Office of Graduate Programs
School of Engineering and Applied Science
Thornton Hall A108
20200715

Thesis/Dissertation Cover and Approval Pages

**** Complete (type) the following Form Variables and then click UPDATE***

Form Variables

Document Title	Reinforcement Learning Based Scheduling for the Next-Generation Astronomy Observation
Student Name	John Shen
Document Type	Thesis
Degree	Master of Science
Semester/Year Degree Awarded	December 2024
Advisor 1 Name	Haiying Shen
Advisor 2 Name	
Committee Member 1 Name	Shangtong Zhang
Committee Member 2 Name	Yen-Ling Kuo
Committee Member 3 Name	
Committee Member 4 Name	
Committee Member 5 Name	
Committee Member 6 Name	

UPDATE

Reinforcement Learning Based Scheduling for the Next-Generation Astronomy Observation

---

A

Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Master of Science

by

John Shen

December 2024

APPROVAL SHEET

This

Thesis

is submitted in partial fulfillment of the requirements
for the degree of

Master of Science

Author: John Shen

This Thesis has been read and approved by the examing committee:

Advisor: Haiying Shen

Advisor:

Committee Member: Shangtong Zhang

Committee Member: Yen-Ling Kuo

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Jennifer L. West, School of Engineering and Applied Science

December 2024

Contents

Reinforcement Learning Based Scheduling for the Next-Generation Astronomy Observation

December, 2024

Abstract

The next-generation Very Large Array (ngVLA), as the next-generation radio/mm telescope operated by the National Radio Astronomy Observatory (NRAO), has ten times the sensitivity and spatial resolution of the previous generations of telescopes and can operate at frequencies spanning up to 116 GHz, which can provide large scale astrophysical imaging for new discovery. There are five ngVLA key science goals: i) unveiling the formation of solar system analogues, ii) probing the initial conditions for planetary systems and life with astrochemistry, iii) charting the assembly, structure, and evolution of galaxies from the first billion years to the present, iv) using pulsar in the galactic center as fundamental tests of gravity, v) understanding the formation and evolution of stellar and supermassive black holes in the era of multi-messenger astronomy.

To achieve the scientific goals, the next step is to extend the scheduling algorithm from the previous generation telescope to perform optimal or near-optimal scheduling solutions for ngVLA. Given antennas, subarrays were created that can be sufficient to cover all high-priority observation requests, then given the selected Scheduling Blocks (SBs) as observation task instances with multiple attributes including frequency, local standard time (LST), Phase Root Mean Square (RMS), weather-related attributes, angular resolution, largest angular scale, etc, the scheduling system arranges them with optimal or near-optimal performance to execute in the given observation time window. The system may first find eligible SBs and corresponding SAs according to the current and forecast weather conditions, then use a scheduling algorithm to generate subarrays eligible for selected SBs and a schedule that considers priorities. Previous scheduling approaches are not sufficiently efficient for ngVLA. Take the brute force method as an example. Searching all possible scheduling combinations is guaranteed to find the optimal scheduling plan, however, the scheduling overhead increases exponentially when the number of SBs increases and the searching is even more complex with the searching of all possible subarray combinations. To have an efficient solution in the long run, we compare multiple scheduling algorithms from the literature and propose to explore the use of abstract structure with policies to generate optimal or near-optimal algorithms, and leverage the off-line Reinforcement Learning (RL) method to solve the scheduling problem. In this work, we propose

using Action Space, which generates all possible scheduling options given the states rather than comparing each SB. We also provided two conventional methods that have optimal or near-optimal performance with three metrics: throughput, cost, and job latency. Lastly, we leverage the RL method to achieve near-optimal or optimal performance compared to the conventional method. When scaling the number of antenna subsets, we find the prediction time of the RL method stays nearly constant while the conventional method has a prediction time that grows exponentially, we found the RL method can greatly reduce execution time compared to conventional methods starting 8 or more antenna subsets while maintaining the highest average performance.

1 Introduction

The National Radio Astronomy Observatory (NRAO) has been building the next-generation Very Large Array (ngVLA). It uses radio interferometers that use interference of superimposed waves to extract astronomy information [36]. It operates at frequencies of 1.2 GHz to 116 GHz and can provide large-scale astrophysical imaging with an order magnitude greater sensitivity and spatial resolution than previous instruments [65]. Previous instruments include Very Large Array (VLA), Expanded VLA (EVLA) (or Jansky Very Large Array (JVLA)), and Atacama Large Millimeter/submillimeter Array (ALMA). VLA, the first generation of radio telescope developed by NRAO scientists, was able to make images with an angular resolution comparable to the best optical telescopes [49]. Later, the expansion of the original VLA resulted in JVLA. The improved version brought two more frequency bands (S and Ka) to provide 100% frequency coverage from 1 to 50 GHz. Compared to VLA, JVLA improved the instantaneous bandwidth up to 8 GHz, which is two orders of magnitude increase from VLA. Most importantly, JVLA started Routine Dynamic Scheduling for almost all observations, allowing scientists to match projects with their required atmospheric conditions with much more efficient use of telescope time [70]. ALMA, on the other hand, started its operations in March 2013. Its receivers can detect wavelengths from 8.5 to 0.3 mm (35 GHz to 950 GHz), and the antennas function as a whole of up to 66 antennas with miles of distance across. In addition, every interferometer is equipped with a correlator that is a supercomputer operating at speeds of 17 quadrillion mathematical operations per second to correlate the antennas' signals and divide the signals by frequency from all instruments to form high-resolution images [4].

Besides the instrument itself, VLA, JVLA, and ALMA use schedule algorithms to select appropriate observing programs to ensure the efficient use of radio telescopes. VLA specifically considers telescope slew rate, source separation, atmospheric opacity, and SB rank priority as necessary input for dynamic scheduling [16]. All observatories use a Dynamic Scheduling Algorithm (DSA) or similar algorithm to break down factors that determine whether a given source of Scheduler Block (SB) can be observed at all at the current time and use the overall score of all factors to decide the highest-score SB to observe. There are

also different scheduling policies to use at different times to fit the scientific and operational needs [20]. However, for ngVLA, previous scheduling algorithms are not suitable as they do not schedule multiple observation tasks simultaneously while the ngVLA array can be logically divided into multiple subarrays with the ability to operate independently. Therefore, additional improvement is needed on previous scheduling policies as well as the ngVLA subarray allocation method to achieve high efficiency in ngVLA operation.

The previous DSA on VLA, JVLA, and ALMA focused on balancing factors to choose appropriate observing programs. In general, the goal of any scheduler assigning resources to perform tasks aims at one or more goals. The goals include fairness, efficiency, high throughput, low response time, predictive, low overhead, high resource utilization, no indefinite postponement, enforcing priorities, etc. Fairness measures the difference between the desired load which each job or process runs in a reasonable amount of time and the actual load where how each job and process runs. The maximum of unfairness is when one job or process uses all the resources at all times [3]. Efficiency, similar to fairness, measures the resource occupancy of a job and process in run time. Maximum efficiency is achieved when the job or process uses the maximum resource possible, and minimum efficiency occurs at the idle state [114]. The measure of throughput is the completion rate of jobs and processes [14]. Throughput on high-priority SBs is one of the few key attributes in the scheduling algorithm for this work which we would like to maximize the number of A grade observation tasks to be scheduled during an observing season.

In general, a well designed scheduling wants tasks to be predictable, maximize resource utilization, enforce priorities. For jobs and tasks to be predictable means given tasks should take the same amount of time and resources in a given condition [17]. However, in a real scheduling scenario, both the observation tasks and the weather conditions have high variance, for the tasks to be predictable will require a large amount of effort in forecasting the weather conditions and simulating the observation tasks. Resource utilization, in general, is maximized when running jobs or processes fully utilize the underutilized resources as it will achieve high resource utilization by keeping the instrument busy at all times [42]. A scheduler that enforces priorities will allows the job or process with the highest priority score to have a better chance to run earlier because of its influence [69].

As ngVLA, the next-generation radio/mm telescope, operated by NRAO improved ten times the sensitivity and spatial resolution of previous generations of the telescope, the scheduling algorithm shall be carefully improved or redesigned to back the telescope operation for efficiency and productivity. The DSA applied to previous generation VLA is concerned only with selecting one single sub-array from SBs to execute at any moment. It analyzes SBs with their attributes including Frequency, LST, Phase RMS, weather attributes like atmospheric opacity, angular resolution, largest angular scale, and observation time to filter for capable SBs for current and forecast conditions, then evaluates SBs with weighting algorithm to find the SB with the highest score for operation. The DSA is well-designed for single SB operation. However, ngVLA with multi-purpose subarrays can schedule multiple observation tasks simulta-

neously at one-time step execution. For instance, there are Main Array (MA), Long Baseline Array (LBS), and Short Baseline Array (SBA), and each subarray can operate independently for multiple observations, in practice, there are in the order of 20 different subarrays from the ngVLA. Therefore, this work aims to address the challenge of scheduling methods for ngVLA. We pay attention to three metrics the most: throughput on high-priority SBs, cost, and job latency, which will guide us in designing the scheduling algorithms. Here, the cost is the key metric to optimize refers to the product of the priority score of all SBs and the time duration that they execute.

In this work, we focus on building an efficient scheduler structure that can simulate SAs and SBs while being scalable and also leverage Reinforcement Learning (RL) techniques to improve upon DSA for astronomy observation scheduling. The scheduler structure is built with a few abstractions, multiple policies, an RL scheduling method with episode learning, and a policy update threshold setting. It can generate SBs, output episodes, and end evaluations. The structure supports a scalable scheduler simulator working along with some scheduling policies and an off-line RL model. We started from fundamental blocks like SAs and SBs to final performance evaluation, our simulator has the capability of creating different numbers of Antenna groups and corresponding SAs, generating tasks with priority scores, and adopting different policies and the RL method to perform near-optimal scheduling strategy. There are a few abstractions used to solve the problem efficiently. First, two abstractions adopted from DSA that have SAs as the power set of antenna groups which will cover all possible combinations of antenna subsets, and SBs as instance representations of all observation requests with attributes to map into one of the SAs. The third one is action space, where each action consists of one or more SAs that are compatible to execute simultaneously. Specifically, we use action space abstraction with policies that can achieve optimal or near-optimal scheduling decisions, then leverage the off-line Reinforcement Learning (RL) method to train the model to reach the best policy performance and RL improves scheduling efficiency. The contribution of the project is as follows: development of a new scheduling model with scalable input, adding of a new abstraction for multi-scheduling scenarios, designing a policy with optimal or near-optimal throughput, cost, and job latency, leveraging off-line RL to solve the scheduling problem with competitive performance, and providing a comprehensive review and analysis of existing literature in a scheduling problem. We also study the limitations and possible directions when solving the scheduling problem with more variables. For instance, when there are time variances between SBs, and/or different numbers of antenna groups available at the beginning, then a generalized solution is needed for optimal or near-optimal scheduling.

The remainder of this paper is structured as follows. Section 2 presents related work, providing background information to the astronomy observation scheduling problem and a literature review on scheduling algorithms in computer science along with their applications. Section 3 presents the design of our RL-based scheduling method. Section 4 presents a performance evaluation for a comprehensive comparison with three scheduling methods and one baseline.

Section 5 closes this thesis with remarks on our future work.

2 Related Work

In this section, we will first present the previous VLA scheduling algorithms with pros and cons. Then, we will present a literature review related to work in job scheduling algorithms as well as using the RL method to solve the problem. Lastly, we will identify significant applications that are related to ngVLA, and compare and contrast different methods in order to design the one to be used in this project.

2.1 Generations of VLAs and Their Scheduling Algorithms

Very Large Array (VLA), as a radio astronomy observatory, consists of tens or hundreds of massive telescopes across nearly 100 kilometers. It can observe astronomy images with spatial resolution and angular resolution competitive with the best optical telescope like the Hubble Space Telescope (HST). The first generation of the VLA operates as an interferometer, which combines signals from multiple telescopes to achieve high angular resolution with key achievements such as observation of black holes, protoplanetary disks, and complex gas motions since operation back in the 1980s. [108]. Karl G. Jansky Very Large Array (JVLA), as the upgraded version of the VLA, installed new receivers and digital back-ends, allowing it to achieve higher resolution and sensitivity. As it covers a broader range of frequencies, from 73 MHz to 50 GHz, JVLA is used to explore more cosmic phenomena and study magnetic fields [55].

The other two variations of VLAs, ALMA and Very Long Baseline Array (VLBA), also use radio signals for astronomy observation. ALMA consists of 66 high-precision antennas and can observe in millimeter wavelengths compared to the centimeter-wavelength of VLA observatory [15]. VLBA uses long baseline interferometry across 8,611 kilometers in New Mexico and can observe radio wavelength ranging from three millimeters to ninety centimeters with scientific achievements including cosmic objects' spins, shapes, and movements [72].

As indicated by Dr. Hiriart [38], the VLA scheduling algorithm can be divided into three stages: i) filtering eligible observation tasks, ii) assigning a priority score to each task, and iii) scheduling selected tasks for a given time horizon. The main contribution of the filter function is fitting the system to predict the opacities in the near future. One of the arguments is to perform predictions from measurements of the surface ambient and dew point temperature [9]. However, the atmosphere is not strictly related to the distribution of water vapor, so using surface measurements cannot predict opacity well. VLA instead uses weighted combinations of surface measurement and day of the year to find the fits for opacity prediction [16]. Then, based on the opacity prediction, ineligible SBs are filtered from the pool of the schedulable blocks. The second stage is to assign a priority score to each eligible SB. The priority score is calculated for each SB in criteria such as primary, urgency, science rank, over-

ride, etc. Each criteria type has its range and weight. The final priority score is the product of a weighted sum using all criteria: $P = \sum_{i=1}^{n} w_i * c_i$, where w denotes the weight and c denotes the criteria. The third stage is assigning priority scores, the algorithm then creates observation scheduling for all eligible tasks. First, it adds all selected SBs to the schedule if the time intervals do not conflict with the schedule window. Second, it sorts SBs by priority, and then fits or inserts SBs into the schedule. Last, it adds SBs outside of the time interval to the schedule. In practice, the final schedule is presented to the operator, and only the first SB in the schedule is accepted. The schedule is rearranged once execution ends, so the algorithm is guaranteed to produce optimal execution performance with the setting [38].

For the next-generation VLA such as ngVLA, the previous scheduling algorithms would not be optimal. This is because in ngVLA, a single telescope can be divided into three primary subsets: Main Array (MA), Long Baseline Array (LBS), and Short Baseline Array (SBA). MA can be further split into three sub-arrays: Core, Plains + Core, and Mid-baseline. This architecture allows ngVLA to execute multiple observatory tasks simultaneously. But DSA that schedules one task at a time will greatly reduce the performance. Therefore, a new algorithm to have ngVLA achieve high efficiency while prioritizing the key science goals with the highest weights is needed [96].

2.2 Scheduling Algorithms in Computer Science

The scheduling problem in the astronomy observation shares similar attributes as the compute resource (e.g., CPU) scheduling problem with goals such as minimizing resource starvation and ensuring fairness among parties utilizing the resource [60] in the Computer Science domain. In computer processes, scheduling is maintained by policies and mechanisms to select a runnable process to become the current process. When the current process is interrupted, the scheduler chooses the process with the highest priority to run [23].

The purpose of scheduling algorithms is to make the target system have fairness, flexibility, reliability, and efficiency in performance by spreading task load on processors, so each job is scheduled to adaptable resources and adaptable time, produces the optimal schedule sequence under proper constraints [90, 57]. The objectives of a scheduler usually concern the sequence and length of time that processes may run and can be summarized as a scheduling algorithm or policy. The scheduling algorithm or policy generally gives each process a fair share of the resources to execute [61, 73]. Keeping the CPU or processor busy all the time will ensure the efficient use of the whole system. High throughput can be achieved by executing the largest possible number of jobs in a given amount of time, and at the same time, the response time would be low because the wait time is short. Being predictable means the finish time of any task should be nearly constant in the same system so the running time of the task batch can be predicted. The overhead should be minimized by keeping resource use efficiency when minimizing scheduling and context switching time [110]. To maximize resource use, processes that will use underutilized resources shall

be in favor and run more often, which will ensure better performance without burdening heavily utilized resources [109]. To avoid indefinite postponement, priorities can be enforced so each process can be scheduled meaningfully with corresponding scores [115].

The schedulers can be classified as long-term schedulers, short-term schedulers, and medium-term schedulers. When job batches get into the system batch queue, the long-term scheduler plans the resource scheduling for the batch jobs. It selects processes from the queue and loads them into memory for execution. The long-term scheduler achieves the best performance when selecting a good process mix of I/O-bound and CPU-bound processes. The long-term schedulers usually ensure that all processes are organized in a fair manner, e.g. allocating computing resources and accounting for overall performance [59, 83, 112]. A short-term scheduler selects the processes from the processes that are ready to execute and changes the processes' state from ready to running. Compared to long-term schedulers, short-term schedulers are used more often and are invoked whenever an event occurs. Such events include time-based interrupts, I/O interrupts or completions, operating system calls, signal sending and receiving, etc. Short-term scheduling may provide an opportunity to preempt a currently running process in favor of another [12, 66]. The medium-term scheduler is responsible for moving suspended or swapped-outed processes back into a pool of ready queues. Any process can be suspended or swapped out because of events or system calls. The processes being swapped, are removed from the main memory and are stored in a swapped queue in the secondary memory, so the space in the main memory is freed for other processes. Once the suspending condition for processes is removed, the medium-term scheduler will attempt to allocate resources for the processes to be ready again [82]. Following, are different scheduling algorithms used in Computer Science.

First Come First Serve (FCFS), as one of the simplest scheduling algorithms, prioritizes processes based on their arrival time in the queue, ensuring that those arriving earliest are scheduled first. Consequently, processes arriving later are scheduled accordingly [28]. It can be easily implemented by using a FIFO queue. However, FCFS suffers from convey effect; that is if there is one process requiring intensive resources to complete, all other processes requiring less resources will be slowed down, increasing the overall wait time [121]. FCFS is the basic scheduling algorithm used by operating systems and networks in the single processor with single interrupt line setting [92].

Shortest Job First (SJF) is a scheduling process used in grid computing [37], CPU scheduling [77, 35], and cloud-based software systems [86]. It selects the waiting process with the smallest execution time to execute next and is generally used for long-term scheduling. It has the advantage of having a minimum average waiting time compared to other scheduling algorithms, but the demerits include starvation or indefinite blocking of processes, resulting in prolonged execution times. Additionally, estimating execution length can be complex in certain scenarios.

Longest Job First (LJF) is as the opposite of SJF, which prioritizes processes or jobs with the longest execution time first. Applications using LJF

include Cloud environment load balancing [53] and Job Shop Scheduling [88]. In real-world scenarios, when two processes have different execution times, LJF schedules the longer one first, but when two processes have the same execution times, it uses arrival time or priority score to break the tie. The advantage of LJF is that the finishing times for all processes are approximately equal but the average waiting time and average turn-around time will be high, and it may lead to convoy effect as FCFS.

Priority Scheduling is one of the preemptive scheduling algorithms that works based on the priority scores of processes. It has a function that assigns each process a priority score, and the processes with higher scores will be executed earlier than those with low scores. If multiple processes have the same score, SJF, LJF, and FCFS can be the tiebreaker. Priority Scheduling are used widely for interactive programs [100], cloud computing system [33], and wireless senor network [45].

Round Robin algorithm can be viewed as a preemptive version of the FCFS algorithm. In this approach, each process or job is assigned a fixed time slot for execution, regardless of whether the currently running process or job has been completed. When the allotted time expires, the current process is halted, allowing the next one in the queue to execute for a fixed duration. Like FCFS, Round Robin is simple and easy to use, and because it uses a fixed time slot for every process, it is also starvation-free and fair to processes in a batch [80]. Round Robin is widely used in cloud computing [76, 102], CPU scheduling [52, 101], and computer networks [89].

Shortest Remaining Time First (SRTF) can be viewed as a preemptive version of SJF, where the processor is allocated to the process or job that is closest to completion. SRTF and SJF are identical when all processes arrive at the same time, but when processes arrive at different times, the SRTF scheduler will compare the remaining time of the current process and the newly-arrived process. Like SJF, SRTF handles short processes very fast, and it only requires a small amount of overhead when a process is completed or a new process is added. However, SRJF may also lead to process starvation where long processes are held off indefinitely, and both SJF and SRJF are practically hard to implement because of the difficulty of predicting the burst time of the processes. Application of SRTF includes data center networks [28, 5], Cloud load balancing [119], and traffic congestion [1, 54].

Longest Remaining Time First (LRTF), as a preemptive version of the LJF algorithm, is similar to SRTF and SJF. LRTF schedules those processes that have the longest remaining processing time for completion first. Similar to SJF, LRTF has the advantage of finishing all jobs around the same time but will cause a high average waiting time and high average turn-around time, which leads to a convoy effect. Application of LRTF includes broadcast scheduling [18].

Highest Response Ratio Next (HRRN) is one of the most optimal non-preemptive scheduling algorithms. The algorithm requires finding the response ratio of all available processes, which is calculated by $(W + S)/S$. Here, **W** is the waiting time of the process, and **S** is the burst time of the process. This algorithm reduces the waiting time for longer jobs while still encouraging short

jobs, which gives better performance than the SJF algorithm. The demerit of HRRN is that sometimes it is impossible to know the burst time of every job, and calculating the response ratio may cause an overload on the processor. HRRN is applied in some soft real-time systems [10, 50], Cloud Computing Policy [91], and grid computing environment [56].

Multiple Queue Scheduling is a method that divides the ready queue into various classes and each class may have its scheduling queue with priority. For instance, there could be system processes classified as queue 1, interactive processes as queue 2, and batch processes as queue 3. Queue 1 has higher priority over queues 2 and 3, and queue 2 has higher priority than queue 3. This way, the Multiple Queue Scheduling guarantees that the system processes are executed instantly and followed by interactive processes. The main advantage is having low scheduling overhead. Multiple Queue Scheduling is applied in telephone call centers [81, 19], Web services [111, 46], and computer systems [103, 62].

Multilevel Feedback Queue Scheduling (MLFQ) brings one step further than Multilevel Queue Scheduling, where the process can move between queues instead of being permanently assigned to a class. By doing so, the scheduling overhead is increased but the scheduling is more flexible. MFLQ is the most complex algorithm to design but during the experiment, MFLQ achieves the smallest average wait time in many cases, maintaining good performance as MLQ and will not have starvation problem [107]. MLFQ is used in food ordering system [44], and real-time operating system [39, 13].

Based on execution order decisions, these algorithms can be classified as static scheduling algorithms and dynamic scheduling algorithms. Algorithms like SJF and LJF are considered static methods because the sequence of task schedules is set before execution and will not change during runtime. Whereas HRRN and MLFQ make decisions during the runtime of the system, they are considered as dynamic since the design is more flexible [69, 97, 106].

2.3 Reinforcement Learning

Reinforcement Learning (RL), as one of the most remarkable branches of machine learning (ML), has been applied to machine scheduling problems [47]. Compared to traditional deep learning methods, RL does not rely on supervised output to improve performance. Instead, it uses a reward function to guide the model towards an optimal solution. As illustrated in Figure 1, given a state S, the learning agent has the option to perform an action in the action space A. Each action it takes will result in a reward through the reward function, and the performance is evaluated by the overall reward it earns [117].

RL algorithms can be divided into model-free RL and model-based RL, depending on whether the learning agent has access to or can learn the model of the environment, which is a function that predicts state transitions and rewards [41]. We present the model-free RL and model-based RL in the subsequent sections.

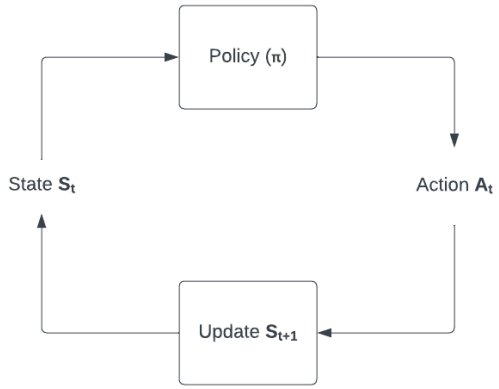

Figure 1: The schedule-action-learning loop. At time t, the State includes all SBs and accumulated rewards. The policy network takes in the state to choose an action. Once the action is executed, the state and the reward will be updated for the next iteration.

2.3.1 Model-Free RL

Within model-free RL, algorithms can be differentiated based on whether they utilize Policy Optimization or Q-learning techniques. For **Policy Optimization** approaches, the optimal policy can be represented as $\pi_\theta(a|s)$ where π is the policy, θ are the parameters to be optimized in the policy, given action a in state s. Through gradient ascent on expected return, the policy gradient can find the best parameters θ to approximate the optimal policy. Gradient ascent is done iteratively by calculating the gradient from some data to update the weights of the policy function that maximizes the expected return. The policy objective $J(\theta)$ can be defined as: $J(\theta) = V^{\pi_\theta}(s_0)$, where V^{π_θ} is the expected value of a policy π_θ with parameters θ. The search for an optimal policy objective can be computationally expensive, so using approximation by searching for the local maximum in $J(\theta)$ can be done by ascending the gradient of the policy using episodic samples. When following the gradient towards the optimal $J(\theta)$, we get: $\theta \leftarrow \theta + \alpha \nabla J(\theta)$, where α is the learning rate, and $\nabla J(\theta)$ can be expressed as $\nabla J(\theta) = \mathbb{E}\big[\nabla \ln \pi_\theta(s,a) Q(s,a)\big]$.

Q-learning, on the other hand, uses approximator $Q_\theta(s,a)$ to learn the optimal action-value function $Q^*(s,a)$. The key idea is to leverage a combination of features and their weights to approximate the Q-function. First, it considers the features that determine the state representation. Second, it performs updates based on the weights of features during learning. Lastly, it estimates the optimal action-value function $Q^*(s,a)$ by summing the features and their weights. The feature vector, $f(s,a)$, is represented by n number of state features, and $|A|$ number of actions: $f(s,a) = (f_1(s,a), f_2(s,a), ..., f_{n\times|A|}(s,a))$. The weight

vector, w, represents the weights for each feature-action pair. Given the feature vector f and weight vector w, the Q-value of a state can be represented as:

$$\begin{aligned} Q(s,a) &= f_1(s,a) \cdot w_1^a + f_2(s,a) \cdot w_2^a + ... + f_n(s,a) \cdot w_n^a \\ &= \sum_{i=0}^{n} f_i(s,a) w_i^a \end{aligned}$$

During learning, the weights are updated using $w_i^a \leftarrow w_i^a + \alpha * \delta * f_i(s,a)$ till weights converge.

Compared to the Policy Optimization methods, which directly optimize for the objective the agent directs, Q-learning methods indirectly optimize an agent's performance. Q-learning is more sample efficient when the training is successful, but the Q-learning method through training Q_θ has many failure modes, which tend to be less stable [67].

Popular Policy Optimization methods includes Policy Gradient [104], A2C/A3C [67], PPO [93] and TRPO [94]. Q-learning methods includes DQN [68], C51 [11], QR-DQN [21], and HER [6]. A few algorithms carefully trade-off between the strengths and weaknesses of either method are DDPG [98], TD3 [27], and SAC [34].

2.3.2 Model-Based RL

Within Model-Based RL (MBRL), algorithms can be categorized based on whether the model can be learned or is pre-defined. MBRL improves the sample efficiency and reduces the amount of trial-and-error throughout the learning environment. The environment model is refereed as the abstraction of the environment dynamics which can be learned or given [64]. There are a few components to achieving the model-based method: planning action purely for the current state, using the planning algorithm as an "expert" to assist policy, creating real and/or simulated episodes to augment the model, and embedding planning into policy itself.

Model-based Deep RL with Model-Free Fine-Tuning is a promising approach for simulated quadrupedal robots to produce stable and plausible gaits [71]. The method first initializes a model-free learner to learn the dynamics function $f_\theta(s_t, a_t)$, and then uses a combined medium-sized neural network dynamics models with Model Predictive Control (MPC) in a model-based RL algorithm. It characterizes having a high task-specific performance by using a model-free method along with sample efficiency by using a model-based method.

Both Alpha-Go [99] and ExIt [7] use sophisticated search techniques like tree search along with deep neural networks to make actions. Deep neural networks generalize plans from searches and search is also guided by the neural network. Feinberg *et al.* [26] proposed a method that uses data augment to assist model-free methods. It uses a hybrid approach that contains a dynamics model to simulate the short-term horizon and an improved Q-learning method to estimate long-term value. Which provides improved sample complexity on a

range of continuous action benchmark tasks. Imagination-Augmented Agents (I2As) [78] uses a novel approach that interprets predictions from a learned environment model to make predictions as additional context in deep policy networks, resulting in improved data efficiency, performance, and robustness to model misspecification.

2.4 Applications of the Scheduling Algorithms

Scheduling problems pervade the computing domain. The effectiveness of a scheduling algorithm is often assessed based on metrics such as throughput, wait time, latency or response time, and fairness. Many systems use schedulers such as operating system schedulers, network schedulers, I/O schedulers, and job schedulers. The goals of the schedulers may vary depending on their purpose. In the following, we introduce the applications of the schedulers.

2.4.1 Process Schedulers

The process scheduler, as the component in the operating system, can determine when and which processes should be running. Four events include I/O request, termination, interrupt, and I/O complete, requiring the scheduler to kick in and make a decision [63]. The nature of the process and events are likely to be random in real scenarios, which means a new process or request can come in at any point and the resource use of each process varies. There are multiple goals from throughput to resource allocation so each algorithm cannot satisfy all goals but only favor some over others. The First Come First Serve Scheduling (FIFO) is the most straightforward approach. It is simple and intuitively fair. However, it is not preemptive and a long-running job will delay all other jobs. The round-robin scheduling dispatches processes in the FIFO sequence but only for a limited amount of time. The advantage of the round-robin method is that every process gets an equal share of the CPU, but it cannot prioritize highly interactive processes. Other popular scheduling methods include shortest remaining time first scheduling, priority scheduling, multilevel queues, multilevel feedback queues, lottery scheduling, etc. These methods usually are used in environments where a single process gets to run at one time, but fine-tuning of each method is needed in the multiprocessor environments, where multiple processes are scheduled at once [40]. There is an RL-based priority assignment method for multi-processor real-time scheduling [58] that consists of the task set embedding mechanism, response time analysis-based policy gradient RL, and guided learning schemes. This method achieves 7.7% enhancement in schedulability ratio in 64-sized task sets and an 8-processor platform, where the schedulability ratio is a metric used to evaluate the effectiveness of a guarantee test for real-time scheduling algorithms.

2.4.2 Network Traffic Schedulers

A network scheduler manages network packets in the transmission. It uses a queuing system to store network packets temporarily until they are dispatched. The core responsibility of network schedulers is reducing network congestion, latency, and packet loss while achieving quality of service (QoS) [24]. Algorithms in network schedulers can broadly fall into classful or classless. Classful queuing disciplines allow the creation of classes and filter packets into each class and queuing corresponds to their class, whereas classless queuing keeps the queuing discipline simple [79]. There exists an RL agent-based network controller [32], which demonstrates comparable performance to established heuristics such as Earliest Deadline First (EDF), achieving optimal results. The study shows that the RL-agent can equally share bottleneck link capacity among the competing flows, resulting in better performance than the idealized TCP protocol.

2.4.3 I/O and Job Schedulers

An I/O scheduler is similar to a process scheduler in many ways with the high-level goal of minimizing time waste, prioritizing certain requests, giving a fair share of disk bandwidth to each running process, and guaranteeing request issues before the deadline [95]. A job scheduler controls the execution of jobs in a computer at a higher level, provides an interface, automates the job process, and prioritizes queues to control jobs [2].

2.4.4 Job-Shop Scheduling Problem (JSSP)

JSSP is a well-known schedule optimization problem in computer science. Just like the traveling salesman problem (TSP), JSSP with the sequence-dependent setup is also NP-hard because TSP is a special case of JSSP with a single job [116]. In general JSSP setup, there are n jobs $J_1, J_2, ..., J_n$ with corresponding processing times, and there are m machines with varying processing power. The goal is to minimize the total length of the processing to finish all n jobs [29]. There are approaches using genetic algorithms [22, 31, 116], conventional heuristics [25, 30, 87] and hybrid approaches [74, 113, 118] using both to solve JSSP.

2.4.5 Other Real-World Schedulers

Other real-world scheduling problems include urban transit schedule problems [105], NASA space shuttle payload processing tasks [120], and machine scheduling problems [51, 84, 8, 75]. The RL-based scheduling methods show promising performance and can achieve better results than conventional scheduling methods [48].**Reinforcement Learning (RL)**, as one of the most remarkable branches of machine learning (ML), has been applied to machine scheduling problems [47]. Compared to traditional deep learning methods, RL does not rely on supervised output to improve performance. Instead, it uses a reward

function to guide the model towards an optimal solution. As illustrated in Figure 1, given a state S, the learning agent has the option to perform an action in the action space A. Each action it takes will result in a reward through the reward function, and the performance is evaluated by the overall reward it earns [117].

3 RL based Scheduling for the Next-Generation Very Large Array

VLA, JVLA, and ALMA use the DSA scheduling algorithms known to be dynamic and select appropriate observing programs while ensuring efficient use of radio telescopes each cycle. However, owing to the design constraints of conventional telescopes, they are limited to executing a single observation task at any given time. In contrast, the **ngVLA** can be strategically segmented into five subsets, each offering diverse sensitivities and angular resolutions. This modular design enables independent operation for multiple observations, catering to a range of scientific objectives simultaneously. With these subsets, researchers can request astronomy observation with specified subsets they intend to use. Therefore, ngVLA cannot adopt DSA directly from VLA, JVLA, and ALMA, which cannot schedule multiple observation tasks simultaneously. Additional improvement is needed in DSA for efficiency in ngVLA operation.

The ngVLA can be divided into three main subsets: Main Array (MA), Long Baseline Array (LBS), and Short Baseline Array (SBA) [65]. The MA subset consists of 214 x 18m antennas [85] with a 1005.4km maximum baseline distance and the LBA subset consists of 30 x 18m antennas with a maximum baseline distance of 8856.4km. The MA and LBA subsets drive the sensitivities and angular resolutions offered by the ngVLA. The SBA subset consists of 19 x 6m antennas [96] and will be sensitive to a portion of the larger angular scales poorly sampled by the MA subset. On top of that, MA can be further split into three subsets: Core, Plains, and Mid-baseline for different observation purposes. Theoretically, there are use cases for all combinations of the five antenna subsets and each observation request will fall into one or multiple of the five subsets. Based on the observation needs, the subsets then calibrate the antennas so the observation program can be executed for a certain period.

The subsets which consist of tens or hundreds of antennas can be viewed as abstract units since antennas work in groups. There are five subsets in ngVLA. Therefore, the scheduling problem can first be simplified by using five subsets to create SAs using subset combinations. There are a total of 63 SA combinations representing the power set of the subsets and the empty set. For instance, subset 1 can be seen as Core, subset 2 is Plains, subset 3 is Mid-baseline, and the Combination of subset 1, 2, and 3 is MA. Subtests 4 and 5 will correspond to LBS and SBA. With these five subsets, there can be combinations like SA-12 Core and Plains, SA-13 Core and Mid-baseline, SA-14 Core, Mid-baseline and LBS, etc.

SBs can be used to represent individual observation tasks. In addition to the required subsets, each SB has its own characteristics and relative priorities. An additional attribute of each request other than sub-array is the scientific importance of the observation task with a priority weight; a request with higher priority implies that it be scheduled earlier than another with a lower priority score. It is also noted that some observations require certain weather conditions; when a weather condition is met, these requests should be scheduled immediately. In this work, to formalize the scheduling problem, the algorithm in the approximation only considers the priority score as weight and needed subsets as features of all SBs.

The problem of the ngVLA observation scheduling can be mapped into the problem of scheduling the 5 classes of subsets to SBs. The scheduling problem then can be constructed as follows. Given a batch of SBs with each SB specifying SA, the SBs are mapped to their requested SAs offline. Each SA inherits the priority score of the highest SB in the SA. By using this method, each SB that uses one subset or multiple subsets can be mapped into one of the 31 SAs and each SA can hold multiple SBs using the same subsets. During the online scheduling, at each time step, each SA chooses its mapped SB with the highest priority and the scheduler selects SAs to run simultaneously at the time step. To choose which SA to run, the scheduler uses a scheduling policy such as the Size-Prioritized policy or Cost-prioritized policy to select SAs in sequence. The selected SAs (forming an action) should have no subset overlapping. For instance, there are SA-12 (SA using subset 1 and subset 2) and SA-23 (SA using subset 2 and subset3), because both SAs will request the use of subset 2, therefore the two SBs in the SAs cannot be scheduled or executed at the same time and we call it has overlapping subsets. By scheduling, the scheduler aims to improve the performance on throughput, cost, and job latency. For instance, if there is an SB that requires SA-145 consisting of subsets 1, 4, and 5, then the SB is mapped into SA-145 which only consists of subsets 1, 4, and 5. If the SB has the highest weight among the SBs mapped to SA-145, SA-145 will pop the SB to execute. When SA-145 is scheduled to execute, this SB is executed.

The scheduling problem can be viewed as JSSP with constraints. SAs correspond to machines in JSSP, SBs represent jobs in JSSP but each SB can only be executed in one specific SA. Therefore, to evaluate the performance of each method, we can use the number of SBs executed divided by the finish time step (i.e., throughput), the sum of the products of all request weights and their corresponding time duration (i.e., cost) and the sum of response times for all requests (i.e., job latency) as the conventional metrics similar to the evaluation in JSSP [43]. For the solution to be optimal, it should have the highest throughput with the lowest cost and job latency. During the evaluation, the average throughput, overall cost, and job latency will be compared across different policies and methods.

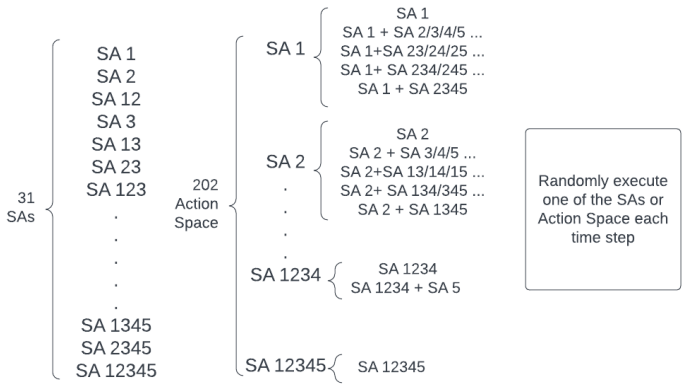

Figure 2: The illustration of SAs using 5 subsets and action space using all possible SAs. Round Robin method without action space will select one of thirty-one SAs whereas the Round Robin method with action space will select from two-hundred and two action spaces.

3.1 Simulator Setup

The simulator is constructed as follows. Class *SubArray* initializes the number of subsets that will be used, and then creates all possible combinations of subsets to form *SubArray* that will hold *ScheduleBlocks*. Class *ScheduleBlock* initialize *ScheduleBlock* instance with required subsets and associated weight. Class *Process* will contain the *SubArray*, and run and record the schedule metrics including cost, throughput, and job-latency. During simulation, *Process* will generate random *ScheduleBlocks* and schedule the execution with a selected scheduling method. The methods include Size-Prioritize method and Cost-Prioritize method and a user can select the methods.

Given a batch of random requests, the simulator first maps SBs into corresponding SAs. For each time step, one or more SAs will be selected to execute their mapped highest-priority SBs. The cost at each time step is the product of the sum of SBs' weights and the current time duration, which will be accumulated over time to get the overall cost. The process of executing SAs will continue until no SB is assigned to SAs. The goal is to increase throughput (minimize the number of time-steps to complete all SBs) and minimize the overall cost and job latency, in which a maximum number of SAs shall be scheduled and prioritize high-weight instances in each time step.

3.2 Round-Robin Policy

The Round-Robin method randomly selects available SBs to schedule. It does not account for either the weight of SBs or the available SA combinations that do not have subset overlap. For instance, when there are two SBs in SA-1

and SA-2, this method would never combine these two SBs even though they can be scheduled in parallel. The performance of this method should not be very efficient for ngVLA. This method is an analog to DSA in VLA, because it schedules only one SB at a time. The performance will be a baseline for the next method to compare to.

Action space consists of all actions, each of which does not have overlapped subsets. Next, we build the action space of the scheduler using all SAs, as shown in Figure 2. We find all combinations of SAs that do not have overlapped subsets. For instance, SA-134 can be combined with SA-2, SA-5, or SA-25 to schedule them in one time step instead of multiple time steps. Then we use the Round-Robin method with the action space to perform the same schedules. Figure 2 illustrates the sets of SAs and Action Space where SAs consist of SBs that use all the subsets they support. For instance, SA-2345 (the SA using subsets 2, 3, 4, and 5) will hold all SBs that exactly require subsets 2, 3, 4, and 5. Action Space on the other hand consists of one or more SAs that have no overlap.

The method without action space is analog to the scheduling algorithm used in the previous VLA where only one observation task can be scheduled in each time-step. On the other hand, ngVLA allows multiple observation tasks to be executed simultaneously, which would potentially improve the throughput, latency, and cost. For instance, there are three SBs in SA-1, SA-23, and SA-45. Round-Robin method without action space would schedule one each time and take three time steps total whereas action space abstraction enables scheduling all three SBs simultaneously in one time step because they have no overlap and can be scheduled in parallel.

3.3 Size-Prioritized Policy

To further improve overall throughput, we use a size-prioritized policy, which always selects as many SAs as possible instead of being random. Like the previous example, there are three SBs in SA-1, SA-23, and SA-45, even though action space enables the best schedule with all three being executed at the same time, the Round-Robin method has only one in seven chance of selecting the best schedule whereas the size-prioritized policy will always select the maximum number of SAs in the action possible. With the action space abstract, the size-prioritized policy selects the action with the most SAs. Because each SA consists of one or less than or equal to N subsets and selected SAs cannot have overlapping subsets, the maximum SA to be selected equals N.

Given available SBs, the Size-Prioritized policy at each time step then generates a list of possible scheduling options in the action space and selects the option with the largest number of SAs. For instance, there are five SAs with SBs, SA-1, SA-2, SA-3, SA-12, SA-23. All parallel schedules include SA-1 + SA-2 with 2 SAs, SA-1 + SA-23 with 2 SAs, SA-1 + SA-3 with 2 SAs, SA-1 + SA-2 + SA-3 with 3 SAs, SA-2 + SA-3 with 2 SAs, SA-3 + SA-12 with 2 SAs. The Size-Prioritize policy will select the action that executes SA-1, SA-2, and SA-3 at the same time because it has the highest number of SAs with 3. Because

multiple SAs will be selected, the throughput is maximized or overall time-step to complete the batch will be minimized and at the same time, the summed weight at each step will very likely be higher than other options of using fewer SAs. Due to the nature of the size-prioritized policy, the maximum throughput shall be reached, but the cost will not be optimal in case some combinations of SAs may have the highest sum weight but not a maximum number of SAs. For instance, there are SA-123 and SA-45 where SA-123 means SA uses subsets 1, 2, and 3, with the highest weight of 7, and there are also SA-1, SA-2, and SA-3 with a weight of 1. The action to schedule SA-123 and SA-45 will get a total weight of 14 with 2 SAs whereas schedule SA-1, SA-2, SA-3, SA-45 will get a total weight of 10 with 4 SAs. The size-prioritized policy will be in favor of the 4 SAs option instead of the higher-weight option. Below, we will introduce a method that considers weight.

3.4 Cost-Prioritized Policy

The Cost-Prioritized policy will take into account the weight associated with each SA. In each step, it first generates all possible scheduling options as the action space, then scores each schedule option as the summed weight of each SA in the option. Note that we do not need to multiple it with the current time steps since it is the same for all the options. The schedule option with the highest score will be selected to execute. Compared to the Size-Prioritized Policy, it may select the option with less number of SAs but it ensures that the action with the highest summed weight is selected. This policy is optimal in reducing the cost, but it may not select the schedule with the highest subset utilization. For instance, three SBs are using one of subsets 1, 2, and 3 all with weight 1, and another SB is using all subsets 1, 2, 3, and 4 with weight 2. The Cost-Prioritized Policy will select the SB group instead of SB with more subsets, which has lower subset utilization.

3.5 RL based Policy

We design an RL-based scheduling method that leverages the RL method to automatically generate an action plan at each time step. The state S includes two 1x31 normalized matrices that capture all available SBs that are ready to schedule, and their associated features including required SAs and priority weight, and all SAs. The action space A is the two-hundred and two actions from the RL Neural Network. The reward is evaluated by the action it chose using the reward function.

The model is initialized to be random. Then, given a batch of SBs, the model randomly selects SBs to generate training set episodes. Based on the training set episodes, we train the model off-line with a function $F(t) = argmax \sum_i^n g_{cost,step}(SB_i)$. Reward function g accounts for the weight and a number of subsets. This is because the weight is directly associated with the cost, and we want the cost to be minimized, and the number of subsets affects the throughput. Then we randomly sample some episodes and use the reward function to evaluate each

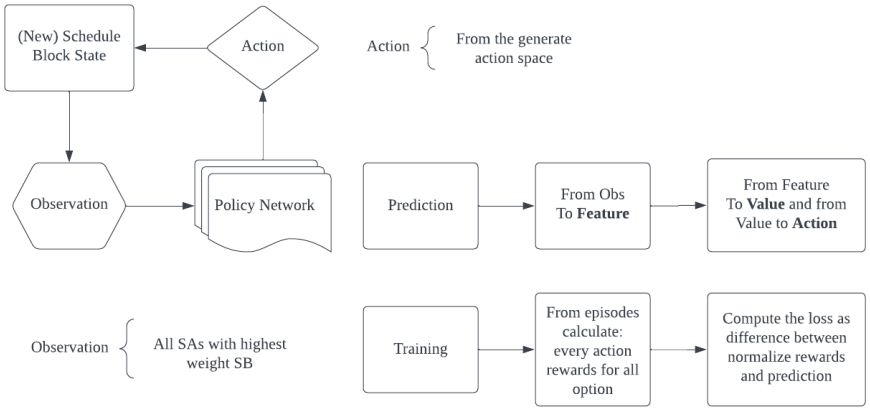

Figure 3: Flow diagram of the RL method for astronomy scheduling

action. Because the action space is discrete instead of continuous, we cannot simply explore nearby action. The action with the highest reward will be the desired option for the episodes. To evaluate the improvement, we first duplicate the policy model before training and then train one model with random episodes from the training sets. After training, the trained model will be compared with the previous duplicated model. Based on the throughput and cost comparison, if the trained model has surpassed the improvement threshold over the original model, the new model is saved, and is duplicated. Then, we use the new model to generate new training examples, train the model with all training examples, and repeat the process until the model does not seem to improve in a couple more of the cycles. In the case that the difference between the trained model and the duplicate model is below the update threshold, the previous duplicated model will be reloaded to generate a new train set.

Both overall cost and overall throughput are used for guiding the models, so there are two policies with reward functions that prioritize either throughput or cost over another. For throughput as the first priority, the reward function $R_{throughput}(s) = throughput + \alpha * summedweight$ where $\alpha * summedweight < 1$ has the throughput as the dominant factor. On the other hand, the reward function for cost as the first priority $R_{cost}(s) = summedweight + \beta * thoughput$ makes summed weight the dominant factor. Selection of the *alpha* and *beta* is not restricted, as long as the output reward secures the action with the highest throughput and cost.

On top of the policy reward function, the threshold for updating improved neural network accounts for cost improvement, step improvement, or both. For instance, the model after training has improvement of x' in step, and y' in cost. If the threshold only concerns step improvement as x, when $x' > x$ then the trained model will be accepted. If the threshold is only concerned about cost

improvement as y, when $y' > y$ the model will be replacing the previous one. Lastly, If the threshold concerns both step and cost improvement $xandy$, the model only updates when both $x' > xandy' > y$. With the setting, we explore the RL method with different values of α and β along with the threshold option to find the best-performing policy. The detailed flow of the RL method is represented in Figure 3.

4 Performance Evaluation

As ngVLA can be divided into five subsets: Core Array, Plains Array + Core Array, Mid-baseline Array, Long Baseline Array (LBS), and Short Baseline Array (SBA) [65] that with a variety of sensitivities and angular resolutions and with ability to operate independently for multiple observations for different purpose, we set the number of subset $N = 5$ and assume each subset can combine freely, which will form $2^5 - 1 = 31$ SAs.

We assume each observation task takes 24 hours as default with thousands of requests/SBs each batch. Then, we take each state as input to generate the best action as output. We iterate the process until no SB is left. We compare the overall throughput, cost, and job latency across different methods.

The brute force search can guarantee to find the optimal policy. First, it uses 31 SAs with SBs with 1-7 weight scores which can produce 31*7=217 unique scored SBs. Then, it searches through the 217 SBs to find combinations that are eligible to execute simultaneously. There will be a total of 79744 combinations which account for the 7 weights. Because the weight of SBs is included, a ranking order of each combination can be assigned. This covers all possible SB combinations. However, in a real scheduling scenario, there are more variances regarding the number of SBs under the same SAs as well as several identical SBs, which may increase the searching space exponentially to guarantee the optimal scheduling plan with throughput, cost, and job latency. The methods below are our attempt to reduce the amount of the search by eliminating most of the obvious cases and using policies to guide the model to produce optimal or near-optimal solutions with a much lower cost.

The Round-Robin method randomly selects one of the SBs or actions for each time step. At time-step 0, 300 random SBs with arbitrary weights between $1 - 7$ are generated and mapped into corresponding SAs. Each SA inherits the priority score of the highest-priority SB in the SA, so the scheduled SA will get the weight of the highest-priority score in the SA to calculate the total cost by multiplying the weight with the time duration. The scheduler continues to execute the action until no SB is left in the queue. The simulation is repeated 100 times with different random seeds and each simulation is considered as an instance.

Figures 4, 5 and 6 show the performance comparison between Round-Robin method with and without the action space when the number of SBs equals to 100, 300 and 1000 in each instance, respectively. The sub-figures on the left are the performance for each instance, and those on the left side show the summary

with a box plot with top line as the maximum, bottom line as minimum, top of the box as 75th percentile, bottom of the box as 25th percentile, and the middle line the average. The values higher than Q3+1.5xIQR or lower than Q1-1.5xIQR are considered outliers and they are marked with circles. Here, Q3 stands for the third quartile, Q1 stands for the first quartile, and IQR is inter quartile range, a measure of the spread of the data as the value of (Q3 - Q1).

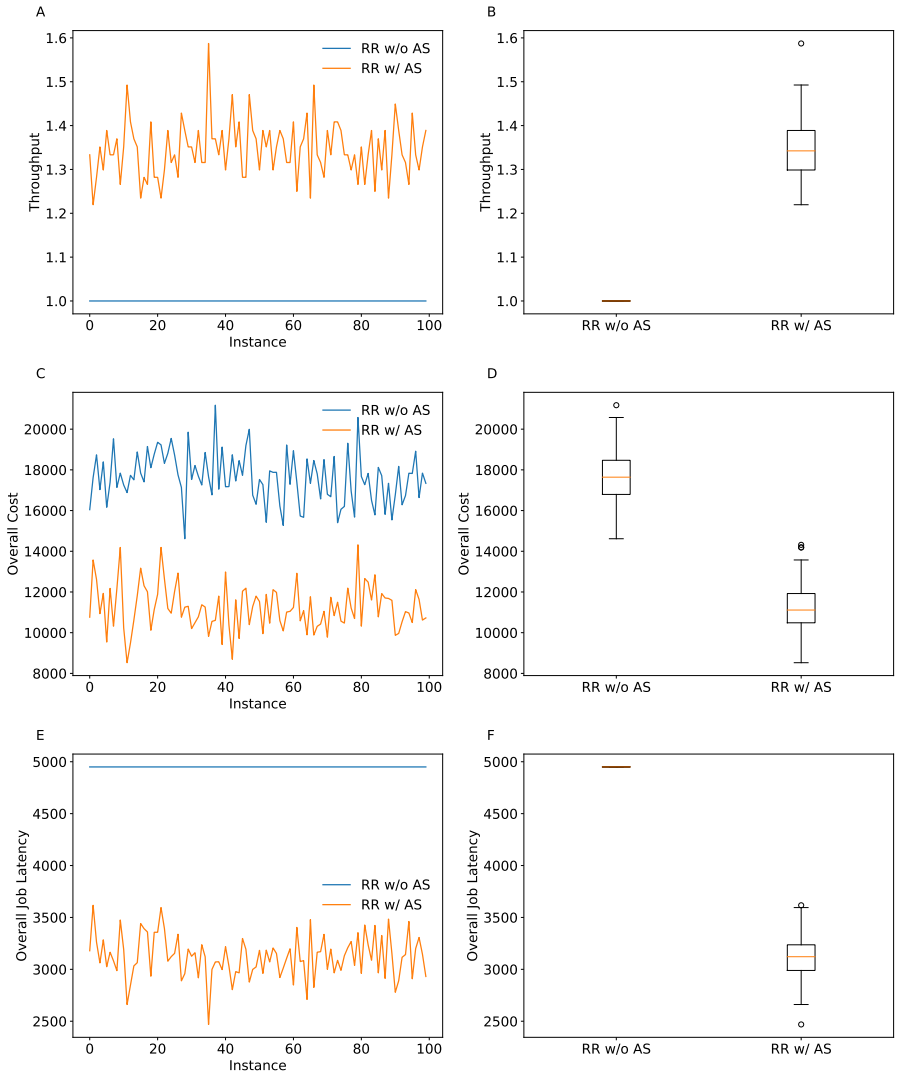

Figure 4: Step, Cost, and Job Latency comparison with Round-Robin method with and without action space for 100 SBs per batch.

Sub-figures (A) and (B) show overall throughput to complete each batch are constantly 1.0 for the Round-Robin without action space, and are 1.34, 1.37 and 1.39 on average for the Round-Robin with action space. In Sub-figure (A), the overall throughput for Round-Robin with action space range from 1.22-1.59, 1.29-1.46 and 1.34-1.45, respectively. On average, Round-Robin with action space reduces the overall throughput of Round-Robin without action space by

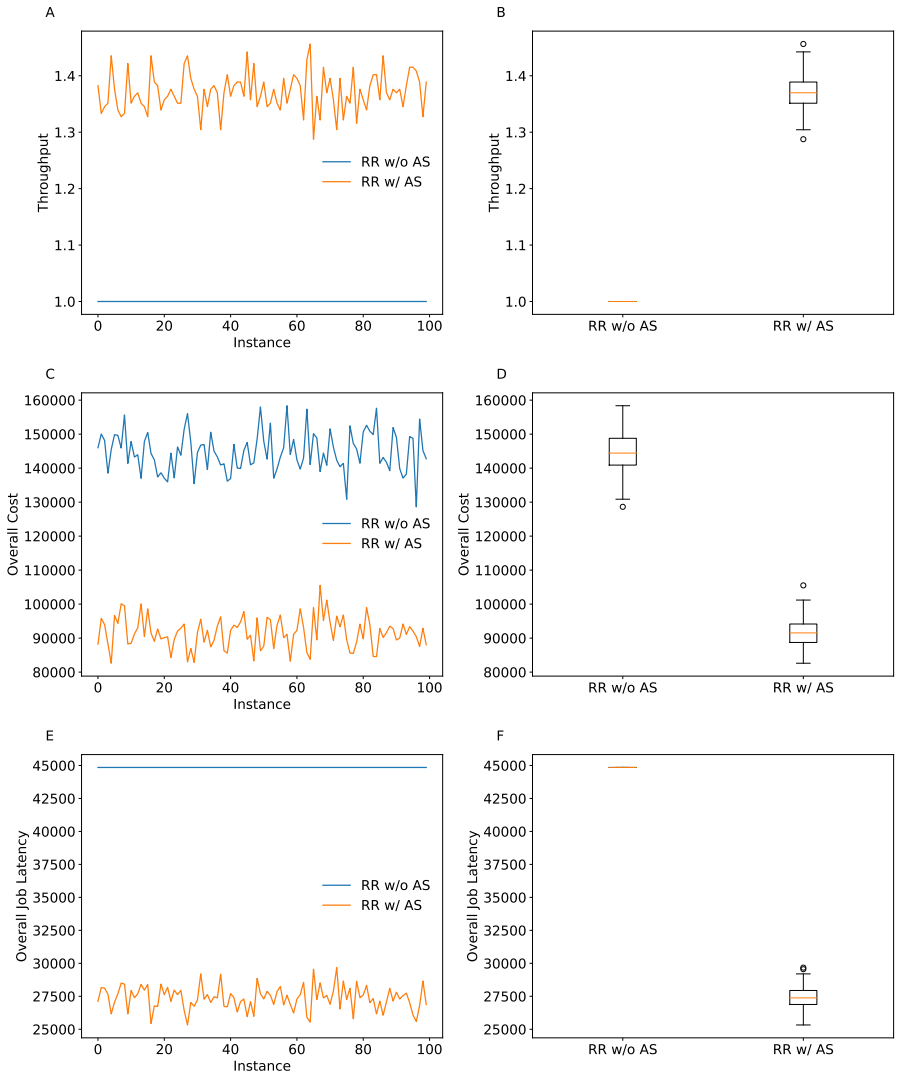

Figure 5: Step, Cost, and Job Latency comparison with Round-Robin method with and without action space for 300 SBs per batch.

36.7% on average.

Sub-figures (C) and (D) show the average overall cost is around 17.5k, 145k, and 1.5M for the Round-Robin without action space, and are 10.9k, 91.5k and 960k on average on average for the Round-Robin with action space. In Sub-figure (C), the overall throughput for Round-Robin without action space range from from 14616-21177, 128624-158361 and 1421392-1621711, respectively and

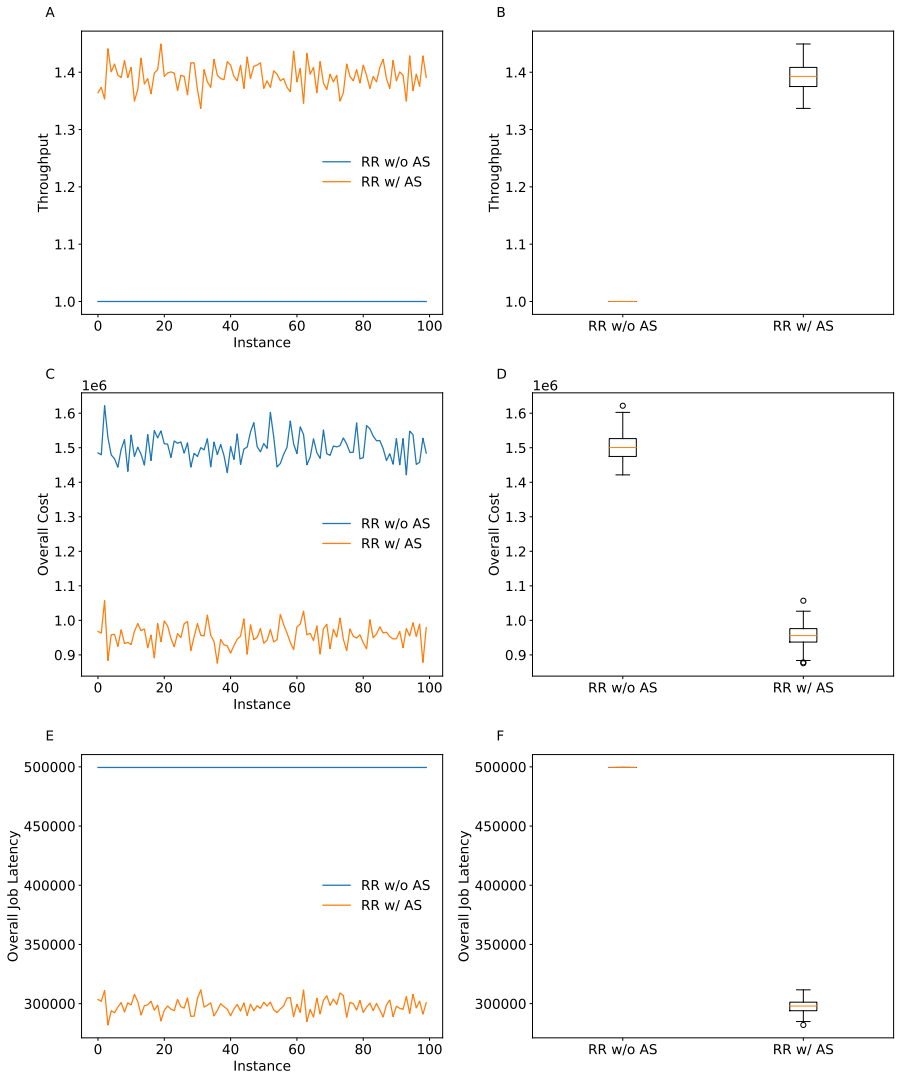

Figure 6: Step, Cost, and Job Latency comparison with RoundRobin method with and without action space for 1000 SBs per batch.

are range from 8525-14316, 82610-105511 and 875826-1056889 with action space. On average, Round-Robin with action space reduces the overall cost of Round-Robin without action space by 36.9% on average.

Sub-figures (E) and (F) show overall job latency are constantly 4.9k, 45k and 500k, respectively, for the Round-Robin without action space. Whereas the method with action space has the average job latency of 3.1k, 27.4k and 299k. In Sub-figure (E), the overall throughput for Round-Robin with action space range from 2469-3617, 25326-29673 and 281978-311656, respectively. On average, Round-Robin with action space reduces the overall cost of Round-Robin without action space by 39.1% on average.

Next, we use the Size-Prioritize policy to select actions with a maximum number of parallelizable SBs. For instance, there are two actions where the first has 4 SAs: SA-1, SA-23, SA-4, SA-5, and the second has 3 SAs: SA-13, SA-24, SA-5, the Size-Prioritize policy will favor the first action since it covers more SAs than the second action. The Cost-Prioritize policy, is similar to the Size-Prioritize method but instead of finding maximum SB instances in a single action, it computes the summed weight for each option at each time step and selects the one with the highest summed weight. We repeated the same process 100 times with 1000, 2000 and 4000 SBs, respectively, using identical random seeds as the Round-Robin method. The experimental results are shown in Figures 7, 8 and 9.

By looking at the results of the Size-Prioritize policy and RL policy in sub-figures (A) and (B) of Figures 7, 8, and 9, the RL method achieves similar throughput as the Size-Prioritize policy, and sometimes outputs higher overall throughput than the Size-Prioritize method. It is because of the action taken in some episodes with ties not optimal. One simple example is there are 4 SAs: SA-23, SA-24, SA-3, and SA-145. With the Size-Prioritize method, both action SA-23+SA-145 and SA-3+SA-145 has 2 SAs. If it chooses the first one, it can be done in the second round, but if it chooses the second one, it has to finish in three rounds. The reason why RL outperforms Size-method is RL considers SA utilization in the policy to break ties in the corner cases with Size-Prioritize method.

Sub-figure (A) of the Figure 7, 8, and 9 is the plot of the throughput of each instance for each method. Cost-Prioritize, Size-Prioritize, and the RL method have similar performance. Sub-figure (B) of the Figure 7, 8, and 9 is the box plot of the throughput for each method. Using the Round-Robin w/o AS method as a baseline, the Round-Robin method has around 39.1% improvement. For the three non-RR methods, the RL method performs comparably to the Size-Prioritize method, and the Cost-Prioritize method has the lowest overall throughput. Size-Prioritize method has around 77.4% improvement, Cost-Prioritize has around 72.6% improvement, RL method has around 74.6% improvement compared to the baseline method.

Sub-figure (C) of the Figure 7, 8, and 9 shows the overall cost of each instance for each method. RR w/o AS has the highest overall cost. When comparing among RL, Size-Prioritize, and Cost-Prioritize methods, the RL method performs slightly below the Cost-Prioritize method, and the Size-Prioritize method

has the highest overall cost.

Sub-figure (D) of the Figure 7, 8, and 9 shows the box plot of the overall cost for each method. Compared to the Round-Robin w/o AS method, the Round-Robin method has around 35.2% improvement. For the three non-RR methods, the RL method performs slightly below the Cost-Prioritize method, and the Size-Prioritize method has the lowest overall cost. Size-Prioritize method has

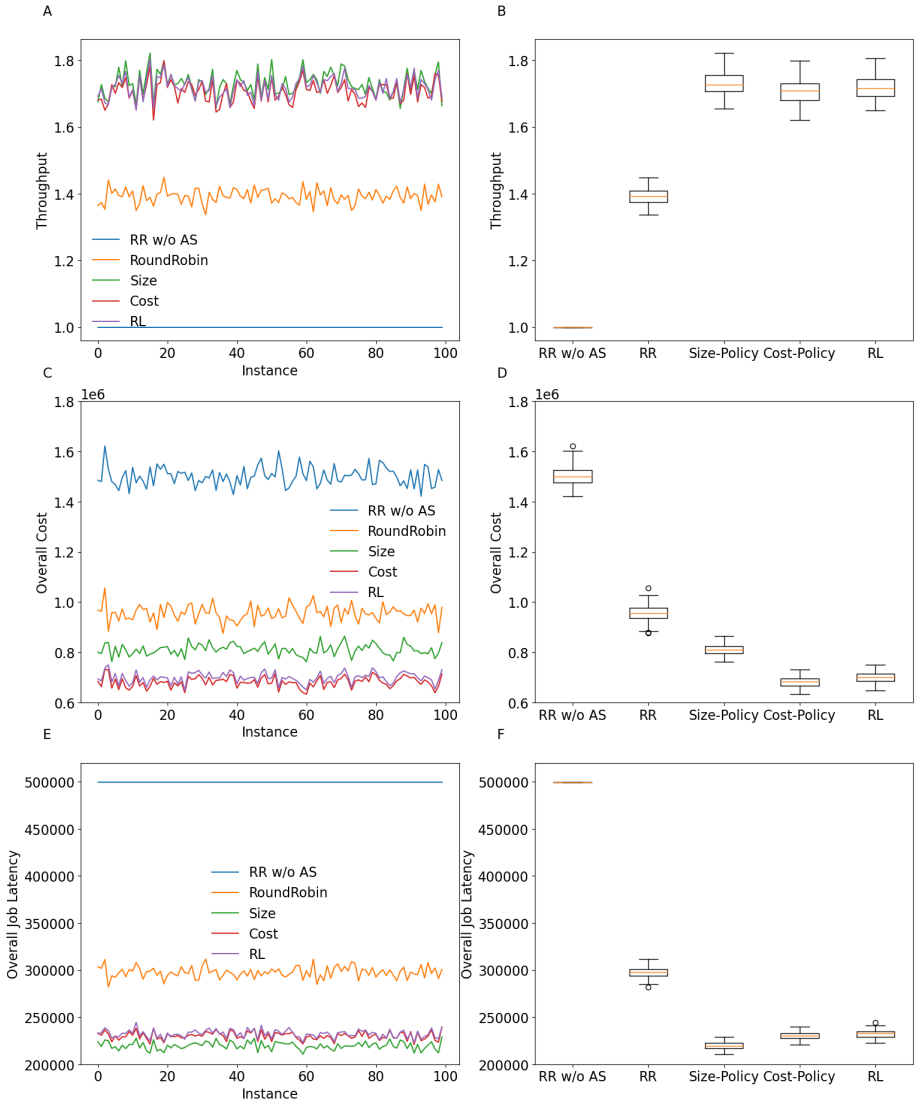

Figure 7: Cost, Steps, Job Latency comparison between methods using 1000 SBs per batch

around 43.6% improvement, Cost-Prioritize has around 53.6% improvement, RL method has around 52.4% improvement.

Sub-figure (E) of the Figure 7, 8, and 9 shows the overall job latency of each instance for each method. Sub-figure (F) of the Figure 7, 8, and 9 shows the box plot of the overall job latency for each method. Compared to the Round-Robin

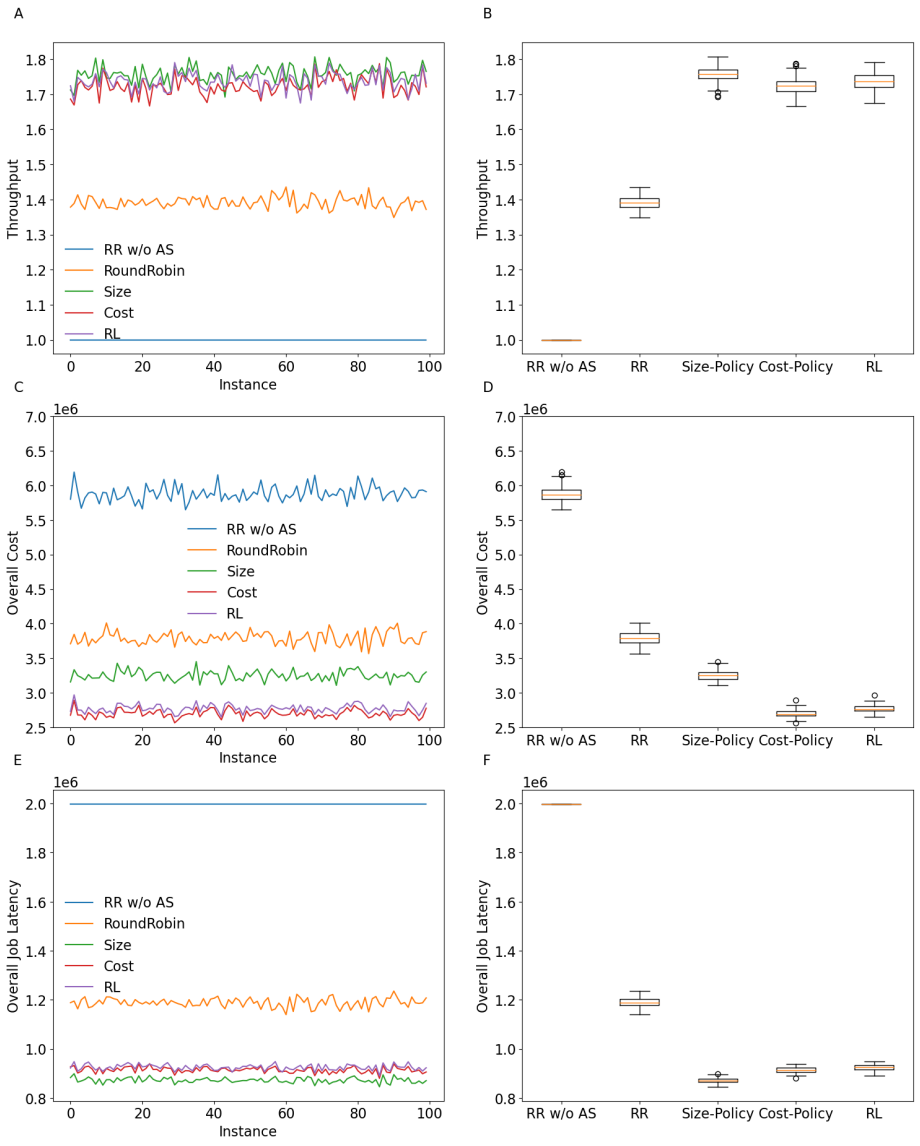

Figure 8: Cost, Steps, Job Latency comparison between methods using 2000 SBs per batch

w/o AS method, Round-Robin method has around 45.6% improvement, Size-Prioritize method has around 56.5% improvement, Cost-Prioritize has around 54.3% improvement, RL method has around 53.8% improvement.

In average improvement among three metrics: RL performs the best with 60.3% improvement compared to the baseline, the Cost-Prioritize method is

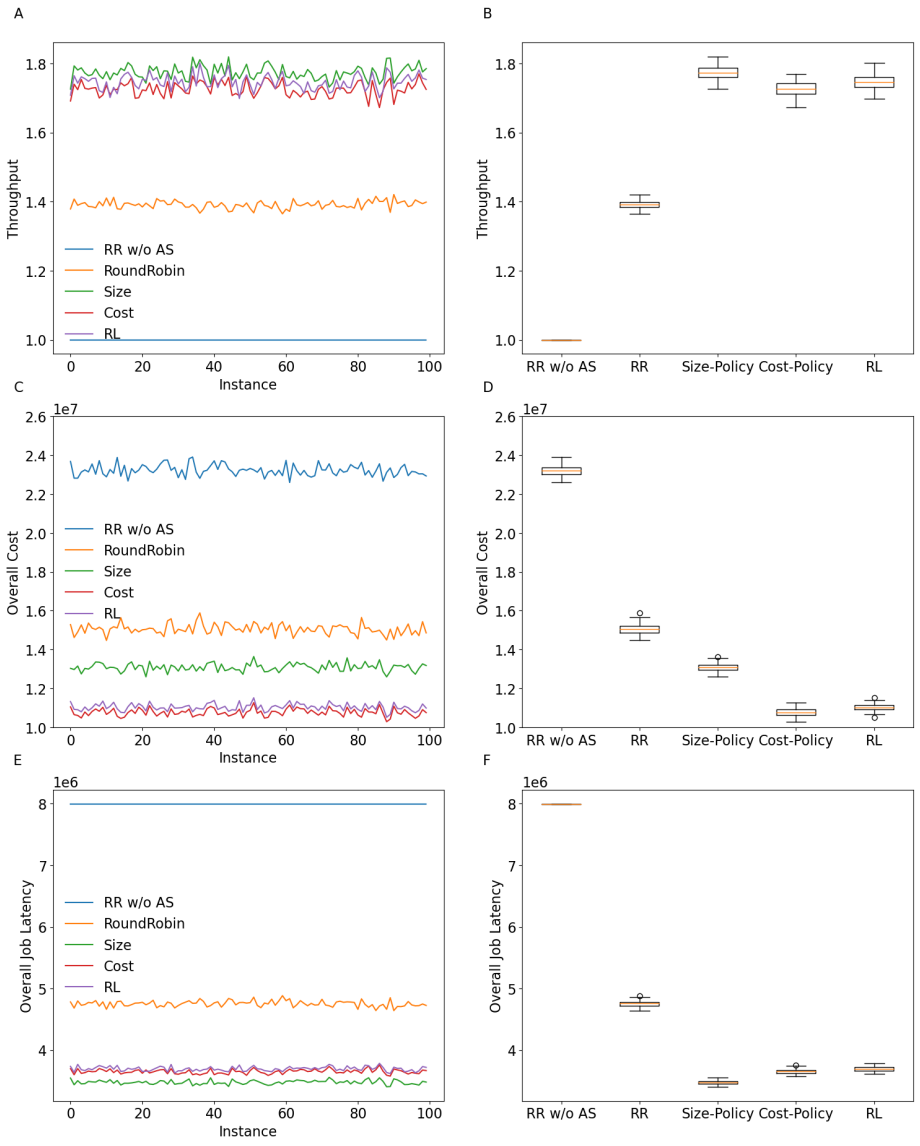

Figure 9: Cost, Steps, Job Latency comparison between methods using 4000 SBs per batch

0.1% below RL method, and the Size-Prioritize method's average improvement is 59.2%.

Figure 10 shows throughput, cost, and job-latency comparison when there are 100 SBs with 8 antenna subsets available. We see that the RL method achieved similar or near-optimal performance in all three metrics and overall performance is the best. The performance on throughput, cost, and job-latency is also consistent with the setting with 5 antenna subsets in Figure 7, 8, 9.

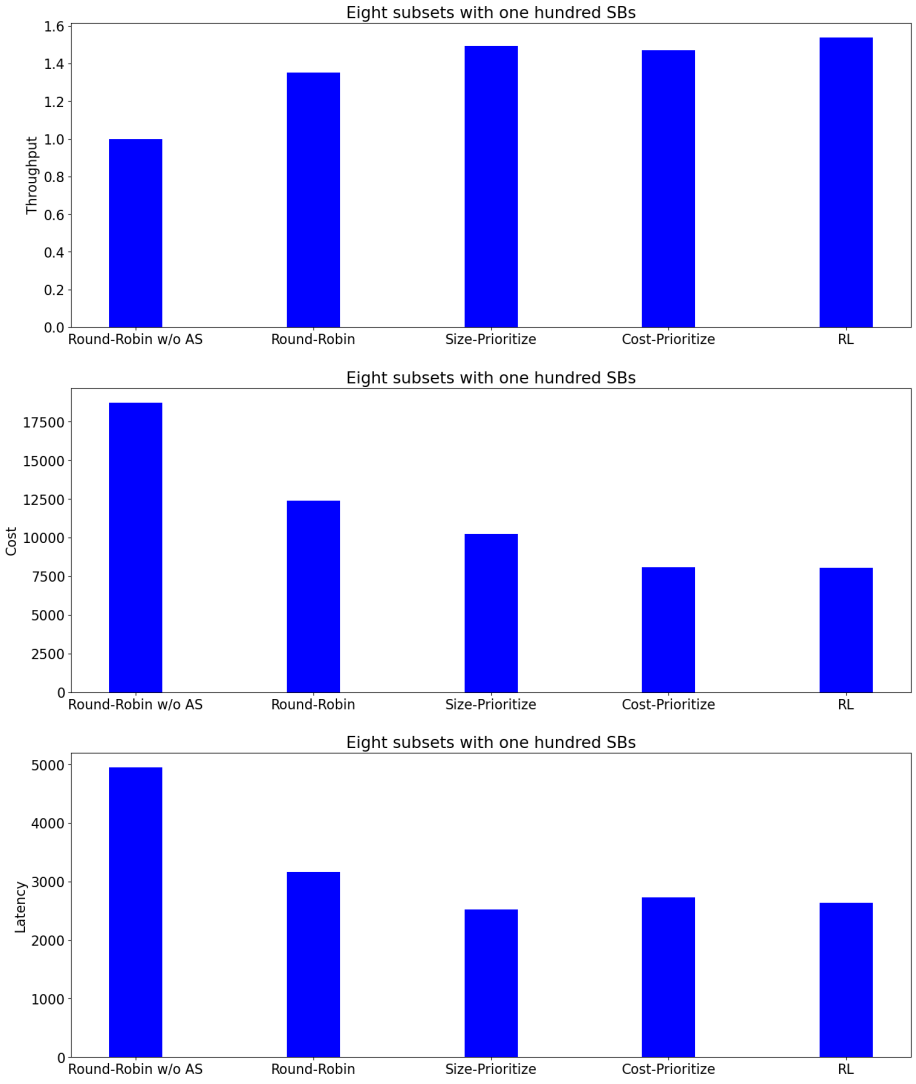

Figure 10: Cost, Steps, Job Latency comparison between methods using 100 SBs with 8 antenna subsets

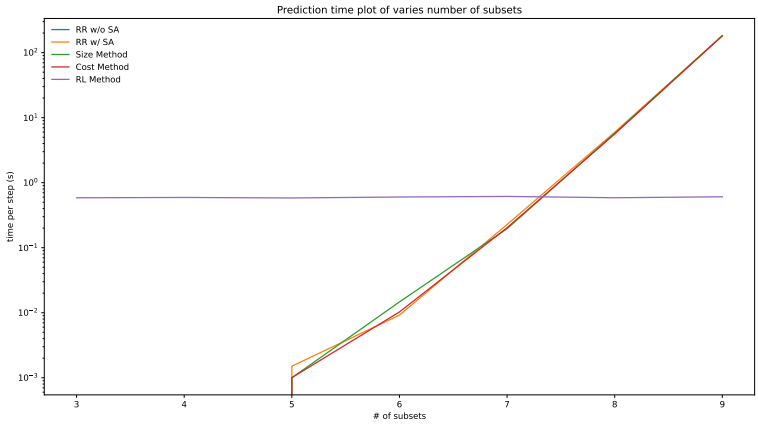

Figure 11: Execution time per step between methods from 3 antenna subsets to 9 antenna subsets

Figure 11 shows the scheduling time of different methods. We measure the time for predicting the decision as the scheduling time. The scheduling time of the RL method is constantly below 1 second while other methods grow as the number of antenna subsets increases. For scheduling time, we found that the conventional methods have lower scheduling time than the RL method when the number of antenna subsets is below 8, but starting with 8 or more antenna subsets, the RL method generates lower scheduling time than the other methods. The time reduction is better when number of antenna subset is large.However, the time taken to train the RL model is roughly 24 hours for 8 subsets scenario, if the number of antenna subsets is 9 or more, training time may take weeks or more.

5 Conclusion

In this work, we simulated the ngVLA scheduling scenario and designed the scheduling algorithms that fulfilled all three metrics: throughput, cost, and job-latency. We compared the difference between single SB and multiple SB execution performance. We also provided two conventional methods, the Size-Prioritize method and Cost-Prioritize, with near-optimal performance in cost and job latency. Because the weight attribute of SBs will affect the cost metric significantly, we leveraged the action space abstract and Priority Scheduling method to tackle the problem. From the experiment, action space grants the Round-Robin method with an average of 32% improvement for throughput, cost, and job latency over the one without. Using Round-Robin with action space as a baseline, the Size-Prioritize method has an average of 59.2% improvement,

the Cost-Prioritize method has an average of 60.2% improvement. Lastly, we designed an RL-based scheduling method that leverages the offline RL method to generate an action plan at each time step automatically and achieved the highest average of 60.3% improvement using throughput, cost and job-latency.

We also show that when scaling the number of antenna subsets to eight or larger, the RL method outperforms other methods in prediction time while maintaining the performance. The prediction time of the RL method is constant while prediction time of heuristic method grow exponentially. However, training time of the RL is growing exponentially when an increase in the number of antenna subsets, requires engineering a more efficient training method.

The future work includes but is not limited to searching antenna combinations for various observation needs, assuming that antenna subsets can be arranged in many more ways instead of being fixed. The scheduling algorithms can also account for the scenarios in which SBs have varied observation times or request arrival times are different as in scheduling problems in Computer Science.

In summary, we designed the two conventional methods: Cost-Prioritized method and the Size-Prioritize method to improve upon the DSA method for the ngVLA multi-scheduling and had great and near-optimal performance in the three metrics. The RL method we implemented has a higher average improvement over the conventional method and the performance is consistent with varying numbers of antenna subsets. We also show that the prediction time of the RL method shows an advantage over other methods when dealing with a setting of eight or more antenna subsets.

References

[1] Fawad Ahmad et al. "Shortest remaining processing time based schedulers for reduction of traffic congestion". In: *2013 International Conference on Connected Vehicles and Expo (ICCVE)*. IEEE. 2013, pp. 271–276.

[2] Kento Aida. "E ect of job size characteristics on job scheduling performance". In: *IPDPS 00/JSSPP 00: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, p. 117.

[3] Miklos Ajtai et al. "Fairness in scheduling". In: *Journal of Algorithms* 29.2 (1998), pp. 306–357.

[4] A Worldwide Collaboration ALMA. *Atacama large millimeter/submillimeter array*. 2020.

[5] Ahmad AlSa'deh and Adnan H Yahya. "Shortest remaining response time scheduling for improved web server performance". In: *Web Information Systems and Technologies: 4th International Conference, WEBIST 2008, Funchal, Madeira, Portugal, May 4-7, 2008, Revised Selected Papers 4*. Springer. 2009, pp. 80–92.

[6] Marcin Andrychowicz et al. “Hindsight experience replay”. In: *Advances in neural information processing systems* 30 (2017).

[7] Thomas Anthony, Zheng Tian, and David Barber. “Thinking fast and slow with deep learning and tree search”. In: *Advances in neural information processing systems* 30 (2017).

[8] M Emin Aydin and Ercan Öztemel. “Dynamic job-shop scheduling using reinforcement learning agents”. In: *Robotics and Autonomous Systems* 33.2-3 (2000), pp. 169–178.

[9] RC Balling Jr and RS Cerveny. “Cosmic ray flux impact on clouds? An analysis of radiosonde, cloud cover, and surface temperature records from the United States”. In: *Theoretical and applied climatology* 75 (2003), pp. 225–231.

[10] HS Behera et al. “A new proposed round robin with highest response ratio next (rrhrrn) scheduling algorithm for soft real time systems”. In: *International Journal of Engineering and Advanced Technology* 37 (2012), pp. 200–206.

[11] Marc G Bellemare, Will Dabney, and Rémi Munos. “A distributional perspective on reinforcement learning”. In: *International conference on machine learning*. PMLR. 2017, pp. 449–458.

[12] Dimitri Bertsekas et al. “Optimal short-term scheduling of large-scale power systems”. In: *IEEE Transactions on Automatic Control* 28.1 (1983), pp. 1–11.

[13] Ayan Bhunia. “Enhancing the performance of feedback scheduling”. In: *International Journal of Computer Applications* 18.4 (2011), pp. 11–16.

[14] Andrew Brzezinski. “Scheduling algorithms for throughput maximization in data networks”. PhD thesis. Massachusetts Institute of Technology, 2007.

[15] Ricardo Bustos et al. “Parque astronómico de atacama: An ideal site for millimeter, submillimeter, and mid-infrared astronomy”. In: *Publications of the Astronomical Society of the Pacific* 126.946 (2014), p. 1126.

[16] Bryan Butler. “Atmospheric Opacity at the VLA VLA Test Memo 232”. In: (2010).

[17] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*. Vol. 24. Springer Science & Business Media, 2011.

[18] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. “Longest wait first for broadcast scheduling”. In: *Approximation and Online Algorithms: 7th International Workshop, WAOA 2009, Copenhagen Denmark, September 10-11, 2009. Revised Papers 7*. Springer. 2010, pp. 62–74.

[19] Mu-Song Chen and Hao-Wei Yen. “Applications of machine learning approach on multi-queue message scheduling”. In: *Expert Systems with Applications* 38.4 (2011), pp. 3323–3335.

[20] D Clarke and J Avarias. "ALMA Scheduling: It's Dynamic!" In: *Astronomical Data Analysis Software and Systems XXI* 461 (2012), p. 177.

[21] Will Dabney et al. "Distributional reinforcement learning with quantile regression". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[22] Lawrence Davis. "Job shop scheduling with genetic algorithms". In: *Proceedings of the first International Conference on Genetic Algorithms and their Applications*. Psychology Press. 2014, pp. 136–140.

[23] Harvey M Deitel. *An introduction to operating systems*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[24] Hossam Fattah and Cyril Leung. "An overview of scheduling algorithms in wireless multimedia networks". In: *IEEE wireless communications* 9.5 (2002), pp. 76–83.

[25] Parviz Fattahi, Mohammad Saidi Mehrabad, and Fariborz Jolai. "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems". In: *Journal of intelligent manufacturing* 18 (2007), pp. 331–342.

[26] Vladimir Feinberg et al. "Model-based value estimation for efficient model-free reinforcement learning". In: *arXiv preprint arXiv:1803.00101* (2018).

[27] Scott Fujimoto and Shixiang Shane Gu. "A minimalist approach to offline reinforcement learning". In: *Advances in neural information processing systems* 34 (2021), pp. 20132–20145.

[28] Chengxi Gao, Victor CS Lee, and Keqin Li. "D-SRTF: Distributed shortest remaining time first scheduling for data center networks". In: *IEEE Transactions on Cloud Computing* 9.2 (2018), pp. 562–575.

[29] M. R. Garey, David S. Johnson, and Ravi Sethi. "The Complexity of Flowshop and Jobshop Scheduling". In: *Math. Oper. Res.* 1 (1976), pp. 117–129. URL: `https://api.semanticscholar.org/CorpusID:207233771`.

[30] Antonio Garrido et al. "Heuristic methods for solving job-shop scheduling problems". In: *Proc. ECAI-2000 Workshop on New Results in Planning, Scheduling and Design (PuK2000)*. 2000, pp. 44–49.

[31] Mitsuo Gen, Yasuhiro Tsujimura, and Erika Kubota. "Solving job-shop scheduling problems by genetic algorithm". In: *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*. Vol. 2. IEEE. 1994, pp. 1577–1582.

[32] Dipak Ghosal et al. "A reinforcement learning based network scheduler for deadline-driven data transfers". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2019, pp. 1–6.

[33] Indrajeet Gupta, Madhu Sudan Kumar, and Prasanta K Jana. "Efficient workflow scheduling algorithm for cloud computing system: a dynamic priority-based approach". In: *Arabian Journal for Science and Engineering* 43.12 (2018), pp. 7945–7960.

[34] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[35] Maryam Hamayun and Hira Khurshid. “An optimized shortest job first scheduling algorithm for CPU scheduling”. In: *J. Appl. Environ. Biol. Sci* 5.12 (2015), pp. 42–46.

[36] Parameswaran Hariharan. *Basics of interferometry*. Elsevier, 2010.

[37] Raham Hashim Yosuf et al. “Scheduling Algorithm for Grid Computing Using Shortest Job First with Time Quantum.” In: *Intelligent Automation & Soft Computing* 31.1 (2022).

[38] Rafael Hiriart. “ngVLA Multi-subarray Scheduling Algorithm ngVLA Computing Memo# 9”. In: (2024).

[39] Kenneth E Hoganson. “Multi-core real-time scheduling in multilevel feedback queue with starvation mitigation (MLFQ-RT)”. In: *Proceedings of the ACMSE 2018 Conference*. 2018, pp. 1–6.

[40] Edwin SH Hou, Nirwan Ansari, and Hong Ren. “A genetic algorithm for multiprocessor scheduling”. In: *IEEE Transactions on Parallel and Distributed systems* 5.2 (1994), pp. 113–120.

[41] Qingyan Huang. “Model-based or model-free, a review of approaches in reinforcement learning”. In: *2020 International Conference on Computing and Data Science (CDS)*. IEEE. 2020, pp. 219–221.

[42] Altaf Hussain et al. “Investigation of Cloud Scheduling Algorithms for Resource Utilization Using CloudSim.” In: *Computing & Informatics* 38.3 (2019).

[43] Ibrahim Mahmood Ibrahim et al. “Task scheduling algorithms in cloud computing: A review”. In: *Turkish Journal of Computer and Mathematics Education (TURCOMAT)* 12.4 (2021), pp. 1041–1053.

[44] Irwan Irwan and Muhammad Zen. “Implementation of Food Ordering Application System Design in a Coffee Shop Using the Multilevel Feedback Queue Algorithm”. In: *Jurnal Info Sains: Informatika dan Sains* 13.03 (2023), pp. 895–902.

[45] Lutful Karim et al. “An efficient priority packet scheduling algorithm for wireless sensor network”. In: *2012 IEEE international conference on communications (ICC)*. IEEE. 2012, pp. 334–338.

[46] AV Karthick, E Ramaraj, and R Ganapathy Subramanian. “An efficient multi queue job scheduling for cloud computing”. In: *2014 World Congress on Computing and Communication Technologies*. IEEE. 2014, pp. 164–166.

[47] Behice Meltem Kayhan and Gokalp Yildiz. “Reinforcement learning applications to machine scheduling problems: a comprehensive literature review”. In: *Journal of Intelligent Manufacturing* (2021), pp. 1–25.

[48] Behice Meltem Kayhan and Gokalp Yildiz. "Reinforcement learning applications to machine scheduling problems: a comprehensive literature review". In: *Journal of Intelligent Manufacturing* 34.3 (2023), pp. 905–929.

[49] Kenneth I. Kellermann, Ellen N. Bouton, and Sierra S. Brandt. "The Very Large Array". In: 2020. URL: `https://api.semanticscholar.org/CorpusID:226437311`.

[50] Zena Hussain Khalil and Ameer Basim Abdulameer Alaasam. "Priority based dynamic round robin with intelligent time slice and highest response ratio next algorithm for soft real time system". In: *Global Journal of Advanced Engineering Technologies* 2.3 (2013), pp. 120–124.

[51] Gyoung H Kim and CS George Lee. "Genetic reinforcement learning for scheduling heterogeneous machines". In: *Proceedings of IEEE International Conference on Robotics and Automation*. Vol. 3. IEEE. 1996, pp. 2798–2803.

[52] RNDSS Kiran, Polinati Vinod Babu, and BB Murali Krishna. "Optimizing CPU scheduling for real time applications using mean-difference round robin (MDRR) algorithm". In: *ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India-Vol I: Hosted by CSI Vishakapatnam Chapter*. Springer. 2014, pp. 713–721.

[53] Mohit Kumar and Subhash Chander Sharma. "Priority Aware Longest Job First (PA-LJF) algorithm for utilization of the resource in cloud environment". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE. 2016, pp. 415–420.

[54] Purba Daru Kusuma. "Truck Scheduling Model in the Cross-docking Terminal by using Multi-agent System and Shortest Remaining Time Algorithm". In: *International Journal of Advanced Computer Science and Applications* 12.7 (2021).

[55] Mark Lacy et al. "The Karl G. Jansky very large array sky survey (VLASS). Science case and survey design". In: *Publications of the Astronomical Society of the Pacific* 132.1009 (2020), p. 035001.

[56] Rohaya Latip and Zulkhairi Idris. "Highest response ratio next (HRRN) vs first come first served (FCFS) scheduling algorithm in grid environment". In: *Software Engineering and Computer Systems: Second International Conference, ICSECS 2011, Kuantan, Pahang, Malaysia, June 27-29, 2011, Proceedings, Part II 2*. Springer. 2011, pp. 688–693.

[57] Eugene L Lawler et al. "Sequencing and scheduling: Algorithms and complexity". In: *Handbooks in operations research and management science* 4 (1993), pp. 445–522.

[58] Hyunsung Lee et al. "Panda: Reinforcement learning-based priority assignment for multi-processor real-time scheduling". In: *IEEE Access* 8 (2020), pp. 185570–185583.

[59] Ricardo M Lima, Ignacio E Grossmann, and Yu Jiao. "Long-term scheduling of a single-unit multi-product continuous process to manufacture high performance glass". In: *Computers & chemical engineering* 35.3 (2011), pp. 554–574.

[60] Chung Laung Liu and James W Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment". In: *Journal of the ACM (JACM)* 20.1 (1973), pp. 46–61.

[61] Ji Liu, Esther Pacitti, and Patrick Valduriez. "A survey of scheduling frameworks in big data systems". In: *International Journal of Cloud Computing* 7.2 (2018), pp. 103–128.

[62] Zhen Liu and Philippe Nain. "Optimal scheduling in some multi-queue single-server systems". PhD thesis. INRIA, 1989.

[63] Robert Love. "The linux process scheduler". In: *Inform IT* (2003).

[64] Fan-Ming Luo et al. "A survey on model-based reinforcement learning". In: *Science China Information Sciences* 67.2 (2024), p. 121101.

[65] Mark McKinnon et al. "ngvla: The next generation very large array". In: *Bulletin of the American Astronomical Society* 51.7 (2019), p. 81.

[66] Carlos A Méndez et al. "State-of-the-art review of optimization methods for short-term scheduling of batch processes". In: *Computers & chemical engineering* 30.6-7 (2006), pp. 913–946.

[67] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[68] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[69] Arezou Mohammadi and Selim G Akl. "Scheduling algorithms for real-time systems". In: *School of Computing Queens University, Tech. Rep* (2005).

[70] Gustaaf van Moorsel. "The very large array after the upgrade". In: *2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS)*. IEEE. 2014, pp. 1–3.

[71] Anusha Nagabandi et al. "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 7559–7566.

[72] Peter J Napier et al. "The very long baseline array". In: *Proceedings of the IEEE* 82.5 (1994), pp. 658–672.

[73] Kashif Nisar et al. "A comprehensive survey on scheduler for VoIP over WLAN". In: *Journal of Network and Computer Applications* 36.2 (2013), pp. 933–948.

[74] Byung Joo Park, Hyung Rim Choi, and Hyun Soo Kim. "A hybrid genetic algorithm for the job shop scheduling problems". In: *Computers & industrial engineering* 45.4 (2003), pp. 597–613.

[75] Carlos D Paternina-Arboleda and Tapas K Das. "Intelligent dynamic control policies for serial production lines". In: *Iie Transactions* 33.1 (2001), pp. 65–77.

[76] Pandaba Pradhan, Prafulla Ku Behera, and BNB Ray. "Modified round robin algorithm for resource allocation in cloud computing". In: *Procedia Computer Science* 85 (2016), pp. 878–890.

[77] Tri Dharma Putra. "Analysis of Preemptive Shortest Job First (SJF) Algorithm in CPU Scheduling". In: *International Journal of Advanced Research in Computer and Communication Engineering* 9.4 (2020), pp. 41–45.

[78] Sébastien Racanière et al. "Imagination-augmented agents for deep reinforcement learning". In: *Advances in neural information processing systems* 30 (2017).

[79] Mohd Rahul. "A comparative evaluation of classless routing protocols (EIGRP) and classful routing protocols (RIP)". In: *Asian Journal of Technology & Management Research [ISSN: 2249–0892]* 4.01 (2014).

[80] Rasmus V Rasmussen and Michael A Trick. "Round robin scheduling–a survey". In: *European Journal of Operational Research* 188.3 (2008), pp. 617–636.

[81] David Raz, Benjamin Avi-Itzhak, and Hanoch Levy. "Fairness considerations of scheduling in multi-server and multi-queue systems". In: *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*. 2006, 39–es.

[82] Moses Reuven and Yair Wiseman. "Medium-term scheduler as a solution for the thrashing effect". In: *The Computer Journal* 49.3 (2006), pp. 297–309.

[83] David Rey and Hillel Bar-Gera. "Long-term scheduling for road network disaster recovery". In: *International journal of disaster risk reduction* 42 (2020), p. 101353.

[84] Simone Riedmiller and Martin Riedmiller. "A neural reinforcement learning approach to learn local dispatching policies in production scheduling". In: *IJCAI*. Vol. 2. Citeseer. 1999, pp. 764–771.

[85] Viviana Rosero et al. "The ngVLA Reference Array Configuration". In: *American Astronomical Society Meeting Abstracts# 233*. Vol. 233. 2019, pp. 361–07.

[86] Jia Ru and Jacky Keung. "An empirical investigation on the simulation of priority and shortest-job-first scheduling for cloud-based software systems". In: *2013 22nd Australian Software Engineering Conference*. IEEE. 2013, pp. 78–87.

[87] Norman Sadeh and Mark S Fox. "Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem". In: *Artificial intelligence* 86.1 (1996), pp. 1–41.

[88] Vahid Alizadeh Sahzabi et al. "Longest jobs first algorithm in solving job shop scheduling using adaptive genetic algorithm (GA)". In: *Fourth International Conference on Machine Vision (ICMV 2011): Machine Vision, Image Processing, and Pattern Analysis*. Vol. 8349. SPIE. 2012, pp. 762–767.

[89] Ibrahim Saidu et al. "A load-aware weighted round-robin algorithm for IEEE 802.16 networks". In: *EURASIP Journal on Wireless Communications and Networking* 2014 (2014), pp. 1–12.

[90] Pinal Salot. "A survey of various scheduling algorithm in cloud computing environment". In: *International Journal of Research in Engineering and Technology* 2.2 (2013), pp. 131–135.

[91] Rakesh Kumar Sanodiya et al. "A Highest Response Ratio Next (HRRN) Algorithm Based Load Balancing Policy For Cloud Computing". In: *International Journal of Computer Science Trends and Technology* 3.1 (2013), pp. 72–76.

[92] Ulrich Schmid and Johann Blieberger. "Some investigations on FCFS scheduling in hard real time applications". In: *Journal of Computer and System Sciences* 45.3 (1992), pp. 493–512.

[93] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[94] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[95] Seetharami Seelam et al. "Enhancements to linux i/o scheduling". In: *Proc. of the Linux symposium*. Vol. 2. 2005, pp. 175–192.

[96] Robert Selina et al. "Science with an ngVLA: the ngVLA reference design". In: *arXiv preprint arXiv:1810.08197* (2018).

[97] Vijayshree Shinde and Seema C Biday. "An Efficient Scheduling Strategy for Overloaded Real Time System". In: *International Journal of Computer Applications* 975 (), p. 8887.

[98] David Silver et al. "Deterministic policy gradient algorithms". In: *International conference on machine learning*. Pmlr. 2014, pp. 387–395.

[99] David Silver et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815* (2017).

[100] Kyle Singer et al. "Priority scheduling for interactive applications". In: *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*. 2020, pp. 465–477.

[101] Maria Ulfah Siregar. "A new approach to CPU scheduling algorithm: genetic round robin". In: *International Journal of Computer Applications* 47.19 (2012).

[102] Ramasubbareddy Somula and R Sasikala. "Round robin with load degree: an algorithm for optimal cloudlet discovery in mobile cloud computing". In: *Scalable Computing: Practice and Experience* 19.1 (2018), pp. 39–52.

[103] Jeffrey R Spirn. "Multi-queue scheduling of two tasks". In: *Proceedings of the 1976 ACM SIGMETRICS conference on Computer performance modeling measurement and evaluation*. 1976, pp. 102–108.

[104] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12 (1999).

[105] Jinjun Tang, Yifan Yang, and Yong Qi. "A hybrid algorithm for urban transit schedule optimization". In: *Physica A: Statistical Mechanics and its Applications* 512 (2018), pp. 745–755.

[106] Jay Teraiya and Apurva Shah. "Analysis of dynamic and static scheduling algorithms in soft real-time system with its implementation". In: *Soft Computing: Theories and Applications: Proceedings of SoCTA 2018*. Springer. 2020, pp. 757–768.

[107] Malhar Thombare et al. "Efficient implementation of multilevel feedback queue scheduling". In: *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE. 2016, pp. 1950–1954.

[108] A Richard Thompson et al. "The very large array". In: *Astrophysical Journal Supplement Series, vol. 44, Oct. 1980, p. 151-167.* 44 (1980), pp. 151–167.

[109] Avi Timor et al. "Using underutilized CPU resources to enhance its reliability". In: *IEEE Transactions on Dependable and Secure Computing* 7.1 (2008), pp. 94–109.

[110] John C Turner. "A probability model for computer system overhead". In: *European Journal of Operational Research* 2.3 (1978), pp. 185–190.

[111] Shuang Wang et al. "Multi-queue request scheduling for profit maximization in IaaS clouds". In: *IEEE Transactions on Parallel and Distributed Systems* 32.11 (2021), pp. 2838–2851.

[112] Xinhua Wang, Xiaodong Wang, and Alexei Ashikhmin. "Long-term scheduling and power control for wirelessly powered cell-free IoT". In: *IEEE Internet of Things Journal* 8.1 (2020), pp. 332–344.

[113] Yuping Wang et al. "A new hybrid genetic algorithm for job shop scheduling problem". In: *Computers & Operations Research* 39.10 (2012), pp. 2291–2299.

[114] Xinzhou Wu, Rayadurgam Srikant, and James R Perkins. "Scheduling efficiency of distributed greedy scheduling algorithms in wireless networks". In: *IEEE transactions on mobile computing* 6.6 (2007), pp. 595–605.

[115] Yuming Xu et al. "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues". In: *Information Sciences* 270 (2014), pp. 255–287.

[116] Takeshi Yamada and Ryohei Nakano. "Genetic algorithms for job-shop scheduling problems". In: *Proceedings of modern heuristic for decision support* 6781 (1997).

[117] Rui Yang et al. "Rethinking goal-conditioned supervised learning and its connection to offline rl". In: *arXiv preprint arXiv:2202.04478* (2022).

[118] Shengxiang Yang and Dingwei Wang. "A new adaptive neural network and heuristics hybrid approach for job-shop scheduling". In: *Computers & Operations Research* 28.10 (2001), pp. 955–971.

[119] Muhammad Zakria et al. "Cloud-fog based load balancing using shortest remaining time first optimization". In: *Advances on P2P, Parallel, Grid, Cloud and Internet Computing: Proceedings of the 13th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2018)*. Springer. 2019, pp. 199–211.

[120] Wei Zhang and Thomas G Dietterich. "A reinforcement learning approach to job-shop scheduling". In: *Ijcai*. Vol. 95. Citeseer. 1995, pp. 1114–1120.

[121] Wei Zhao and John A Stankovic. "Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems". In: *1989 Real-Time Systems Symposium*. IEEE Computer Society. 1989, pp. 156–157.