figure 9. The DCOH and PHH-C1X were selected from the first and second groups of insignificances, respectively. This means that only the CTDHH, PHH-C1X, BSFABC, NeSS, and DCOH were included in the further analysis.

Table 20 shows the average cost of schedules obtained from each algorithm setting for each workflow application, workflow size, and deadline. Algorithm settings that were not able to obtain a feasible schedule, with a makespan within the deadline, were marked as N/A. From the table, the DCOH, NeSS, and PHH were able to produce feasible schedules in most scenarios.

The BSFABC algorithm produced the lowest costs for most scenarios for small workflows (size S) and some scenarios for medium workflows (size M). However, it was unable to produce a feasible solution when the deadline is very tight

TABLE 18. Summary of experiment factors and levels.

Factor	Levels
Workflow	- CyberShake (C) - Epigenomics (E) - Inspiral (I) - Montage (M)
Size	- Small (~30 tasks) - Medium (100 tasks) - Large (~1,000 tasks)
Deadline	- 2x - 4x - 8x
Algorithm/Agent	- PHH-C1X - PHH-E1X - PHH-I1X - PHH-M1X - PHH-C16X - PHH-E16X - PHH-I16X - PHH-M16X - BNSS - BSFABC - DCOH - CTDHH

(2x) or when the workflow is larger (size L). The CTDHH algorithm also had a lower chance of producing feasible schedules for larger workflows.

The PHH-C1X was able to obtain the best cost for most scenarios with the tight deadline (2x) and most scenarios for the large workflow (size L).

Table 20 compares the overall algorithm performance comparison by average cost ranks for each experimental scenario. A lower rank indicates a lower cost. The average ranks of the PHH algorithm are the lowest compared to other algorithms. This means the costs produced by the PHH algorithms are the lowest in most experiments. Table 22 shows the average performance ranks of the algorithms specifically for large workflows. The table shows that the average rank for the PHH algorithm is still the lowest and lower than its rank in the previous table. This indicates that the PHH algorithm outperformed other algorithms for large workflows. Although the pairwise comparison between PHH-C1X and DCOH for large workflows is not significant (p -value = 0.089), PHH-C1X still outperforms DCOH with lower average ranks for small and medium workflows in most scenarios.

The metaheuristic-based algorithms (BSFABC, DCOH, NeSS, and CTDHH) use complex exploration and exploitation behaviors to search through the search space, which can help them avoid local optima and find global ones. These behaviors allow them to achieve the lowest cost in Table 20 for most scenarios with small and medium workflows. These algorithms use a soft constraint, in the form of a penalty added to the fitness function, to avoid infeasible solutions. This penalty guides the algorithm to search in a direction that reduces the penalty and eventually enters the feasible area of the search space. However, this becomes more difficult and time-consuming when the search space is large, as seen

in Table 20 where these algorithms are unable to produce feasible solutions or perform relatively poorly with large workflows or tight deadlines compared to PHH.

The hyper-heuristic-based CTDHH algorithm with metaheuristics as LLHs inherits pros and cons from the metaheuristic-based algorithms. It only selects LLHs based on workflow size and number of VMs, without considering workflow structures and VM attributes. In this experiment, the same LLH was selected for the different workflows with the same size. For example, the medium workflows (100 tasks) of CyberShake, Epigenomics, Inspiral, and Montage were scheduled with the same LLH algorithm, which did not take advantage of the hyper-heuristic method. This limits the CTDHH algorithm to only being suitable for environments that regularly execute the same workflow applications with the same structure but different sizes.

The proposed PHH algorithm consumes time during offline training for the different classes of problem instances but uses less time to generate a schedule compared to the competing algorithms. Since the LLHs of the proposed algorithm are constructive heuristics, it only takes one episode to generate a complete schedule. This allows a relatively shorter time to generate a schedule for a larger workflow.

Unlike metaheuristic-based algorithms, the reinforcement learning-based PHH algorithm does not explore and exploit the search space during the run time after training. It only takes actions based on its experiences with the observed state. Table 20 shows that, for small workflows, the PHH algorithm produced costs that are equal to or worse than other algorithms. Additionally, the PHH algorithm takes actions based on state information in the moving window, meaning that it only perceives a specific amount of state information at a time. However, the moving window method for state information does not compromise solution quality for the proposed method when the workflow is larger, as seen in Table 20 where the PHH algorithm produces lower costs for most scenarios with large workflows compared to the other algorithms.

The PHH algorithm was not able to perform optimally on the Epigenomics workflow possibly due to its unique structure as depicted in Figure 8. The workflow consists of many parallelized long sequential tasks, and the limited information in the task moving window may have hindered the PHH agents' ability to recognize the differences in environment stages as the window moved along. This may have resulted in suboptimal scheduling decisions, leading to higher costs compared to other algorithms.

One of the benefits of using reinforcement learning-based algorithms is the ability to train multiple agents on different problem instances. This can increase the chances of obtaining better solutions, as the multiple agents can work together to solve the problem. The use of constructive low-level heuristics (LLHs) in the reinforcement learning algorithm also allows the agent to quickly find a solution within a single episode, without the need for complex exploration and exploitation behaviors as in traditional metaheuristic

TABLE 19. Average cost (USD) of schedules obtained from each workflow and deadline. The lowest costs are marked in bold. N/A means the algorithm cannot obtain a feasible schedule where its makespan is within the deadline.

Workflow /Size	Deadline (seconds)	BSFABC	CTDHH	DCOH	NeSS	PHH-C16X	PHH-C1X	PHH-E16X	PHH-E1X	PHH-I16X	PHH-I1X	PHH-M16X	PHH-M1X
C/S	27.62 (2x)	0.15	N/A	0.19	0.25	0.21	0.19	0.18	0.20	0.18	0.19	0.19	0.18
C/S	55.24 (4x)	0.06	0.06	0.11	0.17	0.10	0.09	0.09	0.11	0.11	0.10	0.12	0.06
C/S	110.48 (8x)	0.00	N/A	0.01	0.11	N/A	0.02	0.05	0.05	0.03	0.04	0.17	0.00
E/S	414.18 (2x)	5.26	5.27	5.26	5.32	5.25	5.26	5.26	5.26	5.26	5.26	5.29	5.26
E/S	828.36 (4x)	4.17	3.40	3.23	5.32	4.84	3.53	3.90	3.34	3.69	3.97	4.34	3.25
E/S	1656.72 (8x)	0.01	N/A	0.20	3.87	1.14	0.78	1.42	1.66	1.43	1.67	3.49	0.66
I/S	90.40 (2x)	N/A	N/A	1.87	1.99	1.94	1.87	1.91	1.91	1.95	1.91	1.88	1.86
I/S	180.80 (4x)	1.25	N/A	1.48	1.99	1.77	1.30	1.41	1.42	1.47	1.41	1.65	1.32
I/S	361.60 (8x)	0.21	0.44	0.86	1.58	1.03	1.22	1.08	1.07	0.91	1.04	0.95	0.98
M/S	5.58 (2x)	N/A	N/A	0.06	0.07	0.07	0.06	0.06	0.06	0.06	0.06	0.06	0.06
M/S	11.16 (4x)	0.03	0.04	0.04	0.07	0.05	0.04	0.03	0.04	0.04	0.03	0.04	0.03
M/S	22.32 (8x)	0.00	N/A	0.02	0.03	0.02	0.02	0.02	0.02	0.01	0.02	0.02	0.02
C/M	27.48 (2x)	N/A	N/A	0.67	0.73	0.62	0.62	0.62	0.64	0.65	0.63	0.67	0.62
C/M	54.96 (4x)	0.35	0.56	0.55	0.73	N/A	0.40	0.46	0.40	0.46	0.44	0.49	0.46
C/M	109.92 (8x)	0.05	N/A	0.39	0.46	N/A	0.31	0.20	0.22	0.14	0.27	0.16	0.22
E/M	2031.50 (2x)	N/A	N/A	121.00	121.02	120.99	120.99	120.99	120.99	121.00	121.00	120.99	120.99
E/M	4063.00 (4x)	103.65	N/A	108.18	121.02	114.29	107.19	108.19	109.84	107.47	109.43	112.48	107.08
E/M	8126.00 (8x)	56.09	N/A	73.75	121.02	60.81	77.16	75.23	84.75	83.77	84.26	94.80	66.30
I/M	93.80 (2x)	N/A	N/A	6.50	6.65	6.40	6.44	6.46	6.43	6.47	6.45	6.49	6.39
I/M	187.60 (4x)	N/A	N/A	6.08	6.65	5.88	5.65	5.63	5.58	5.68	5.64	5.63	5.69
I/M	375.20 (8x)	3.68	N/A	5.17	6.65	5.96	4.61	4.58	4.40	4.65	4.61	4.24	4.62
M/M	9.88 (2x)	N/A	N/A	0.29	0.32	0.30	0.28	0.28	0.28	0.28	0.28	0.28	0.29
M/M	19.76 (4x)	N/A	0.28	0.24	0.32	0.22	0.22	0.22	0.22	0.21	0.22	0.22	0.23
M/M	39.52 (8x)	0.14	0.11	0.15	0.32	0.20	0.14	0.12	0.12	0.11	0.11	0.07	0.17
C/L	26.40 (2x)	N/A	N/A	6.69	6.84	6.68	6.66	6.68	6.68	6.66	6.67	6.69	6.71
C/L	52.80 (4x)	N/A	N/A	6.48	6.84	N/A	6.45	6.51	6.49	6.47	6.48	6.48	6.54
C/L	105.60 (8x)	N/A	N/A	6.12	6.84	6.35	6.24	6.12	6.12	6.07	6.10	6.24	6.61
E/L	1234.78 (2x)	N/A	N/A	566.47	570.02	569.31	569.30	569.53	569.52	569.45	569.51	569.30	569.39
E/L	2469.56 (4x)	N/A	N/A	553.20	570.02	560.22	554.94	553.64	553.75	553.59	553.51	562.60	561.10
E/L	4939.12 (8x)	N/A	N/A	529.24	570.02	540.31	544.85	542.32	545.38	542.22	547.44	558.21	549.71
I/L	95.16 (2x)	N/A	N/A	67.30	67.74	67.08	66.99	67.02	67.02	67.01	67.02	N/A	N/A
I/L	190.32 (4x)	N/A	N/A	66.60	67.74	66.40	66.13	66.19	66.17	66.16	66.18	66.38	66.53
I/L	380.64 (8x)	N/A	N/A	65.17	67.74	65.48	64.43	64.48	64.50	64.46	64.49	64.98	65.34
M/L	62.58 (2x)	N/A	N/A	3.23	3.43	3.43	3.04	3.07	3.06	3.06	3.06	3.06	3.05
M/L	125.16 (4x)	N/A	N/A	3.09	3.43	2.86	2.67	2.73	2.71	2.66	2.74	2.66	2.67
M/L	250.32 (8x)	N/A	N/A	1.79	3.43	2.72	2.35	1.61	1.66	1.55	1.64	1.72	2.52

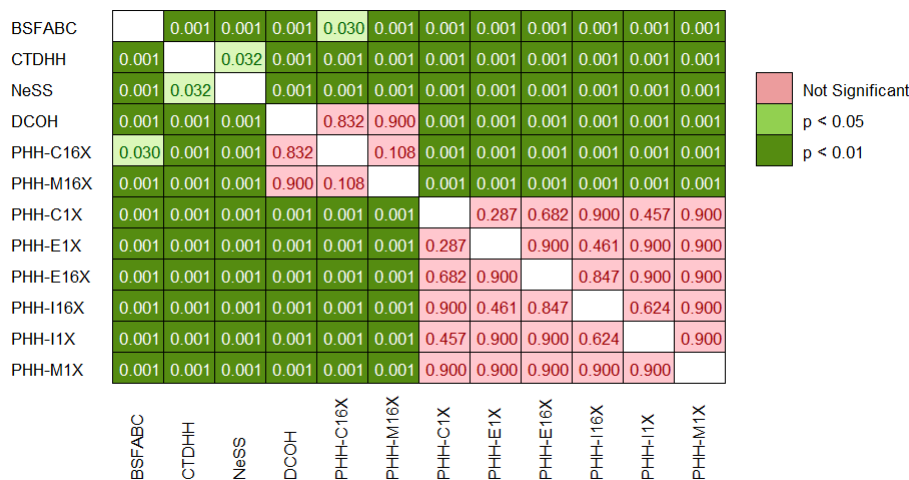


FIGURE 9. Pairwise P-values calculated from experiment results to indicate whether the difference in solution quality (cost) from each pair of algorithms is statistically significant or not among all workflows. E.g., costs from PHH-C1X and DCOH are significantly different (p-value = 0.001), but costs from DCOH and PHH-C16X are not (p-value = 0.832).

algorithms. This can be especially useful in scenarios where time is a factor, such as when scheduling large workflow applications with various deadlines, including unforeseen

ones. Additionally, the trained agents can be improved over time through re-training with new, frequent classes of workflow instances. This flexibility and adaptability make the

TABLE 20. Comparison of average costs (USD) of schedules obtained from the selected algorithms for each workflow application, workflow size, and deadline. The lowest costs among the selected algorithms are marked in bold. N/A means the algorithm cannot obtain the feasible schedule, i.e., its makespan exceeded the deadline.

Workflow /Size	Deadline (seconds)	BSFABC	DCOH	NeSS	CTDHH	PHH-C1X
C/S	27.62 (2x)	0.15	0.19	0.25	N/A	0.19
C/S	55.24 (4x)	0.06	0.11	0.17	0.06	0.09
C/S	110.48 (8x)	0.00	0.01	0.11	N/A	0.02
E/S	414.18 (2x)	5.26	5.26	5.32	5.27	5.26
E/S	828.36 (4x)	4.17	3.23	5.32	3.40	3.53
E/S	1656.72 (8x)	0.01	0.20	3.87	N/A	0.78
I/S	90.40 (2x)	N/A	1.87	1.99	N/A	1.87
I/S	180.80 (4x)	1.25	1.48	1.99	N/A	1.30
I/S	361.60 (8x)	0.21	0.86	1.58	0.44	1.22
M/S	5.58 (2x)	N/A	0.06	0.07	N/A	0.06
M/S	11.16 (4x)	0.03	0.04	0.07	0.04	0.04
M/S	22.32 (8x)	0.00	0.02	0.03	N/A	0.02
C/M	27.48 (2x)	N/A	0.67	0.73	N/A	0.62
C/M	54.96 (4x)	0.35	0.55	0.73	0.56	0.40
C/M	109.92 (8x)	0.05	0.39	0.46	N/A	0.31
E/M	2031.50 (2x)	N/A	121.00	121.02	N/A	120.99
E/M	4063.00 (4x)	103.65	108.18	121.02	N/A	107.19
E/M	8126.00 (8x)	56.09	73.75	121.02	N/A	77.16
I/M	93.80 (2x)	N/A	6.50	6.65	N/A	6.44
I/M	187.60 (4x)	N/A	6.08	6.65	N/A	5.65
I/M	375.20 (8x)	3.68	5.17	6.65	N/A	4.61
M/M	9.88 (2x)	N/A	0.29	0.32	N/A	0.28
M/M	19.76 (4x)	N/A	0.24	0.32	0.28	0.22
M/M	39.52 (8x)	0.14	0.15	0.32	0.11	0.14
C/L	26.40 (2x)	N/A	6.69	6.84	N/A	6.66
C/L	52.80 (4x)	N/A	6.48	6.84	N/A	6.45
C/L	105.60 (8x)	N/A	6.12	6.84	N/A	6.24
E/L	1234.78 (2x)	N/A	566.47	570.02	N/A	569.30
E/L	2469.56 (4x)	N/A	553.20	570.02	N/A	554.94
E/L	4939.12 (8x)	N/A	529.24	570.02	N/A	544.85
I/L	95.16 (2x)	N/A	67.30	67.74	N/A	66.99
I/L	190.32 (4x)	N/A	66.60	67.74	N/A	66.13
I/L	380.64 (8x)	N/A	65.17	67.74	N/A	64.43
M/L	62.58 (2x)	N/A	3.23	3.43	N/A	3.04
M/L	125.16 (4x)	N/A	3.09	3.43	N/A	2.67
M/L	250.32 (8x)	N/A	1.79	3.43	N/A	2.35

TABLE 21. Average performance rank of algorithms for all scenarios.

Algorithm	Avg Rank	The P-Value of pairwise comparison with PHH-C1X
PHH-C1X	1.81	-
DCOH	2.19	0.001
BSFABC	3.12	0.001
NeSS	3.67	0.001
CTDHH	4.21	0.001

proposed PHH algorithm well-suited for solving workflow scheduling problems in dynamic environments.

This study demonstrated the ability of the proposed PHH algorithm to optimize the cost of scheduling workflows in a hybrid cloud environment. The results showed that the PHH algorithm was able to learn how to schedule workflows to minimize costs and was able to provide solutions

TABLE 22. Average performance rank of algorithms for only large workflows.

Algorithm	Avg Rank	The P-Value of pairwise comparison with PHH-C1X
PHH-C1X	1.42	-
DCOH	1.58	0.089
NeSS	3.00	0.001
BSFABC	4.50	0.001
CTDHH	4.50	0.001

to unforeseen problem instances with equivalent or better quality compared to meta-heuristic algorithms. In particular, the PHH algorithm performed well for large workflow applications and tight deadlines, outperforming the other algorithms in terms of cost optimization. Additionally, the PHH algorithm had the advantage of being trained on various

problem instances, allowing it to adapt to different workflow sizes and deadlines, and it was able to generate schedules quickly, making it an attractive solution for organizations that frequently execute large workflow applications with various deadlines. Overall, the results of this study suggest that the PHH algorithm is a promising approach for optimizing the cost of scheduling workflows in a hybrid cloud environment.

V. CONCLUSION

Hyper-heuristics are an alternative approach to addressing the problem of finding the appropriate heuristic to solve a given task. They aim to increase the generality of search algorithms across different problems. However, this increased generality may result in lower performance compared to meta-heuristics. Reinforcement learning offers a potential solution to this issue by allowing algorithms to learn from previously solved problem instances and apply that knowledge to new ones. The use of reinforcement learning in hyper-heuristics is a promising approach for tackling complex optimization problems that are difficult to solve with traditional methods.

This research proposed a framework for policy-based hyper-heuristics with reinforcement learning (PHH) that can be used to effectively solve optimization problems and address the issue of generality in search algorithms. The low-level heuristics (LLH) used in the proposed framework are constructive, meaning that the framework can quickly find a solution compared to using perturbative LLHs. The reinforcement learning component of the proposed framework uses a policy-based approach, which directly learns a policy function that maps states to actions without the need for a separate value function. The Proximal Policy Optimization (PPO) algorithm was used to update the policy function in a way that keeps the new policy close to the old one. This stabilizes the learning process and improves sample efficiency.

The effectiveness of the proposed PHH framework cannot be evaluated using HyFlex because the constructive LLHs are not supported. We then implemented three HyFlex's problem domains for comparisons: traveling salesman, capacitated vehicle routing, and bin packing problems. The experiments highlighted the potential of our framework to generalize across different problem domains, making it a promising tool for solving a wide range of optimization problems. The proposed framework outperformed Simulated Annealing, Genetic Algorithm, and online-training DQN-based hyper-heuristic for all large problem instances, while only training with smaller instances. The proposed PHH provided up to 98% better solution quality, indicating its potential to address complex optimization challenges.

The proposed PHH framework was also tested on a workflow scheduling optimization task on a hybrid cloud with a deadline constraint. Four medium-sized workflows with 1x and 16x deadlines were used to train eight agents. The trained agents were then used to schedule unforeseen workflows and deadlines to evaluate their performance in solving new problem instances. Four real-world workflow applications of three sizes and five deadlines were used in the experiment.

The performance of the trained agents was compared to three other types of algorithms: hyper-heuristics with meta-heuristic perturbative LLHs (CTDHH), a numerical optimization algorithm (Best-so-far ABC), and combinatorial optimization algorithms (DCOH and NeSS).

The trained agents from the proposed PHH framework were run to schedule all workflows independently, and the agent with the best result and statistical significance was selected. The selected agent outperforms state-of-the-art algorithms in 7 out of 12 scenarios with large workflow applications with 1,000 tasks, 6 out of 12 scenarios with medium workflow applications with 100 tasks, and 3 out of 12 scenarios with small workflow applications with 30 tasks. In the experiment with the tightest deadline, the trained PHH agent outperformed state-of-the-art algorithms in 10 out of 12 scenarios with all workflow applications and all sizes. The average performance rank of the trained PHH agent is 1.81 out of 5 for all scenarios, and 1.42 for scenarios with large workflow applications. Overall, the trained agents performed better than the other algorithms in most scenarios for large workflows and in most scenarios with tight deadlines.

The results of this study provide strong support for the use of the PHH framework in solving combinatorial problems for large problem instances when there are sufficient time and training problem instances available before the actual problem instance is encountered.

REFERENCES

- [1] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: A survey," *J. Supercomputing*, vol. 71, no. 9, pp. 3373–3418, 2015.
- [2] S. Muthuraman and V. P. Venkatesan, "A comprehensive study on hybrid meta-heuristic approaches used for solving combinatorial optimization problems," in *Proc. World Congr. Comput. Commun. Technol. (WCCCT)*, Feb. 2017, pp. 185–190.
- [3] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, Mar. 2011.
- [4] S. Voß, "Meta-heuristics: The state of the art," *Proc. Workshop Local Search Planning Scheduling*, in *Lecture Notes in Computer Science*, vol. 2148, 2001, pp. 1–23.
- [5] E. G. Talbi, *Metaheuristics: From Design to Implementation*. Hoboken, NJ, USA: Wiley, 2009.
- [6] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, Dec. 2013.
- [7] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Comput. Operations Res.*, vol. 134, Oct. 2021, Art. no. 105400.
- [8] P. Ross, "Hyper-heuristics," in *Search Methodologies*. Cham, Switzerland: Springer, 2014, pp. 611–638.
- [9] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic, and E. K. Burke, "HyFlex: A benchmark framework for cross-domain heuristic search," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7245. Berlin, Germany: Springer, 2012, pp. 136–147.
- [10] P. Cowling and E. Soubeiga, "Neighborhood structures for personnel scheduling: A summit meeting scheduling problem (abstract)," in *Proc. 3rd Int. Conf. Pract. Theory Automated Timetabling*, Constance, Germany, E. K. Burke and W. Erben, Eds. Aug. 2000, pp. 16–18.
- [11] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2079. 2001, pp. 176–190.

- [12] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," *Intell. Syst. Ref. Libr.*, vol. 1, no. 1, pp. 177–201, 2009.
- [13] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches: Revisited," in *International Series in Operations Research and Management Science*, vol. 272. Cham, Switzerland: Springer, 2019, pp. 453–477.
- [14] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *Eur. J. Oper. Res.*, vol. 285, no. 2, pp. 405–428, Sep. 2020.
- [15] S. S. Choong, L.-P. Wong, and C. P. Lim, "An artificial bee colony algorithm with a modified choice function for the traveling salesman problem," *Swarm Evol. Comput.*, vol. 44, pp. 622–635, Feb. 2019.
- [16] E. N. Alkhanak and S. P. Lee, "A hyper-heuristic cost optimisation approach for scientific workflow scheduling in cloud computing," *Future Gener. Comput. Syst.*, vol. 86, pp. 480–506, Sep. 2018.
- [17] J. Lin, L. Zhu, and K. Gao, "A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem," *Exp. Syst. Appl.*, vol. 140, Feb. 2020, Art. no. 112915.
- [18] A. Rasouli Kenari and M. Shamsi, "A hyper-heuristic selector algorithm for cloud computing scheduling based on workflow features," *OPSEARCH*, vol. 58, no. 4, pp. 852–868, Dec. 2021.
- [19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
- [20] M. Naem, S. T. H. Rizvi, and A. Coronato, "A gentle introduction to reinforcement learning and its application in different fields," *IEEE Access*, vol. 8, pp. 209320–209344, 2020.
- [21] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 50–56.
- [22] J. Park, J. Chun, S. H. Kim, Y. Kim, and J. Park, "Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning," *Int. J. Prod. Res.*, vol. 59, no. 11, pp. 3360–3377, Jun. 2021.
- [23] M. Sharafath Abdul Hameed and A. Schwung, "Graph neural networks-based scheduler for production planning problems using reinforcement learning," 2020, *arXiv:2009.03836*.
- [24] M. Melnik and D. Nasonov, "Workflow scheduling using neural networks and reinforcement learning," *Proc. Comput. Sci.*, vol. 156, pp. 29–36, Jan. 2019.
- [25] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
- [26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.
- [27] X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao, "Deep reinforcement learning: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Sep. 28, 2022, doi: 10.1109/TNNLS.2022.3207346.
- [28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
- [29] D. Falcão, A. Madureira, and I. Pereira, "Q-learning based hyper-heuristic for scheduling system self-parameterization," in *Proc. 10th Iberian Conf. Inf. Syst. Technol. (CISTI)*, Aveiro, Portugal, Jun. 2015, pp. 1–7.
- [30] A. Madureira, I. Pereira, P. Pereira, and A. Abraham, "Negotiation mechanism for self-organized scheduling system with collective intelligence," *Neurocomputing*, vol. 132, pp. 97–110, May 2014.
- [31] J. Lin, Y.-Y. Li, and H.-B. Song, "Semiconductor final testing scheduling using Q-learning based hyper-heuristic," *Exp. Syst. Appl.*, vol. 187, Jan. 2022, Art. no. 115978.
- [32] İ. Gölük and F. B. Ozsoydan, "Q-learning and hyper-heuristic based algorithm recommendation for changing environments," *Eng. Appl. Artif. Intell.*, vol. 102, Jun. 2021, Art. no. 104284.
- [33] A. Dantas, A. F. D. Rego, and A. Pozo, "Using deep Q-network for selection hyper-heuristics," in *Proc. Genetic Evol. Comput. Conf. Companion*. New York, NY, USA: Association for Computing Machinery, Jul. 2021, pp. 1488–1492.
- [34] W. Qin, Z. Zhuang, Z. Huang, and H. Huang, "A novel reinforcement learning-based hyper-heuristic for heterogeneous vehicle routing problem," *Comput. Ind. Eng.*, vol. 156, Jun. 2021, Art. no. 107252.
- [35] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.
- [36] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1995–2003.
- [37] M. Sewak, "Introduction to reinforcement learning," in *Deep Reinforcement Learning*. Singapore: Springer, 2019.
- [38] M. Sewak, "Deep Q network (DQN), double DQN, and dueling DQN," in *Deep Reinforcement Learning*. Singapore: Springer, 2019.
- [39] R. Magalhães, M. Martins, S. Vieira, F. Santos, and J. Sousa, "Encoder-decoder neural network architecture for solving job shop scheduling problems using reinforcement learning," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2021, pp. 1–8.
- [40] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, Nov. 1991.
- [41] M. Delorme, M. Iori, and S. Martello, "BPPLIB: A library for bin packing and cutting stock problems," *Optim. Lett.*, vol. 12, no. 2, pp. 235–250, Mar. 2018.
- [42] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, 2021.
- [43] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *J. Amer. Stat. Assoc.*, vol. 32, no. 200, pp. 675–701, Dec. 1937.
- [44] P. B. Nemenyi, "Distribution-free multiple comparisons," Ph.D. thesis, Dept. Math., Princeton Univ., Princeton, NJ, USA, 1963.
- [45] P. Virtanen, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, Feb. 2020.
- [46] M. Terpilowski, "Scikit-posthocs: Pairwise multiple comparison tests in Python," *J. Open Source Softw.*, vol. 4, no. 36, p. 1169, Apr. 2019.
- [47] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues," *J. Syst. Softw.*, vol. 113, pp. 1–26, Mar. 2016.
- [48] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, 2002.
- [49] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, Mar. 2013.
- [50] *Workflow Generator*. Accessed: Nov. 30, 2022. [Online]. Available: <https://confluence.pegasus.isi.edu/display/pegasus/Deprecated+Workflow+Generator>
- [51] J. Zhou, T. Wang, P. Cong, P. Lu, T. Wei, and M. Chen, "Cost and makespan-aware workflow scheduling in hybrid clouds," *J. Syst. Archit.*, vol. 100, Nov. 2019, Art. no. 101631.
- [52] A. Banharsakun, T. Achalakul, and B. Sirinaovakul, "The best-so-far selection in artificial bee colony algorithm," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 2888–2901, Mar. 2011.
- [53] U. Taetragool, B. Sirinaovakul, and T. Achalakul, "NeSS: A modified artificial bee colony approach based on nest site selection behavior," *Appl. Soft Comput.*, vol. 71, pp. 659–671, 2018.
- [54] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Global Optim.*, vol. 39, no. 3, pp. 459–471, Oct. 2007.
- [55] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Sci. Program.*, vol. 14, nos. 3–4, pp. 217–230, 2006.
- [56] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Jun. 2010, pp. 400–407.
- [57] K. Li, S. Li, Y. Xu, and Z. Xie, "A DAG task scheduling scheme on heterogeneous computing systems using invasive weed optimization algorithm," in *Proc. 6th Int. Symp. Parallel Architectures, Algorithms Program.*, Jul. 2014, pp. 262–267.
- [58] R. Sharma, N. Nayak, K. R. Krishnanand, and P. K. Rout, "Modified invasive weed optimization with dual mutation technique for dynamic economic dispatch," in *Proc. Int. Conf. Energy, Autom. Signal*, Dec. 2011, pp. 660–665.



ORACHUN UDOMKASEMSUB received the bachelor's degree from the King Mongkut's University of Technology Thonburi, Thailand, in 2011, where he is currently pursuing the Ph.D. degree in computer engineering. His research interests include cloud computing, optimization, swarm intelligence, and deep learning.



BOONCHAROEN SIRINAOVAKUL received the B.Eng. degree from the King Mongkut's Institute of Technology Ladkrabang, Thailand, the M.S. degree from Wichita State University, USA, and the D.Eng. degree from the King Mongkut's Institute of Technology Ladkrabang. He was with the public and private sectors on various optimization and operations research problems. He is currently a Professor with the Department of Computer Engineering, Faculty of Engineering,

King Mongkut's University of Technology Thonburi, Thailand. His research interests include optimization, swarm intelligence, and deep learning.



TIRANEE ACHALAKUL is currently the Director of the Government Big Data Institute (GBDi), the data intelligence service provider under the Ministry of Digital Economy and Society of Thailand. She has been working in the fields of big data analytics, high-performance computing, cloud and virtualization, and software engineering, since 2000. She has extended experiences working in the IT industry and academia in both the USA and Thailand. During the past 20 years, she has been

both an educator and a consultant. She serves on advisory boards for multiple agencies as well as teaches university students with KMUTT. She is also the Founder of the Big Data Experience Center, the KMUTT Student Incubator (Hatch), and a few technological startups.

• • •