

Article

How to Dynamically Schedule Multiple Knowledge-Intensive Projects Under Uncertainty: An Approximate Dynamic Programming Approach

Hongbo Li ¹, Wei Chen ¹, Zehui Wei ^{1,*}, Qingkang Zhu ¹ and Xianchao Zhang ^{2,3,*}

¹ School of Management, Shanghai University, Shanghai 200444, China; hongbo_li@t.shu.edu.cn (H.L.); chenwei000918@163.com (W.C.); zqkang1994@163.com (Q.Z.)

² Provincial Key Laboratory of Multimodal Perceiving and Intelligent Systems, Jiaxing University, Jiaxing 314001, China

³ Engineering Research Center of Intelligent Human Health Situation Awareness of Zhejiang Province, Jiaxing University, Jiaxing 314001, China

* Correspondence: wz010608@shu.edu.cn (Z.W.); zhangxianchao@zjxu.edu.cn (X.Z.)

Highlights

Please indicate how your work links to systems science via your contributions to systems practice, theory, and or methodology.

- Modelling of multi-skilled multi-project scheduling under random arrivals and uncertain durations as a Markov decision process.
- Development of a rollout-based approximate dynamic programming algorithm with a reduced action space.

What are the main findings and/or the implications of the main findings?

- The proposed policy cuts total human resource cost by 3.1–16.7% compared with benchmark algorithms on medium/large instances, while completing more projects.
- Worker count and skill breadth interact, guiding workforce configuration decisions; the algorithm adapts online without retraining.

Abstract

Knowledge-intensive projects in research and development (R&D), software, and high-technology sectors are delivered by knowledge workers who each command several skills. When many such projects compete for a shared pool of multi-skilled workers, and neither project arrival times nor durations are known in advance, deciding who works on what at each moment becomes a sequential decision problem. Therefore, we propose the multi-skilled, multi-project dynamic scheduling problem with random project arrivals and uncertain durations and formulate it as a Markov decision process (MDP) that minimizes the total human resource cost. Since the cost-to-go function is computationally intractable, we develop a rollout-based approximate dynamic programming (ADP) algorithm that approximates it via Monte Carlo simulation embedded with a randomized base policy and restricts the action space to ten representative allocation policies. On benchmarks extended from the Project Scheduling Problem Library (PSPLIB), the proposed policy lowers the average total cost by 3.1% to 16.7% relative to Q-learning on medium- and large-scale instances while completing more projects with shorter delays; on small-scale instances Q-learning attains a lower nominal cost, revealing a trade-off among cost, completion rate, and delay.

Academic Editor: Ruben Ruiz-Torrobiano

Received: 8 June 2026

Revised: 14 July 2026

Accepted: 27 July 2026

Published: 1 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: knowledge-intensive projects; multi-skilled project scheduling; dynamic scheduling; Markov decision process; approximate dynamic programming; rollout

1. Introduction

The global economy is increasingly organized around projects: roughly one-fifth of all economic activity is now delivered in project form, accounting for some 12 trillion US dollars of value annually [1]. A growing share of this activity is knowledge-intensive, concentrated in research and development (R&D), software, and high-technology sectors, where competitive advantage rests on how well an organization deploys the expertise of its people rather than on physical capacity. Yet resources in such settings are frequently mismanaged, and an estimated 11.4% of project investment is wasted as a result [2]. These facts have kept project scheduling, which seeks to allocate resources so that cost and performance objectives are met under the relevant constraints [3,4], at the center of sustained academic attention. Done well, scheduling curbs the rushed deployment of people, trims idle capacity, and lowers coordination overhead, and it has been adopted across construction, manufacturing, information technology, and the coordination of standardization systems [5].

Because the work of these projects is intellectual, the same person can flexibly apply different competencies, and as technology advances, such knowledge workers increasingly hold several skills at once, so that multi-skilled human resources have become the organizing reality of modern project teams [6]; at the same time, firms routinely run many such projects side by side [7]. The central managerial question is therefore how to channel a limited pool of multi-skilled workers to the diverse skill needs of competing projects, a challenge that has motivated the multi-skilled multi-project scheduling problem (MSMPSP). At its core lies the optimal deployment of multi-skilled workers across projects so as to minimize the total cost of human resources.

Allowing workers to be multi-skilled and having many projects makes the MSMPSP substantially harder than its single-skill predecessor. A classical project-scheduling problem only asks when each activity should start; once resources can supply more than one skill, a second layer of decisions appears, namely, which worker's which skill should serve which project's requirement, and the problem remains NP-hard [8]. Consider a software team: if every programmer knows only C++, scheduling reduces to timing the activities, but once the team mixes C++ and Java specialists, with some engineers fluent in both, a schedule must first fix the technical approach and then assign people accordingly [9]. In a multi-project environment, the resulting sharing of and contention for multi-skilled workers across projects magnify both the scale and the difficulty. Early MSMPSP research largely assumed a deterministic world in which all project information is known at the outset, and within that frame a rich body of exact and (meta)heuristic algorithms has been developed [8,10–12].

In practice, however, knowledge-intensive project environments are anything but static. Projects do not all sit on the desk at time zero; they arrive one after another at random points, and their durations are equally uncertain [13,14]. A manager must therefore decide at each moment, given the current state of resources and demands, how to allocate workers, knowing that today's choice shapes not only the present but also the resources available in every later stage. Schedules computed from deterministic models tend to lose their grip once this uncertainty unfolds. Reinforcement learning and approximate dynamic programming (ADP) have consequently drawn growing interest in scheduling [15,16]: by casting the process as a Markov decision process (MDP) in which current choices are guided by estimates of future states, these methods are well suited to dynamic,

uncertain settings. Existing ADP work, however, has centered on resource-constrained or single-skill problems, and extending it to the multi-skilled multi-project setting is far from straightforward. In a single-skill problem, the resource side of the state reduces to a scalar or a low-dimensional vector of available capacities, and a decision only fixes which activities to start, so the number of candidate actions per stage remains moderate. Once workers command several skills and many projects compete for them, the state must track the remaining workload of every skill of every worker, so its dimension grows to the order of $S \times K$, where S is the number of skills and K is the number of employees, and a decision must simultaneously determine which projects to serve, which skill of which employee to deploy, and how much workload to assign, so the number of feasible allocation plans per stage grows exponentially in the numbers of skills and workers. This joint expansion of the state and action spaces renders single-skill ADP designs computationally intractable in our setting and calls for a dedicated treatment of both the cost-to-go approximation and the action space, which existing studies have not provided.

This paper takes up precisely that setting. The purpose of this study is threefold: to formally define and model the multi-skilled multi-project dynamic scheduling problem with uncertain project arrivals and durations (MSMPDSP), to develop a scalable online solution method for it, and to characterize when and why this method outperforms alternative scheduling policies. Treating both project arrivals and durations as uncertain, we seek a scheduling policy that allocates multi-skilled workers so as to minimize the total cost of human resources in a dynamic, uncertain environment, with the project completion rate and delay tracked as complementary evaluation indicators. In short, we ask how a portfolio of knowledge-intensive projects should be scheduled under uncertainty, and we answer with a rollout-based ADP policy. Our main contributions are summarized as follows:

- (1) First, we model the MSMPDSP as an MDP whose state is the available skill workload of each worker, whose decision is the resource-allocation plan, and whose objective is to minimize the total cost of human resources. A state-transition function captures the system's dynamic evolution and explicitly encodes the randomness of project arrivals, the heterogeneity of skill demands, and the scarcity of multi-skilled workers.
- (2) Second, we devise a rollout-based ADP algorithm for this model. It estimates the value of future-stage decisions by Monte Carlo simulation and refines the schedule through a base policy. To keep computation tractable, the action set is reduced to ten heuristic, priority-rule-based allocation policies, narrowing the per-stage search from an exponential number of candidates to a constant number and yielding a tractable approximation of the full action space; the optimality cost of this restriction is quantified empirically on small instances.
- (3) Third, we run extensive experiments on benchmark instances. Against Q-learning and several heuristic baselines across varied scenarios, the proposed algorithm achieves a lower total cost together with a higher completion rate on medium- and large-scale problems, whereas on small-scale problems it trades a higher nominal cost for a substantially higher completion rate and shorter delays; a sensitivity analysis additionally uncovers an interaction effect between the number of workers and the number of skill types, offering practical guidance for project-management decisions.

Figure 1 outlines the overall research framework of this study, proceeding from the dynamic scheduling problem under uncertainty, through the MDP formulation and the proposed rollout-based ADP algorithm, to the resulting computational results and managerial insights.

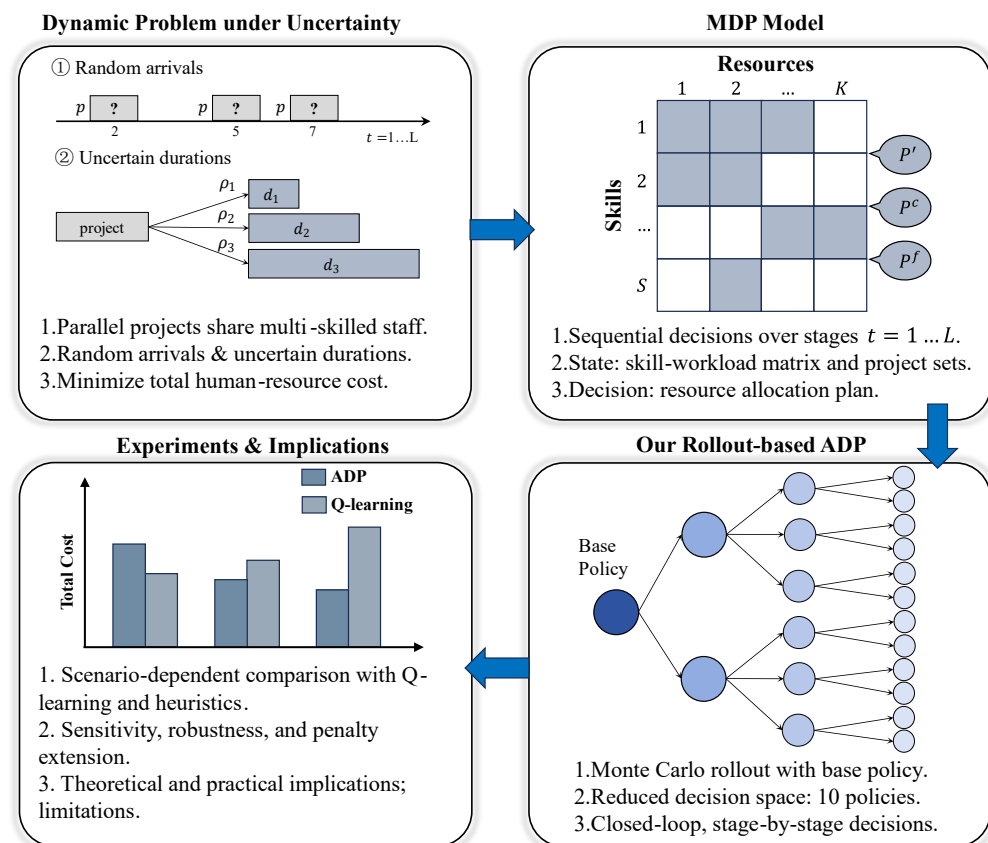


Figure 1. Overall research framework of this study. The question marks indicate the unknown worker-to-project skill assignments to be determined by the scheduling decision.

The four panels respectively summarize the problem setting, the sequential decision model, the solution methodology, and the empirical findings, providing a roadmap for the subsequent sections.

The remainder of this paper is organized as follows. Section 2 reviews the related literature. Section 3 describes the research problem and develops the Markov decision process model. Section 4 presents our rollout-based ADP algorithm in detail. Section 5 reports the computational experiment setup and analyzes the results. Section 6 discusses theoretical contributions, practical implications, limitations, and future research directions. Section 7 concludes the paper.

2. Literature Review

2.1. Multi-Project Scheduling Under Uncertainty

The multi-project scheduling problem under uncertainty aims to allocate limited resources to multiple parallel projects and to continuously refine the schedule as project information is progressively revealed over time [13]. From the perspective of the decision environment, existing research has mainly developed along three dimensions: the randomness of project arrivals, the uncertainty of activity durations, and the fluctuation of resource availability.

For the randomness of project arrivals, Melchioris [17] addressed dynamic and stochastic multi-project planning with randomly arriving projects, formulating the problem as a Markov decision process and developing approximate dynamic programming methods to obtain near-optimal scheduling policies. Satic et al. further extended this line of work by considering the dual randomness of both project arrival times and activity durations [18]. They proposed a simulation-based ADP algorithm and, through comparison

with heuristic policies, demonstrated the superiority of sequential decision-making methods in dynamic environments. Choi et al. addressed the dynamic selection and scheduling of new product development project portfolios, examining the reallocation of resources under uncertain project outcomes [19].

For the uncertainty of activity durations, Li et al. studied the stochastic resource leveling problem with flexible structures, formulating a chance-constrained programming model and designing a genetic algorithm to solve it [20,21]. Creemers characterized activity durations using phase-type distributions and, within a Markov decision process model, analyzed the optimal scheduling policy under precedence relationships [22]. Rostami et al. addressed the multi-mode multi-project scheduling problem under stochastic durations, proposing a scenario-based stochastic programming approach [23].

For the fluctuation of resource availability, Xie et al. studied the project scheduling problem under stochastically varying resource availability [16]. They proposed a rollout-based ADP algorithm and theoretically proved its sequential improvement property relative to the base policy. Lamas and Demeulemeester handled resource unavailability disturbances through a proactive scheduling policy, balancing the stability of the baseline schedule against the flexibility of its execution [24].

At the methodological level, Li and Womer modeled the resource-constrained stochastic project scheduling problem as a Markov decision process and designed a closed-loop ADP algorithm. They showed that sequential decision-making methods outperform traditional priority rules in minimizing the expected makespan [15]. This methodological paradigm has been widely extended in research on dynamic project management [25,26]. Although the studies above have advanced the field of multi-project scheduling under uncertainty, most existing research assumes a single resource type and does not address the sharing of and competition for multi-skilled human resources across multiple projects.

2.2. Multi-Skilled Project Scheduling Under Uncertainty

The MSMPSP is a class of NP-hard combinatorial optimization problems in project management [8]. Its core lies in allocating multi-skilled human resources to different activities across multiple projects so as to optimize a given management objective. Snauwaert and Vanhoucke provided a systematic classification of multi-skilled resource-constrained project scheduling problems, offering a unified research framework for the field [27].

In deterministic settings, the MSMPSP has been studied extensively. From a modeling perspective, Correia et al. proposed a mixed-integer linear programming (MILP) model [8], Heimerl and Kolisch developed an optimization model aimed at minimizing workforce cost [11], and Almeida et al. systematically compared the solution performance of several modeling frameworks [28]. From the perspective of the objective function, Correia and Saldanha-da-Gama were the first to introduce, in multi-skilled project scheduling, an objective structure that decomposes resource cost into a fixed component and a variable component [29]. From an algorithmic perspective, Li and Womer improved solution efficiency through a Benders decomposition algorithm that combines MILP and constraint programming (CP) [30]. Cai and Li, Attia et al., and Snauwaert and Vanhoucke solved the MSMPSP under different objectives using genetic algorithms [12,31,32]. Almeida et al. proposed a priority-rule-based heuristic framework [10]. Torba et al. [33] designed a hybrid genetic algorithm combined with simulated annealing for a practical setting in the semiconductor manufacturing industry. More recently, Snauwaert and Vanhoucke [34] proposed a unified solution framework for multi-skilled project scheduling problems with hierarchical skills, and Türk et al. [35] examined a cost-oriented multi-skilled project scheduling problem in the shipyard industry from tactical and operational perspectives through genetic-algorithm-based approaches.

By comparison, research on the MSMPSP under uncertainty remains relatively limited and has mainly focused on two aspects: duration uncertainty and changes in skill proficiency. Regarding duration uncertainty, Li et al. studied a software project scheduling problem with stochastic activity durations [36]. Similarly, Liu et al. investigated multi-skilled project scheduling for high-end equipment development, explicitly considering duration uncertainty alongside newcomer cultivation [37]. They formulated a scenario-based chance-constrained stochastic programming model that captures the effect of duration fluctuations on activity start times and total employee working hours. Regarding changes in skill proficiency, Chen et al. studied a multi-period technician scheduling problem with stochastic customer arrivals and experience-based service times, approximating the value function through a parametric piecewise-linear model [38]. Annear et al. addressed the dynamic assignment of multi-skilled workers in a shop-floor environment and proposed an ADP algorithm incorporating a lookahead rollout structure [39]. Bouajaja and Dridi reviewed methods in the field of human resource allocation and pointed out that the scheduling of multi-skilled resources in dynamic environments remains an open problem [40].

2.3. Reinforcement Learning and Metaheuristics for Dynamic Scheduling

In recent years, deep reinforcement learning (DRL) and learning-assisted metaheuristics have become prominent tools for dynamic scheduling problems in which decisions must respond to events revealed over time. Wang et al. [41] studied a dynamic decentralized resource-constrained multi-project scheduling problem with product transfers and random new project arrivals, and designed a DRL-based coordination mechanism that revises local schedules when new projects enter the system. Yu et al. [42] proposed a DRL approach for the dynamic job-shop scheduling problem with new job arrivals and time-varying conditions, showing that learned dispatching policies can adapt to arrival-driven disturbances. Hybrid designs that couple learning with metaheuristics have also emerged. Yang et al. [43] combined the non-dominated sorting genetic algorithm II (NSGA-II) with Q-learning for a bi-objective multi-mode resource-constrained multi-project scheduling problem, and Xu et al. [44] systematically compared genetic programming hyper-heuristics with reinforcement learning for learning scheduling rules, concluding that the two paradigms are complementary in dynamic environments.

These studies confirm the growing maturity of learning-based and hybrid methods for dynamic scheduling. Nevertheless, the DRL literature has concentrated on shop-floor structures or on project settings with anonymous, single-type resource capacities, whereas the recent multi-skilled scheduling literature remains largely deterministic or confined to a fixed project set. To the best of our knowledge, no existing learning-based or metaheuristic method simultaneously handles random project arrivals, uncertain durations, and the sharing of multi-skilled human resources across a dynamically evolving project portfolio, which is exactly the setting addressed in this paper. Moreover, DRL methods require an offline training phase whose learned policy may degrade when the arrival process shifts, whereas the rollout mechanism adopted in this paper requires no training and improves upon its base policy online.

In summary, this study differs from existing literature in the following three aspects. First, in terms of problem setting, prior research on the MSMPSP under uncertainty has mostly introduced random disturbances in durations or skill proficiency within a static framework where the set of projects is given in advance. In contrast, this paper simultaneously characterizes the randomness of project arrivals, the variability of durations, and the dynamic, multi-stage nature of the scheduling process, which more closely reflects the actual management scenarios faced by research-intensive enterprises in a continuous project-flow environment. Second, in terms of modeling perspective, existing studies have

mainly adopted chance-constrained programming or scenario-based programming for single-stage solutions, whereas this paper models the MSMPDSP as a Markov decision process. Here, the resource allocation at each stage is treated as a sequential decision, and the dynamic evolution of the system is characterized through a state transition function. Third, in terms of solution method, existing ADP methods for multi-skilled scheduling have mainly addressed single-project or deterministic project sets. This paper designs a Rollout-based ADP algorithm that approximates the expected future cumulative cost by combining Monte Carlo simulation with a base policy and controls the computational effort through a decision space consisting of ten representative policies. This method enables efficient solutions in large-scale dynamic multi-project environments. Unlike offline learning approaches, it makes each decision from the currently observed state without a separate training phase, which suits environments where the arrival process may vary over time.

3. Problem Statement and the Markov Decision Process Model

3.1. Problem Statement

The MSMPDSP is described as follows.

A. Planning horizons and project arrivals. Consider a multi-project management environment with a planning horizon of length L , divided into L stages of unit duration, where $t (t = 1, 2, \dots, L)$ denotes the stage index. Throughout the planning horizon, projects arrive successively over time. At the beginning of each stage t , at most one new project P_t arrives with a certain probability. Let P denote the set of all projects that arrive within the planning horizon. Each arriving project p requires a specific set of skills to be executed. The total workload demanded by project p for skill s is W_{ps} , which is determined upon the arrival of the project. Let S denote the set of skills involved. We assume that at most one new project arrives at the beginning of each stage. This assumption keeps the exogenous information process one-dimensional and the state transition compact, and it can be interpreted as the discrete-time counterpart of an arrival stream observed at a sufficiently fine time granularity, under which the probability of two or more arrivals within a single stage is negligible. The assumption does not limit the model structurally: if simultaneous arrivals are allowed, the arriving project P_t is replaced by an arriving set of projects, the exogenous information is extended accordingly, and the decision and state transition logic in Section 3.3 carries over unchanged, although the per-stage computational effort grows with the number of simultaneous arrivals.

B. Multi-skilled human resources. The skill requirements of projects are met by human resources possessing multiple skills. Let K denote the set of all available human resources. Employee $k \in K$ masters a set of skills $S_k \subseteq S$. Let K_s denote the set of human resources possessing skill $s \in S$. Even when two employees possess the same skill, their work capacities may differ. Let η_{sk} denote the workload that employee $k \in K_s$ can provide for skill s within one stage. Within a single stage, an employee can work using only one of the multiple skills they master, but may be assigned to multiple projects.

C. Uncertainty of project durations. During actual project execution, factors such as requirement changes, technical difficulties, and differences in employee capability may make the project duration $\tilde{d}_p$ difficult to specify precisely at the time of the scheduling decision. Following the approach of Li et al. [36], this paper characterizes duration uncertainty using a finite scenario set Ω_p . Each scenario $\omega \in \Omega_p$ corresponds to a possible realization of the duration of project p , denoted $d_p(\omega)$; that is, $d_p(\omega)$ is one possible value of the random variable $\tilde{d}_p$ under scenario ω . Scenario ω occurs with probability $p(\omega)$, and $\sum_{\omega \in \Omega_p} p(\omega) = 1$.

D. Stage decisions. At each stage t , the project manager faces the following decision situation. Having observed the projects that have arrived but not yet been scheduled, as well as the projects currently under execution, and given the status of the available human resources, the manager decides whether to allocate resources to a project awaiting scheduling so as to initiate its execution, and specifically which skills of which employees to assign to which projects. This decision requires a trade-off among the limited availability of multi-skilled resources, the competition for resources across projects, and the new projects that may arrive in the future. Specifically, allocating resources immediately to a project currently awaiting scheduling may lead to a resource shortage when more urgent projects emerge in subsequent stages. Conversely, deferring the allocation will postpone the completion time of the current project and may leave existing resources idle.

E. Cost structure and scheduling objective. Resource allocation incurs two types of costs. (a) Fixed cost C_k^f : the fixed charge incurred when employee k is used in a given stage. This charge is counted only once within the same stage and varies with the skill level of the employee. (b) Variable cost C_{sk}^v : the cost incurred per unit of workload that employee k performs for skill s in a unit of time, which likewise varies with the skill level. Note that C_{sk}^v is indexed by both the employee and the skill, so skill-dependent variable costs, which are common in knowledge-intensive projects where the same engineer may command different rates for coding and for testing, are directly accommodated by the model. The special case in which the variable cost of an employee is identical across skills, adopted in the illustrative example of Section 3.2 and in the experiments of Section 5, is a simplification of this general structure and requires no change to the model or the algorithm. Since the project duration depends on the resource allocation plan, assigning a more proficient employee may entail a higher per-unit-time cost but can shorten the duration and thereby reduce the number of stages over which costs accumulate. As a result, the scheduling decision involves a trade-off between cost and duration. The MSMPDSP studied in this paper aims to minimize the total cost of human resources by selecting an appropriate resource allocation plan at each stage within the planning horizon L . When the end of the planning horizon is reached, projects that have not yet been scheduled are carried over to the next planning horizon for further processing. We emphasize that the scheduling objective is confined to the resource usage costs defined above. The number of completed projects and the extent of project delays are not part of the objective function; they are reported in Section 5 as complementary evaluation indicators that reveal how a cost-oriented policy behaves along dimensions of managerial interest. An extension of the model that explicitly penalizes unscheduled and delayed projects is presented in Section 5.6.

3.2. Example

To illustrate the characteristics of the problem more intuitively, this section presents an example. The example involves three employees (HR_1, HR_2, HR_3), each of whom masters two skills, s_1 and s_2 , but at different proficiency levels. To simplify the example, we assume that, for a given employee, the variable cost is identical across skills. The cost parameters and skill workloads of the employees are shown in Table 1.

Table 1. Human resource parameters for the example.

Employee HR_k	Skill s_1 Workload	Skill s_2 Workload	Fixed Cost C_k^f	Variable Cost C_k^v
HR_1	2	3	100	180
HR_2	1	1	100	180
HR_3	3	3	90	170

Assume that the total number of stages in the planning horizon is $L = 5$. Within the planning horizon, two projects (project 1 and project 2) arrive in stage 1, project 3 arrives in stage 2, and no new projects arrive in the remaining stages. The detailed project information is shown in Table 2.

Table 2. Project parameters for the example.

Project P	Skill Demand s_1	Skill Demand s_2	Project Duration d_p
P_1	3	2	1
P_2	1	2	2
P_3	2	2	2

We now describe a feasible scheduling process based on the “maximum-skill-workload resource priority” policy. Under this policy, when allocating resources to a particular skill of a particular project, priority is given to the human resource with the largest workload for that skill in the current stage. Figure 2 illustrates the resource allocation results of the scheduling process described above.

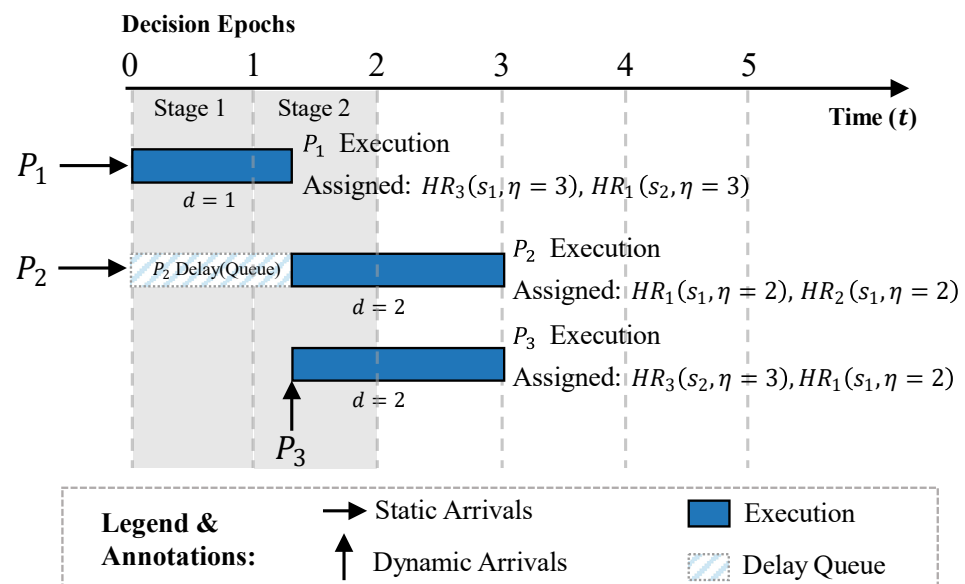


Figure 2. Gantt chart of the dynamic multi-skilled scheduling process for the example. Gray blocks denote idle (unallocated) employee capacity; dotted vertical lines mark stage boundaries.

In stage 1, project 1 and project 2 arrive. Resources are first allocated to project 1. Skill s_1 requires 3 units of workload, so according to the rule above, HR_3 (workload 3) is assigned to skill s_1 of project 1, exactly meeting the demand. Skill s_2 requires 2 units of workload, so HR_1 (workload 3) is assigned to skill s_2 . All skill demands of project 1 are thus satisfied, and project 1 begins execution. At this point, the remaining resources are insufficient to meet the demand of project 2, so project 2 is carried over to the next stage.

In stage 2, project 3 arrives. Since project 1 has a duration of one stage and was completed at the end of stage 1, all employees return to the available state. Project 2, which was carried over, is scheduled first. According to the allocation rule, its resource demand can be met, and project 2 begins execution. Project 3 is then examined; the remaining resources likewise satisfy its demand, so project 3 also begins execution.

In stage 3, no new projects arrive. Both project 2 and project 3 have a duration of two stages and are completed at the end of this stage. No new projects arrive in the subsequent

stages 4 and 5. Based on the employee assignments in each stage, the total cost of human resources can be calculated.

3.3. Markov Decision Process Model

In the MSMPDSP, the scheduling decision at each stage, namely which projects to start and how to allocate multi-skilled human resources, affects not only the current resource configuration but also the state of the available resources in subsequent stages. Therefore, unlike deterministic static scheduling, the solution to the MSMPDSP is not a fixed schedule but rather a scheduling policy π . At each decision point, the policy π dynamically determines the resource allocation plan according to the currently observed information about the system state.

Once the random variables governing project arrivals are fixed, the MSMPDSP reduces to the deterministic MSMPSP. Since the latter has been proven to be NP-hard [8], the MSMPDSP is also NP-hard.

3.3.1. Stage

Definition 1. A stage is a time point at which the decision maker makes a scheduling decision.

The planning horizon is divided into L stages of unit duration, with t denoting the stage index, $t = 1, \dots, L$. At the beginning of each stage t , the decision maker observes the current system state and makes a resource allocation decision a_t . This decision remains unchanged within stage t until the next decision is made at the beginning of stage $t + 1$. Stage $t = 1$ marks the starting point of scheduling, and stage L is the terminal point of the planning horizon.

3.3.2. States

The state S_t at stage t consists of two parts: the resource side and the project side. The resource-side state describes the availability of each human resource at stage t , denoted $R_t = \{[R_t^{sk}]_{|S| \times |K|}\}$, where R_t^{sk} represents the remaining workload of skill s of employee k at stage t . R_t can be represented by a matrix with $|S|$ rows and $|K|$ columns:

$$R_t = \begin{bmatrix} R_t^{11} & R_t^{12} & \dots & R_t^{1|K|} \\ R_t^{21} & R_t^{22} & \dots & R_t^{2|K|} \\ \vdots & \vdots & \ddots & \vdots \\ R_t^{|S|1} & R_t^{|S|2} & \dots & R_t^{|S||K|} \end{bmatrix}, \quad (1)$$

Each row of the matrix corresponds to a skill, and each column corresponds to a human resource. The initial state of the system is denoted $S_1 = \{[R_1^{sk}]_{|S| \times |K|}\}$, at which point the skill workloads of all employees are at their initial full levels.

The project-side state describes the execution status of all projects at stage t , including the set of projects that have arrived but have not yet been scheduled, P_t' ; the set of projects currently under execution, P_t^c ; and the set of completed projects, P_t^f . Each project $p \in P_t'$ is characterized by its skill demand workloads $W_{ps}, s \in S$. To formalize the tracking of active projects, each project $p \in P_t^c$ is associated with its remaining duration $d_{p,t}$ and with the resource allocation fixed at its start stage. When project p starts execution at stage t , its remaining duration is initialized as $d_{p,t} = d_p$, and at the end of each stage of execution it decreases by one, that is, $d_{p,t+1} = d_{p,t} - 1$. Once the remaining duration reaches zero, project p is removed from the executing set and added to the completed set, and the skill workloads it occupies are released back to the resource-side state through the term M_t in Equation (6). In summary, the complete state at stage t can be expressed as $S_t = (R_t, P_t', P_t^c, P_t^f)$, in which P_t^c records the pair $(p, d_{p,t})$ for every active project; this

state contains all the information required to make a decision at stage t and fully specifies the evolution of the projects under execution.

Corresponding to the state, the exogenous information ω_t in our problem refers to the random quantities at stage t that are not affected by the decision a_t . It consists of two parts: (a) whether a new project arrives at stage t and its attributes (the skill demand workload W_{ps}); and (b) if a new project arrives, the realized value of its duration d_p , drawn from the scenario set Ω_p with probability $p(\omega)$. The exogenous information ω_t is observed at the beginning of stage t and subsequently participates in the state transition from S_t to S_{t+1} .

3.3.3. Decisions

At stage t , let P'_t denote the set of projects awaiting scheduling, that is, the projects that have arrived but have not yet started execution. The project manager must decide whether to allocate resources to the projects in P'_t . The decision at stage t is denoted a_t and takes the form of a resource allocation plan $a_t = [x_{ptsk}]_{|P| \times |S| \times |K|}$, where x_{ptsk} represents the workload of skill s of employee k assigned to project p at stage t . The decision must satisfy the following constraints:

$$R_t^{sk} \geq \sum_{p \in P} x_{ptsk}, \forall s \in S, k \in K, t = 1, \dots, L, \quad (2)$$

$$\sum_{s \in S} z_{ks}^t \leq 1, \forall k \in K, \quad (3)$$

$$x_{ptsk} \geq 0, z_{ks}^t \in \{0, 1\}, \quad (4)$$

Constraint (2) ensures that, for each employee k , the total workload of skill s allocated across all projects does not exceed the employee's remaining workload. In constraint (3), z_{ks}^t indicates whether employee k uses skill s at stage t ; this constraint requires that each employee can use at most one skill within a single stage. When $z_{ks}^t = 0$ for all $s \in S$, employee k remains idle at stage t and incurs no cost, which occurs, for instance, when no project awaits scheduling or when deploying the employee is not cost-effective.

Constraint (4) defines the ranges of the decision variable and the auxiliary variable. When the remaining resources in a stage are insufficient to meet the execution requirements of any project, the default decision is taken, namely no project is scheduled.

Let A denote the decision space consisting of all possible resource allocation plans, and let $A_t \subseteq A$ denote the set of feasible resource allocation plans at stage t . Then $a_t \in A_t$.

3.3.4. State Transition Function

Let $T(\cdot)$ be the state transition function from the current stage t to the next stage $t + 1$:

$$S_{t+1} = T(S_t, a_t, \omega_t), \quad (5)$$

where ω_t is the exogenous random information at the current stage t , that is, whether a new project arrives and its attributes. The state S_{t+1} at stage $t + 1$ depends only on the current state S_t , the current decision a_t , and the exogenous information ω_t , thereby satisfying the Markov property.

The state transition equation can be expressed specifically as:

$$R_{t+1}^{sk} = R_t^{sk} - \sum_{p \in P'_t} x_{ptsk} + \sum_{m \in M_t} m, \forall k \in K, s \in S, \quad (6)$$

where $M_t = \{m_{ptsk} : s \in S, k \in K, \forall p \in P\}$ collects the skill workloads released at stage t by project completions; specifically, m_{ptsk} equals the workload of skill s of employee k occupied by project p if the remaining duration of p reaches zero at stage t , and equals zero otherwise. Equation (6) shows that the state update consists of three parts: the current remaining workload, the workload consumed by scheduling new projects (the subtractive term), and the workload released by project completions (the additive term).

The basic idea of the state transition is to update the state variables according to the decision made in the previous stage and the random information observed in the current stage. At stage t , the decision maker observes the current system state and makes a decision a_t , after which the projects continue execution. Upon entering the next stage $t + 1$, the decision maker observes the exogenous information ω_{t+1} (whether a new project arrives) and checks whether any project was completed in the previous stage and released its resources. The system state S_{t+1} is updated accordingly, and the decision a_{t+1} is then made. This process continues until the end of the planning horizon.

3.3.5. Cost Function

After the scheduling decision a_t is made at stage t , the human resource usage remains unchanged from stage t to stage $t + 1$ for all projects to which resources are allocated at stage t . Accordingly, we define the single-stage cost function $f(S_t, a_t, S_{t+1})$ to measure the human resource cost incurred by taking decision a_t in state S_t .

$$f(S_t, a_t, S_{t+1}) = \sum_{k \in K} \left(\sum_{s \in S} \left(C_{sk}^v \times \sum_{p \in P} x_{ptsk} \right) \right) + \sum_{k \in K} \left(H \left(\sum_{s \in S} \sum_{p \in P} x_{ptsk} \right) \times C_k^f \right), \quad (7)$$

where $H(x) = 1$ if $x > 0$ and $H(x) = 0$ if $x = 0$ is an indicator function. The first part of Equation (7) represents the variable cost, that is, the cost incurred by employees performing skills, while the second part represents the fixed cost. The indicator function $H(x)$ ensures that the fixed cost of the same employee is counted only once within a single stage. A smaller value of the cost function indicates that the resource allocation plan in that stage is better in terms of cost control. From a managerial standpoint, charging the fixed cost on a per-stage basis reflects the recurring coordination, communication, and administrative overhead that arises whenever an employee is mobilized in a period, such as stand-up meetings, progress reporting, and task handovers in knowledge-intensive organizations. This treatment follows the fixed and variable cost structure introduced for multi-skilled project scheduling by Correia and Saldanha-da-Gama [29]. Fixed costs tied to project onboarding or to whole contract durations can also be captured within this framework by charging the corresponding amount once at the project start stage, which requires only a reinterpretation of C_k^f and no structural change to the model.

3.3.6. Optimality Equation

Let $J_t(S_t)$ denote the expected minimum cumulative cost starting from stage t to the end of the planning horizon L , with state S_t as the starting point, and define the terminal value $J_L(S_L) = 0$. The optimality equation (the Bellman optimality equation) is then:

$$J_t(S_t) = \min_{a_t \in A_t} \{f(S_t, a_t, S_{t+1}) + E[J_{t+1}(T(S_t, a_t, \omega_t))]\}, \quad (8)$$

Solving the equation above yields the optimal decision $a_t(\pi^*)$ under the optimal decision sequence π^* . Equation (8) shows that the optimal decision $a_t(\pi^*)$ at stage t is the allocation plan in A_t that minimizes the sum of the immediate cost of the current stage and the cost-to-go function of the next stage. By letting t range from 1 to $L - 1$ and solving Equation (8) stage by stage, the optimal scheduling policy for each stage can be obtained. This is also the fundamental idea of dynamic programming.

However, owing to the NP-hard nature of the MSMPDSP and the presence of uncertainty, the dimensionality of the state space grows exponentially with the project scale. Solving Equation (8) directly is hampered by the curse of dimensionality in the cost-to-go function $J_{t+1}(S_{t+1})$. Therefore, an approximate dynamic programming algorithm needs to be designed to solve Equation (8) efficiently, which will be discussed in Section 4.

4. Rollout-Based ADP Algorithm

This section designs a Rollout-based ADP algorithm to efficiently solve the MDP model presented in Section 3.3.

Research on ADP originates from two fields: dynamic programming in operations research [45] and reinforcement learning in artificial intelligence [46]. A common strategy for designing an ADP algorithm is to replace the cost-to-go function $J_{t+1}(S_{t+1})$ in Equation (8) with an approximation term. This approximation term should be not only easy to compute but also as accurate as possible in estimating the value of the cost-to-go function [15,46]. The rollout policy is an effective way of constructing such an approximation term, and is particularly suitable for combinatorial optimization problems [47]. In a rollout policy, a set of pre-designed heuristic rules (referred to as the base policy) is used to compute an approximate value of $J_{t+1}(S_{t+1})$. In this sense, the rollout policy can be regarded as a heuristic version of the policy iteration algorithm in dynamic programming [48]. It has been successfully applied to resource-constrained project scheduling problems [16].

Our rollout-based ADP algorithm is specifically tailored to the particular structure of multi-skilled multi-project scheduling, with dedicated designs in the following three aspects. First, in terms of constructing the decision space, to address the issue that the decision space of the MSMPDSP grows exponentially with the number of skills and resources, we construct a finite decision space consisting of ten representative resource allocation policies, based on heuristic strategies and priority rules from the two dimensions of resource selection and project selection (see Section 4.2). This reduces the number of candidate decisions to be evaluated at each stage from exponential to constant order, enabling the algorithm to solve large-scale problems efficiently. Second, in terms of approximating the cost-to-go function, we approximate $J_{t+1}(S_{t+1})$ in Equation (8) through Monte Carlo simulation combined with a randomized base policy. For each candidate decision, the arrivals of projects in future stages are randomly generated according to the probability distribution of project arrivals, and a randomized base policy is used in the simulated future stages to quickly make allocation decisions, so as to estimate the future cumulative cost (see Section 4.1). This approach fully accounts for the influence of random project arrivals on subsequent resource states. Third, the algorithm operates in a closed-loop manner, making decisions at each stage based only on the currently observed information, which enables it to adapt to a dynamically changing project environment.

This section is organized as follows. Section 4.1 introduces how the Monte Carlo simulation-based rollout policy approximates the cost-to-go function. Section 4.2 describes the construction method of the decision space. Section 4.3 presents the overall framework of the algorithm.

4.1. Monte Carlo Simulation-Based Rollout Policy

This paper approximates the cost-to-go function in Equation (8) using Monte Carlo simulation combined with a base policy. At each stage t , for the candidate decisions $a_t \in A_t$, the rollout policy determines the approximate optimal decision $\tilde{a}_t$ as follows:

$$\tilde{a}_t = \arg \min_{a_t \in A_t} \{f(S_t, a_t, S_{t+1}) + E[\hat{J}_{t+1}^{MC}(S_{t+1})]\}, \quad (9)$$

where $E[\hat{J}_{t+1}^{MC}(S_{t+1})]$ is the approximate value of the cost-to-go function obtained through Monte Carlo simulation and the base policy. In each simulation, the base policy is used to

quickly make decisions in the subsequent stages from $t + 1$ to L , so as to estimate the cumulative cost of the future stages.

The computation of $E[\hat{J}_{t+1}^{MC}(S_{t+1})]$ proceeds as follows. For each candidate decision a_t at the current stage t , the immediate cost $f(S_t, a_t, S_{t+1})$ incurred by executing this decision is first computed. Then N Monte Carlo simulations are performed. In each simulation n , the arrivals of projects in the future stages from $t + 1$ to L are randomly generated according to the probability distribution of project arrivals, and at each simulated stage the base policy $h \in H$ is used to quickly make resource allocation decisions. The cost R_n incurred during the simulation is accumulated. After N simulations are completed, the evaluation value of candidate decision a_t is:

$$V(a_t) = f(S_t, a_t, S_{t+1}) + \frac{1}{N} \sum_{n=1}^N R_n, \quad (10)$$

In Equation (10), the first term is the immediate cost, and the second term is the average future cumulative cost over the N simulations. Finally, the candidate decision with the smallest evaluation value is selected as the actual decision for the current stage, that is, $\tilde{a}_t = \arg \min_{a_t \in A_t} V(a_t)$.

Regarding the choice of the base policy, we compared three base policies in preliminary experiments: (a) a randomized policy, which randomly selects a decision from the decision space at each simulated stage; (b) a round-robin policy, which selects decisions in turn according to a predefined order; and (c) a minimum-cost policy, which selects the decision with the lowest immediate cost. The results show that the randomized policy, through diversified exploration, can cover a broader range of the policy space and achieves a better balance in terms of total cost (see Section 5.3). Therefore, this paper adopts the randomized policy as the base policy.

The number of Monte Carlo simulations N affects the accuracy of the cost-to-go function estimation. A larger N yields a more accurate estimate but also increases the computation time accordingly. The experimental results in Section 5.3 will analyze the influence of different values of N on the performance of the algorithm. Algorithm 1 presents the pseudocode of the decision value estimation procedure.

Algorithm 1. Monte Carlo simulation-based decision value estimation

Input: current state S_t , feasible decision set A_t , set of projects awaiting scheduling P' , number of simulations N , set of base policies H

Output: approximate optimal decision a_t^*

```

1 For each  $a_t \in A_t$  do
2 If  $a_t$  can satisfy the resource demand of some project in  $P'$  then
3   Simulate the execution of decision  $a_t$ , and compute the immediate cost  $r_t \leftarrow f(S_t, a_t, S_{t+1})$ ;
4 Else
5    $r_t \leftarrow 0$ ;
6 End if
7  $R_{future} \leftarrow 0$ ;
8 For  $n = 1$  to  $N$  do
9   Save a state snapshot:  $S \leftarrow S_{t+1}$ ;  $P \leftarrow$  a copy of  $P'$ ;
10   $R_n \leftarrow 0$ ;
11  Select a base policy  $h$  from the set of base policies  $H$ ;
12  For  $\tau = t + 1$  to  $L$  do
13    Simulate by probability whether a new project arrives at stage  $\tau$ ;
14    If a new project arrives then update  $P$ ;
15    End if
16    Check and process completed projects, release resources, and update  $S$ ;
17    If  $P \neq \emptyset$  and a feasible allocation exists then

```

```

18     Select decision  $a_t$  according to base policy  $h$ ;
19      $R_n \leftarrow R_n + f(S, a_t, S')$ ;
20     Update state  $S$  and the project set;
21   End if
22 End for
23    $R_{future} \leftarrow R_{future} + R_n$ ;
24 End for
25  $V(a_t) \leftarrow r_t + R_{future} / N$ ;
26 End for
27  $a_t^* \leftarrow \operatorname{argmin}_{a_t \in A_t} V(a_t)$ ;
28 Return  $a_t^*$ 

```

4.2. Decision Space Construction

In the MSMPDSP, a decision corresponds to the specific allocation plan of multi-skilled human resources across projects. Since both the number of skills $|S|$ and the number of human resources $|K|$ are discrete parameters, the complete decision space expands rapidly as these parameters increase. At each stage t , if all possible resource allocation plans are enumerated without restriction, the number of candidate decisions grows exponentially. To enable our ADP algorithm to evaluate all candidate decisions within a reasonable time, it is necessary to reduce the decision space effectively.

To this end, based on different priority rules and resource allocation logics, we construct ten representative decisions that together form the decision space A of the algorithm. Each decision policy is designed along two dimensions: (a) the resource selection dimension, that is, which employee is given priority when allocating a particular skill; and (b) the project selection dimension, that is, which project's skill demand is given priority. Table 3 lists the definitions of the ten decision policies.

Table 3. Ten resource allocation policies in the decision space.

No.	Resource Selection Dimension	Project Selection Dimension	Policy Description
ac_1	Maximum skill workload	Default order	Give priority to the employee with the largest current skill workload
ac_2	Minimum skill workload	Default order	Give priority to the employee with the smallest current skill workload
ac_3	Maximum cost	Default order	Give priority to the employee with the highest cost
ac_4	Minimum cost	Default order	Give priority to the employee with the lowest cost
ac_5	Maximum total workload	Default order	Give priority to the employee with the largest total workload across all skills
ac_6	Minimum total workload	Default order	Give priority to the employee with the smallest total workload across all skills
ac_7	Maximum skill workload	Largest-demand project first	Select the project with the largest skill demand first, then assign the employee strongest in that skill
ac_8	Minimum skill workload	Largest-demand project first	Select the project with the largest skill demand first, then assign the employee weakest in that skill
ac_9	Maximum skill workload	Smallest-demand project first	Select the project with the smallest skill demand first, then assign the employee strongest in that skill
ac_{10}	Minimum skill workload	Smallest-demand project first	Select the project with the smallest skill demand first, then assign the employee weakest in that skill

The ten policies above are constructed as a structured, rather than arbitrary, sample of the allocation logics available to a project manager. Along the resource selection dimension, ac_1 to ac_6 exhaust the three natural ordering criteria for employees, namely single-

skill workload, cost, and total workload, each in ascending and descending directions, and thereby embody the logics of capability first (ac_1), load balancing (ac_2), high-value first (ac_3), cost saving (ac_4), comprehensive capability first (ac_5), and comprehensive balancing (ac_6). Along the project selection dimension, ac_7 to ac_{10} combine the two extreme project orderings, largest demand first and smallest demand first, with the two extreme resource orderings, so that the interaction between project urgency and resource strength is also represented. The set therefore covers both corner strategies and balanced strategies in the two-dimensional design space, while any additional rule would increase the per-stage evaluation effort linearly without introducing a qualitatively new logic.

Through this construction, the algorithm needs to evaluate only ten candidate decisions at each stage. Compared with enumerating all feasible resource allocation plans, whose number grows exponentially with S and K , this reduces the search to constant order while preserving diverse decision logics, and the rollout mechanism further compensates for the coarseness of any single rule by evaluating every candidate under simulated future uncertainty. It should be stressed, however, that the restricted decision space is a tractable approximation of the complete action space and does not guarantee global optimality; the magnitude of the resulting gap relative to full-action benchmarks is quantified empirically on small instances in Section 5.4.3. This design rationale is similar to the method of constructing base policies through priority rules in Li and Womer [15], but is extended to address the characteristics of multi-skilled multi-project scheduling.

4.3. Framework of Our ADP Algorithm

Based on the rollout policy and decision space described above, this section presents the overall framework of our ADP algorithm. Algorithm 2 shows its pseudocode. The algorithm takes as input the set of human resources K , the set of skills S , the cost parameters C_k^f and C_{sk}^v , the planning horizon L , the number of Monte Carlo simulations N , the decision space A , and the set of base policies H . It outputs the total cost C_{total} and the decision sequence π^* generated by the rollout policy.

The main procedure of the algorithm is as follows. First, the decision sequence π^* , the total cost C_{total} , the stage variable t , the state variable S_t , and the various project sets are initialized. The algorithm then executes stage by stage starting from stage $t = 1$. At the beginning of each stage t , the algorithm first checks whether a new project arrives, and if so, updates the project set P and the set of projects awaiting scheduling P' . Next, it checks whether any project under execution is completed in the current stage; if so, the resources occupied by the project are released and the state S_t is updated. After that, if there are projects awaiting scheduling and the resource conditions permit, the algorithm invokes the decision value estimation procedure in Algorithm 1 to select the approximate optimal decision a_t^* for the current stage, executes this decision, and updates the relevant variables. If no project can be scheduled in the current stage, the default decision is taken. At the end of each stage, the immediate cost is added to C_{total} , the decision a_t^* is appended to the decision sequence π^* , and the system is advanced to the next stage through the state transition function. After all stages are completed, the algorithm outputs the total cost C_{total} and the decision sequence π^* .

Algorithm 2. Rollout-based ADP algorithm

Input: set of human resources K , set of skills S , cost parameters C_k^f and C_{sk}^v , planning horizon L , number of Monte Carlo simulations N , decision space A , set of base policies H

Output: total cost C_{total} , decision sequence π^*

1 **Initialize:** $\pi^* \leftarrow \emptyset$; $C_{total} \leftarrow 0$;

2 **Initialize:** $t = 1$; $S_t = S_1$; $P \leftarrow \emptyset$; $P^f \leftarrow \emptyset$; $P^c \leftarrow \emptyset$; $P' \leftarrow \emptyset$;

3 **For** $t = 1$ **to** L **do**

```

4 If a new project  $P_t$  arrives in the current stage then
5    $P \leftarrow P \cup \{P_t\}; P' \leftarrow P' \cup \{P_t\};$ 
6 End if
7 For each  $p \in P^c$  do
8   If project  $p$  is completed in stage  $t$  then
9      $P^c \leftarrow P^c \setminus \{p\}; P^f \leftarrow P^f \cup \{p\};$ 
10    Release the resources occupied by project  $p$  and update state  $S_t$ ;
11  End if
12 End for
13 If  $P' \neq \emptyset$  and a feasible decision exists then
14    $a_t^* \leftarrow \text{DecisionValueEstimation}(S_t, A_t, P', N, H);$ 
15   Execute  $a_t^*$ : schedule the corresponding project  $p^*$  and allocate resources;
16    $r_t \leftarrow f(S_t, a_t^*, S_{t+1});$ 
17   Update:  $P^c \leftarrow P^c \cup \{p^*\}; P' \leftarrow P' \setminus \{p^*\};$ 
18 Else
19   Default decision  $a_t^* \leftarrow a^0$  (the empty allocation; no project is scheduled);  $r_t \leftarrow 0$ ;
20 End if
21  $C_{total} \leftarrow C_{total} + r_t$ ;
22  $\pi^* \leftarrow \pi^* \cup \{a_t^*\};$ 
23 State transition:  $S_{t+1} \leftarrow T(S_t, a_t^*, \omega_t);$ 
24 End for
25 Return  $C_{total}$  and  $\pi^*$ 

```

Several points regarding Algorithm 2 are noted as follows. First, since the random event corresponding to whether a project arrives is unknown until it occurs, the algorithm makes decisions at each stage based only on the currently observed information. Second, the algorithm does not enumerate all possible decision combinations in A_t , but instead restricts the evaluation scope to a manageable scale through the ten policies constructed in Section 4.2. Third, the algorithm can be used online; that is, during project execution, whenever a new project arrives or a project is completed, the algorithm runs to determine which projects should be scheduled in the current stage. Figure 3 depicts the flowchart of the proposed algorithm, in which the rollout decision evaluation is embedded within the stage-by-stage loop, and the MDP state is updated as projects arrive and complete.

In addition, the efficiency advantage of the proposed algorithm is reflected in two aspects. On the one hand, the reduction in the decision space means that only ten candidate decisions need to be evaluated at each stage. On the other hand, the base policy used in the Monte Carlo simulation is simple to compute, so the overhead of each simulation is low. Therefore, the algorithm can solve large-scale problems within a reasonable computation time, which will be verified in the experiments in Section 5.

Finally, it should be noted that π^* is optimal only with respect to the restricted decision space A and the simulation-based approximation of the cost-to-go function; it is not guaranteed to coincide with the global optimum over the complete action space, a point examined empirically in Section 5.4.3.

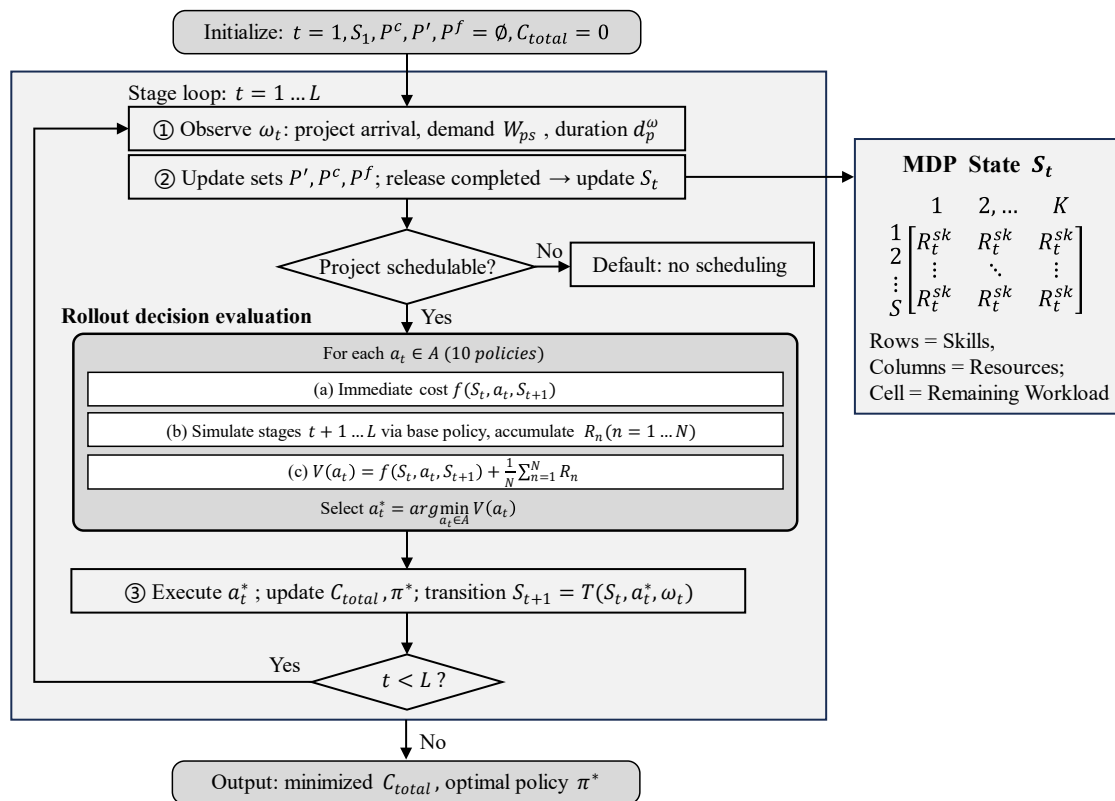


Figure 3. Flowchart of the rollout-based ADP algorithm. The superscript asterisk denotes the selected (approximately optimal) decision.

5. Computational Experiments

This section evaluates the performance of our ADP algorithm through computational experiments. Section 5.1 introduces the generation method of the benchmark dataset. Section 5.2 defines the performance measures. Section 5.3 determines the parameters of our ADP algorithm through preliminary experiments. Section 5.4 reports and analyzes the computational results. Section 5.5 presents the sensitivity analysis. Section 5.6 extends the model and the algorithm to penalty-augmented objectives. All algorithms were implemented in Python 3.8. The computational experiments were conducted on a Linux cluster, where each compute node was equipped with two Intel Xeon processors (96 cores in total) and 384 GB of memory, and tasks were submitted in an exclusive-node mode.

5.1. Benchmark Dataset

Since no standard benchmark dataset exists for our problem, we generate problem instances based on the Project Scheduling Problem Library (PSPLIB) [49], as detailed below.

Whether a new project arrives in each stage is drawn from a Bernoulli distribution with arrival probability $p = 0.5$. For each arriving project, the workload required for skill s , denoted W_{ps} , is drawn from a uniform distribution $U(0, a \times totalEfficiency_{skill})$, where $a = 0.2$ is a coefficient that controls the range of the required workload, and $totalEfficiency_{skill}$ is the sum of the working capacities of all human resources for that skill. The per-unit-time working capacity η_{sk} of each employee k for skill s is drawn from a uniform distribution $U(1, 10)$. The cost of an employee is proportional to their skill level and is generated from the interval $[c_{base}/Skill_{max}, c_{base}/Skill_{min}]$.

For the project durations, a scenario set $\Omega_p = \{1, 2, \dots, 7\}$ is constructed in advance for each project p , with each scenario occurring with probability $p(\omega) = \frac{1}{7}$. When a project arrives in the system, a duration realization d_p is drawn from Ω_p according to $p(\omega)$,

and d_p then remains fixed throughout the entire life cycle of the project. This realization is consistent with the setting in Section 3.1, and the scenario structure is retained so that the framework can later be extended to non-uniform distributions. The parameter values involved in the experiments are summarized in Table 4.

Table 4. Parameter settings of the dataset.

Type	Parameter	Range
Project	Number of required skill types $\ S\ $	$\{3,7,10\}$
	Skill workload requirement W_{ps}	$U(0,0.2 \times totalEfficiency_{skill})$
	Duration scenario set Ω_p	$\{1,2,\dots,7\}, p(\omega) = \frac{1}{7}$
Human resources	Number of human resources $\ K\ $	$\{5,10,15\}$
	Skill working capacity η_{sk}	$U(1,10)$
	Variable cost C_{sk}^v and fixed cost C_k^f	$[c_{base}/Skill_{max}, c_{base}/Skill_{min}]$
Environment	Project arrival probability	$Bernoulli(0.5)$

5.2. Performance Measures

We adopt the following six measures to evaluate the performance of the algorithms.

(1) Total cost $C_{total} = \sum_{t=1}^L r_t$, where r_t is the human resource cost incurred in stage t . This measure corresponds directly to the objective function of the MDP model.

(2) Project completion per unit cost $C_{efficiency} = P_f / C_{total}$, which measures the number of projects that can be completed per unit of cost. A larger value indicates higher cost efficiency.

(3) Average human resource utilization $K_{rate} = \frac{1}{L} \sum_{t=1}^L \frac{\sum_{k \in K} x_k}{K}$, where $x_k \in \{0,1\}$ indicates whether employee k is used in the current stage. Under comparable project completion rates, a lower utilization indicates a more economical resource allocation.

(4) Project completion rate $C_{rate} = P_f / P$, that is, the ratio of the number of completed projects to the total number of arriving projects. This measure reflects the scheduling capability of the algorithm.

(5) Total delay stages $TDS = \sum_{t=1}^L D_t$, where D_t is the number of projects that have not yet started in stage t . A smaller TDS indicates better scheduling timeliness.

(6) Average relative deviation $ARD = \frac{1}{n} \sum_{i=1}^n \left| \frac{C_{total,i}^{QL} - C_{total,i}^{Rollout}}{C_{total,i}^{Rollout}} \right| \times 100\%$, which measures the relative difference in total cost between the Q-learning algorithm and our rollout-based ADP algorithm on the same instances. A positive ARD indicates that the rollout-based ADP algorithm obtains a lower total cost than Q-learning, whereas a negative ARD indicates that Q-learning obtains a lower total cost.

Among these measures, only C_{total} corresponds to the objective function of the MDP model in Section 3.3. The remaining measures, in particular the completion rate C_{rate} and the total delay stages TDS , are complementary evaluation indicators rather than optimization targets. They are reported to reveal how a cost-oriented policy behaves along dimensions of managerial interest and to expose possible trade-offs between cost and throughput, and this distinction underlies the analysis in Sections 5.4 to 5.6.

5.3. Parameter Settings of Our ADP Algorithm

Our ADP algorithm involves two key parameters: the base policy and the number of Monte Carlo simulations N . The base policy determines how the decision sequence for future stages is quickly generated in each simulation, and thus directly affects the quality of the cost-to-go estimation. This section determines these two parameters through preliminary experiments.

We compare three base policies under the setting $L = 10, K = 10, S = 7, N = 500$. The randomized policy selects a decision at random from the decision space in each simulation stage; the round-robin policy selects decisions sequentially in a predefined order; and the minimum-cost policy selects the decision with the lowest immediate cost. Each policy is run on 20 instances. The comparison results are reported in Tables 5 and 6.

Table 5. Total cost of our ADP algorithm under different base policies.

Base Policy	L	N	C_{total}	$C_{efficiency}$	K_{rate}	TDS
Randomized policy	10	500	22,896	0.00008983	0.525	9.7
Round-robin policy	10	500	22,665	0.00009751	0.506	8.6
Minimum-cost policy	10	500	35,350	0.00005600	0.545	7.6

Table 5 shows that the randomized policy and the round-robin policy produce very similar total costs, with a difference of less than 2%. By contrast, the total cost of the minimum-cost policy is much higher, exceeding the randomized and round-robin policies by about 54% and 56%, respectively. This result indicates that a myopic preference for the lowest immediate cost does not necessarily reduce the cumulative cost over the whole planning horizon. The minimum-cost policy tends to select lower-cost but less capable employees, which may prolong resource occupation and increase the accumulated cost in later stages.

Table 6. Project completion counts and rates of our ADP algorithm under different base policies.

Base Policy	Total Projects	Completed Projects	Not Started Projects	In-Progress Projects	C_{rate}
Randomized policy	4.7	1.7	2.4	0.6	0.397
Round-robin policy	4.7	2.0	2.2	0.5	0.451
Minimum-cost policy	4.7	2.0	1.9	0.8	0.426

Table 6 shows that the round-robin policy obtains the highest completion rate, followed by the minimum-cost policy and the randomized policy. However, the randomized policy is not selected because it dominates the other base policies in all preliminary indicators. Rather, it is selected because its total cost is close to that of the round-robin policy, while its stochastic nature is more suitable for the Monte Carlo rollout procedure. In the rollout evaluation, the role of the base policy is to generate diverse future decision trajectories rather than to act as a standalone scheduling rule. A deterministic policy such as round-robin may repeatedly simulate similar future paths, whereas the randomized policy provides broader exploration of possible future decisions. Taking both computational design and preliminary performance into account, the randomized policy is adopted as the base policy in the subsequent experiments. The influence of the number of simulations will be analyzed in the main experiments of Section 5.4.1 through a comparison over $N \in \{100, 500, 1000\}$.

5.4. Results

5.4.1. Comparison with Q-Learning Under Different Problem Scales

To evaluate the performance of our ADP algorithm, we introduce the Q-learning algorithm [46] as a comparison baseline. Q-learning is a classical model-free reinforcement learning algorithm that approximates the optimal policy by continuously updating the state-action value mapping in the Q table. The update rule of the Q table is

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha[r_t + \gamma \min_a Q(S_{t+1}, a) - Q(S_t, a_t)], \quad (11)$$

where α is the learning rate and γ is the discount factor. During training, an ϵ -greedy policy is used to balance exploration and exploitation. To address the curse of dimensionality in the state space, we adopt a state aggregation technique that aggregates the multi-dimensional skill workload of each employee into a total skill amount and discretizes it. Based on an $L_9(3^4)$ orthogonal experiment, the parameters of the Q-learning algorithm are set as $\alpha = 0.1, \gamma = 0.9, \epsilon = 0.1, r_p = 53$.

According to the length of the planning horizon L , the experiments are divided into three scenarios: Scenario A ($L = 10$), Scenario B ($L = 20$), and Scenario C ($L = 30$). In all three scenarios, $K = 10, S = 7$. In each scenario, the number of simulations N of our ADP algorithm is set to 100, 500, and 1000, and the number of training episodes of the Q-learning algorithm is correspondingly set to 10,000, 20,000, and 30,000. Each combination is run on 10 instances, and the average values are reported. The comparison of the solution performance and the project completion of the two algorithms is summarized in Tables 7 and 8, respectively.

Table 7. Comparison of total cost between our ADP algorithm and Q-learning across scenarios.

Scenario	Algorithm	L	Iterations	C_{total}	$C_{efficiency}$	K_{rate}	TDS
A	ADP	10	100	28,419, 10,545, 37,080	0.00009296	0.566	7.5
			500	28,413, 10,532, 37,020	0.00009296	0.567	7.7
			1000	28,729, 10,731, 37,020	0.00009190	0.576	7.4
	Q-learning	10	10,000	23,469, 8670, 24,690	0.00006033	0.430	15.3
			20,000	26,872, 10,428, 34,150	0.00005706	0.416	14.2
			30,000	23,469, 8670, 24,690	0.00006033	0.430	15.3
B	ADP	20	100	57,694, 14,489, 42,270	0.00008932	0.581	42.4
			500	56,231, 12,392, 36,240	0.00008680	0.574	44.9
			1000	56,231, 12,392, 36,240	0.00008680	0.574	44.9
	Q-learning	20	10,000	60,566, 15,074, 54,220	0.00005827	0.530	62.5
			20,000	58,022, 10,013, 36,320	0.00005983	0.524	63.0
			30,000	65,080, 15,713, 51,230	0.00005984	0.547	58.5
C	ADP	30	100	92,787, 15,979, 61,910	0.00008523	0.633	88.0
			500	86,886, 10,150, 34,170	0.00008569	0.607	96.2
			1000	86,574, 10,073, 36,260	0.00008712	0.602	93.8
	Q-learning	30	10,000	101,857, 7730, 24,460	0.00006186	0.576	125.3
			20,000	98,090, 10,089, 29,910	0.00006108	0.575	125.1
			30,000	103,933, 10,021, 32,150	0.00005598	0.598	129.1

Note: Each cell in the C_{total} column reports three values in the format “mean, standard deviation, range” over the ten instances of each configuration, where the range is the maximum minus the minimum; all other columns report the mean over the ten instances.

Table 8. Comparison of project completion rates between our ADP algorithm and Q-learning across scenarios.

Scenario	Algorithm	L	Iterations	Total Projects	Completed Projects	Not Started Projects	In-Progress Projects	C_{rate}
A	ADP	10	100	4.5	2.3	1.3	0.9	0.560
			500	4.5	2.3	1.4	0.8	0.560
			1000	4.5	2.3	1.3	0.9	0.560
	Q-learning	10	10,000	4.5	1.4	2.6	0.5	0.357
			20,000	4.5	1.5	2.2	0.8	0.397
			30,000	4.5	1.4	2.6	0.5	0.357
B	ADP	20	100	9.6	5.0	4.0	0.6	0.533
			500	9.6	4.8	4.2	0.6	0.514

C	Q-learning	20	1000	9.6	4.8	4.2	0.6	0.514
			10,000	9.6	3.4	5.4	0.8	0.362
			20,000	9.6	3.4	5.5	0.7	0.362
			30,000	9.6	3.8	4.7	1.1	0.402
	ADP	30	100	14.8	7.9	6.0	0.9	0.533
			500	14.8	7.4	6.4	1.0	0.504
			1000	14.8	7.5	6.3	1.0	0.510
			10,000	14.8	6.3	7.7	0.8	0.428
	Q-learning	30	20,000	14.8	6.0	8.1	0.7	0.408
			30,000	14.8	5.8	8.0	1.0	0.396

Note: All values are averaged over the ten instances of each configuration.

Table 7 also reveals a special training behavior of Q-learning: in Scenario A, the results under 10,000 and 30,000 training episodes are identical. This stems from the state aggregation technique: after the multi-dimensional skill workload of each employee is aggregated into a total skill amount, the number of reachable aggregated states is small, so the Q table converges to the same greedy policy under both budgets and additional episodes revisit the same state-action pairs. In Scenarios B and C, the total cost of Q-learning does not decrease monotonically with the training budget; in Scenario B it even reaches its worst value of 65,080 at the largest budget of 30,000 episodes. This indicates that, under state aggregation, Q-learning tends to oscillate around local optima and that a larger training budget does not necessarily translate into a better policy. By contrast, our ADP algorithm is insensitive to the number of simulations N : the variation of C_{total} across the three values of N is within 1.1% in Scenario A, within 2.6% in Scenario B, and within 7.2% in Scenario C, and in Scenario C the cost decreases mildly as N grows. The lookahead sampling mechanism is thus considerably more robust to its parameter than the training-based mechanism of Q-learning. A direct comparison of the total cost of the two algorithms is shown in Figure 4.

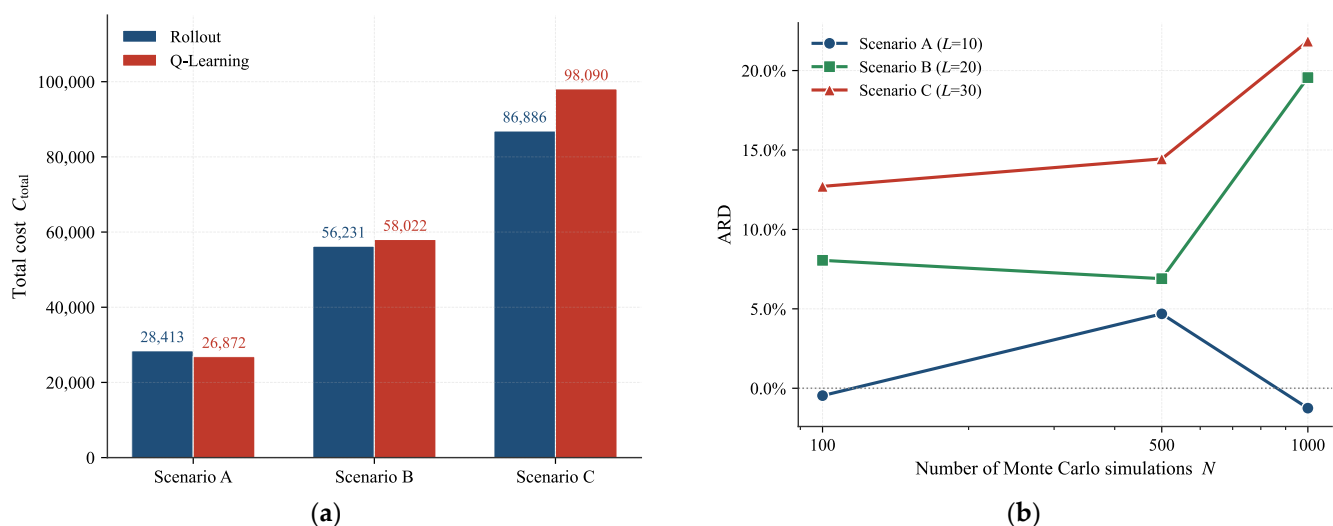


Figure 4. Comparison of total cost between our ADP algorithm and Q-learning across scenarios: (a) Total cost comparison; (b) ARD across simulation budgets.

In Scenarios B and C, our ADP algorithm dominates Q-learning on every criterion: at matched computational settings it lowers the total cost by 3.1% to 16.7%, completes more projects, and shortens delays, with the ARD remaining positive throughout and widening as the horizon lengthens. Scenario A is the only regime where Q-learning attains a lower nominal cost, but Table 8 shows this is achieved by leaving far more projects

unscheduled: its completion rate stays around 0.36 against 0.56 for our algorithm, with delays roughly twice as long. Accordingly, the cost efficiency $C_{efficiency}$ of our algorithm is higher than that of Q-learning in every setting, Scenario A included. The relative advantage of lookahead therefore grows with problem scale and complexity, and the low cost of Q-learning on the smallest instances reflects passive scheduling rather than efficient resource use.

To illustrate the difference between the two algorithms more intuitively, Figure 5 presents a scatter plot of the (C_{rate}, C_{total}) pairs across all scenarios, where each dashed line connects the mean values of ADP and Q-learning under the same scenario.

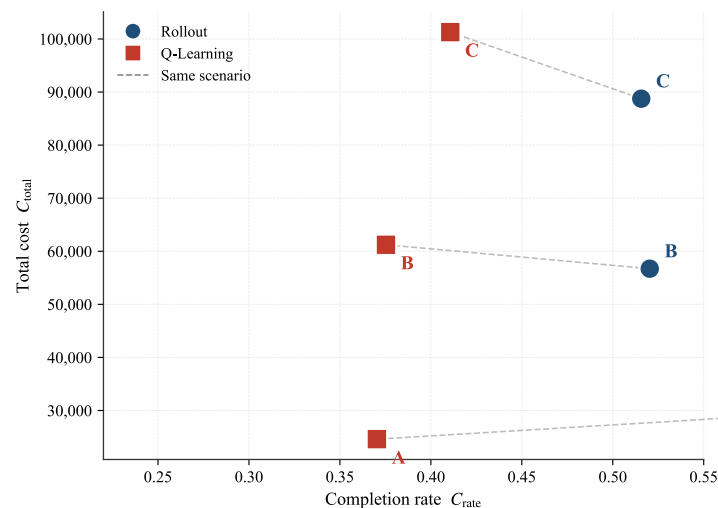


Figure 5. Trade-off between total cost and completion rate for our ADP algorithm and Q-learning across scenarios. A, B, and C denote Scenarios A, B, and C, corresponding to planning horizons $L = 10, 20$, and 30

The scatter plot confirms this reading: in every scenario the ADP points lie to the right of the Q-learning points, indicating a higher completion rate, and in Scenarios B and C they also lie below, indicating more completions at lower cost. Only in Scenario A does Q-learning sit lower but clearly to the left, the signature of its passive scheduling. Overall, the comparison is scenario-dependent: our algorithm dominates on medium and large instances, whereas on the smallest instances it trades a moderately higher cost for substantially more completions and shorter delays. It is also far less sensitive to its single parameter than Q-learning is to its training budget, which eases practical deployment.

We further assess statistical significance with two-sided Wilcoxon signed-rank tests on the paired instance-level results. The completion-rate advantage of our algorithm is significant at the 5% level in eight of the nine configurations (p between 0.004 and 0.023) and marginal in the ninth ($p = 0.063$); the cost differences are insignificant in Scenarios A and B but become significant in Scenario C at the two larger sizes. The improvement in completion is thus consistent across all scales, while the cost improvement is statistically firm on the largest instances.

5.4.2. Comparison with Heuristic Baselines and Alternative Base Policies

We further compare our ADP algorithm with three fixed heuristic policies drawn from the decision space of Table 3: a minimum-cost heuristic that always selects the allocation of policy ac_4 , a maximum-skill heuristic that always selects the allocation of policy ac_1 , and a maximum-projects maximum-skill heuristic that schedules as many projects as possible under the maximum-skill logic. To isolate the contribution of the rollout mechanism, we also run two rollout variants in which the randomized base policy is replaced

by the minimum-cost policy and by the maximum-skill policy, respectively. All methods are tested on the Scenario B configuration ($L = 20$, $K = 10$, $S = 7$, $N = 500$). Table 9 reports the average results.

Table 9. Comparison with heuristic baselines and rollout variants.

Algorithm	C_{total}	$C_{efficiency}$	K_{rate}	TDS	Completed	Not Started	C_{rate}
Rollout-based ADP (randomized base)	56,231	0.00008680	0.574	44.9	4.8	4.2	0.514
Q-learning	58,022	0.00005983	0.524	63.0	3.4	5.5	0.362
Heuristic minCost	62,183	0.00008013	0.473	46.4	4.7	4.1	0.496
Heuristic maxSkill	106,967	0.00006584	0.605	16.1	6.9	1.4	0.720
Heuristic maxProjectsMaxSkill	108,954	0.00006652	0.607	16.6	7.0	1.3	0.734
Rollout with minCost base	56,955	0.00009093	0.572	43.4	5.0	4.0	0.532
Rollout with maxSkill base	62,073	0.00008324	0.602	39.6	5.0	3.8	0.527

Table 9 shows that our ADP algorithm achieves the best cost-completion balance. Compared with the minimum-cost heuristic, it reduces the total cost from 62,183 to 56,231 with a slightly higher completion rate. The two maximum-skill heuristics complete more projects but at nearly twice the cost. Moreover, the rollout mechanism improves upon every base policy it embeds, and among the three rollout variants the randomized base policy yields the lowest total cost, which supports its use in the main experiments.

5.4.3. Analysis of Restricted Decision Space and Computational Effort

The decision space of Section 4.2 restricts the complete action space, so we quantify the resulting performance loss on two small configurations ($L = 5$, $K = 5$, $S = 3$ and $L = 6$, $K = 5$, $S = 3$, arrival probability 0.5) where full enumeration is feasible, using ten instances per configuration. FullExact enumerates all feasible non-empty scheduling actions at each stage within a safeguard limit of 50,000 candidates per stage; TenActionDP uses the same evaluation procedure but restricts the candidate set to the actions generated by the ten representative policies in Table 3; and RandomSample- m supplements the ten actions with up to m randomly sampled feasible plans ($m = 100, 1000, 10,000$). The safeguard limit is never reached at $L = 5$, so FullExact is an exact benchmark there; at $L = 6$ the limit is reached on at least one instance, so FullExact is a capped full-action benchmark. Table 10 reports the results.

Table 10. Solution quality of the compared action-space designs on small instances, 10 instances per configuration.

Configuration	Method	C_{total}	Gap vs. FullExact (%)	C_{rate}	Completed	TDS
$L = 5, K = 5, S = 3$	FullExact	3204	0	0.293	0.9	0.2
	RandomSample-100	3607	16.5	0.300	1.0	0
	RandomSample-1000	3298	2.9	0.300	1.0	0.2
	RandomSample-10000	3204	0.0	0.300	1.0	0.2
	TenActionDP	3803	24.9	0.300	1.0	0
$L = 6, K = 5, S = 3$	FullExact	3622	0	0.380	1.1	0.4
	RandomSample-100	4009	10.9	0.417	1.3	0.3
	RandomSample-1000	3831	4.6	0.417	1.3	0.4
	RandomSample-10000	3628	0.1	0.397	1.2	0.4
	TenActionDP	4416	26.2	0.417	1.3	0

The results indicate that the restriction is not free: the total cost of TenActionDP exceeds that of FullExact by 24.9% at $L = 5$ and by 26.2% at $L = 6$, both averaged over the per-instance relative gaps. Narrowing this gap by random sampling is expensive: RandomSample-1000 still leaves a gap of 2.9% to 4.6%, and only RandomSample-10000, which evaluates three orders of magnitude more candidates, matches the FullExact cost. Meanwhile, TenActionDP attains completion rates and delays no worse than FullExact, because FullExact minimizes cost only and occasionally prefers to leave a project unscheduled. The ten-policy design thus trades a measured cost gap for the scalability that makes the rollout algorithm applicable to realistic problem sizes.

Table 11 reports the number of evaluated actions and the elapsed time of the compared action-space designs.

Table 11. Number of evaluated actions and elapsed time of the action-space designs, 10 instances per configuration.

Configuration	Method	Avg. Evaluated Actions	Max. Evaluated Actions	Elapsed Time (s)	Capped
$L = 5, K = 5, S = 3$	FullExact	21.99	17,465	5416.6	No
	RandomSample-100	5.51	108	666.1	No
	RandomSample-1000	10.04	1008	3445.7	No
	RandomSample-10000	22.26	10,008	4860.4	No
	TenActionDP	2.05	8	10.0	No
$L = 6, K = 5, S = 3$	FullExact	12.75	25,393	10,960.2	Yes
	RandomSample-100	5.49	110	99.8	Yes
	RandomSample-1000	12.51	1010	512.2	Yes
	RandomSample-10000	12.75	10,008	5304.3	No
	TenActionDP	2.06	8	28.3	No

Note: “Capped” indicates that the safeguard limit of 50,000 feasible candidate actions per stage is reached; in this case, FullExact is a capped full-action benchmark rather than a complete enumeration; stages that reach the limit are not fully enumerated, so the evaluated-action counts reported in the table can lie below the limit.

TenActionDP evaluates on average about two candidate actions per stage and completes the ten instances in 10.0 and 28.3 s on the two configurations, whereas full enumeration evaluates up to 17,465 and 25,393 actions in a single stage and requires 5416.6 and 10,960.2 s, more than two orders of magnitude longer. Approaching the enumeration result by random sampling is similarly expensive. The ten-policy design is therefore what keeps the online rollout tractable.

We further place our ADP algorithm and Q-learning on a common end-to-end footing. Q-learning concentrates its effort in offline training, after which each online decision reduces to a table lookup, whereas the rollout policy performs no training but carries out N Monte Carlo simulations for each candidate decision online. Table 12 reports the wall-clock training, inference, and online decision times on the main experiments.

Table 12. Wall-clock computational effort of the two algorithms on the main experiments, 10 instances per configuration.

Scenario	N /Episodes	Rollout Online per Instance (s)	Rollout Online Total (s)	Q-Learning Training (s)	Q-Learning Inference per Instance (s)	Q-Learning End-to-End (s)
A	100/10,000	13.7	136.8	53.6	0.015	53.8
A	500/20,000	54.8	547.9	108.4	0.015	108.5
A	1000/30,000	108.2	1082.0	162.5	0.015	162.6

B	100/10,000	69.4	694.4	146.9	0.042	147.4
B	500/20,000	333.1	3331.0	296.2	0.040	296.6
B	1000/30,000	658.7	6586.7	444.0	0.041	444.4
C	100/10,000	191.4	1913.9	277.9	0.062	278.5
C	500/20,000	947.5	9474.9	557.2	0.069	557.9
C	1000/30,000	1882.0	18,819.9	841.2	0.063	841.8

Table 12 shows that Q-learning is clearly faster: once trained, it answers each test instance in well under 0.1 s, and even including training time it completes a configuration of ten instances in 53.8 to 841.8 s, against 136.8 to 18,819.9 s for the rollout policy. The cost and completion advantages of our ADP algorithm are therefore obtained at a higher computational price. However, the trained Q table is tied to a fixed arrival process and must be retrained whenever the environment shifts, whereas the rollout policy requires no training and adapts to the arrival patterns of Section 5.5 without additional offline effort. For real-time deployment on substantially larger instances, the online simulation budget would become the binding constraint, which motivates the acceleration strategies discussed in Section 6.3.

5.5. Sensitivity Analysis

We next examine the influence of the number of human resources K and the number of skills S on the performance of our ADP algorithm. In the experiments, $L = 20$ (Scenario B) is fixed, K takes the values $\{5, 10, 15\}$, and S takes the values $\{3, 7, 10\}$, giving nine parameter combinations. Each combination is run on 10 instances, and the average values are reported. Tables 13 and 14 report the solution performance and the project completion, respectively.

Table 13. Total cost of our ADP algorithm under different (K, S) combinations, $L = 20, N = 500$.

K	S	C_{total}	$C_{efficiency}$	K_{rate}	TDS
5	3	17,285	0.00057558	0.309	0.6
5	7	24,702	0.00016844	0.614	53.8
5	10	14,276	0.00020365	0.405	75.7
10	3	30,815	0.00030815	0.318	0.3
10	7	56,231	0.00008680	0.574	44.9
10	10	58,503	0.00007688	0.621	49.3
15	3	46,830	0.00021891	0.281	0
15	7	85,451	0.00005618	0.567	40.1
15	10	97,230	0.00004877	0.599	50.5

Table 14. Project completion counts under different (K, S) values.

K	S	Total Projects	Completed Projects	Not Started Projects	In-Progress Projects	C_{rate}
5	3	9.6	8.3	0	1.3	0.865
5	7	9.6	4.2	4.7	0.7	0.450
5	10	9.6	2.7	6.4	0.5	0.270
10	3	9.6	8.4	0.1	1.1	0.875
10	7	9.6	4.8	4.2	0.6	0.514
10	10	9.6	4.4	4.4	0.8	0.474
15	3	9.6	8.4	0	1.2	0.875
15	7	9.6	4.8	4.1	0.7	0.508
15	10	9.6	4.6	4.5	0.5	0.497

We first vary the number of skill types S and the workforce size K one at a time. Increasing S at a fixed K sharply degrades performance: at $K = 5$, the completion rate falls from 0.865 at $S = 3$ to 0.270 at $S = 10$, and the same pattern holds for larger workforces. This reflects a dilution effect, as each employee must spread effort over more skills while the projects' skill demands widen, raising the matching difficulty. A low total cost in this regime is therefore misleading, since it signals that few projects are scheduled rather than efficient use of resources. Increasing K at a fixed S helps only up to a point: completion improves initially but soon saturates, while cost keeps rising and the cost efficiency $C_{efficiency}$ declines monotonically, so under high skill-matching difficulty the marginal return of additional headcount falls rapidly.

Figures 6 and 7 display the joint effect. Total cost rises with K and steepens as S grows, whereas the completion rate is governed almost entirely by S : it stays near 0.87 and is essentially flat in K when $S = 3$, but drops sharply and responds only weakly to K at $S = 7$ and $S = 10$. The non-parallel curves reveal a clear interaction: the same headcount increment yields very different returns depending on skill complexity. The managerial reading is that workforce size and skill breadth must be configured jointly rather than expanded along a single dimension, as elaborated in Section 6.2.

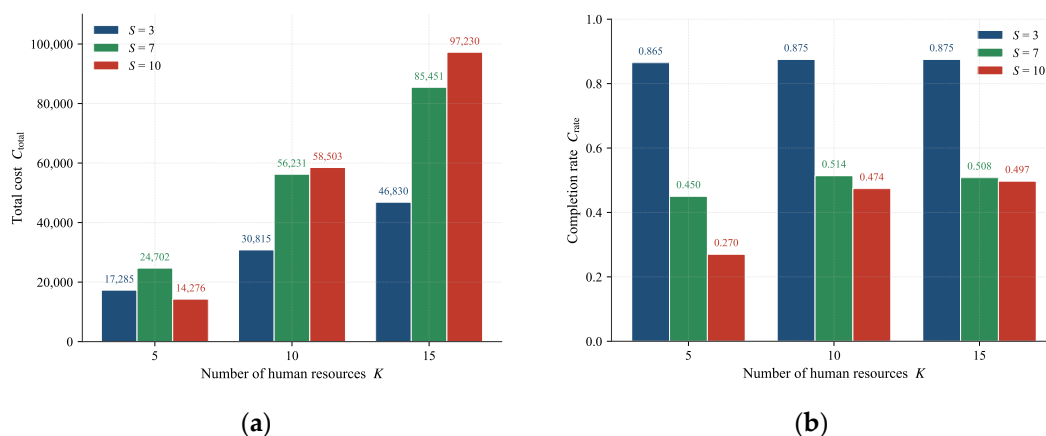


Figure 6. Total cost and completion rate under different (K, S) combinations: (a) Total cost; (b) Completion rate.

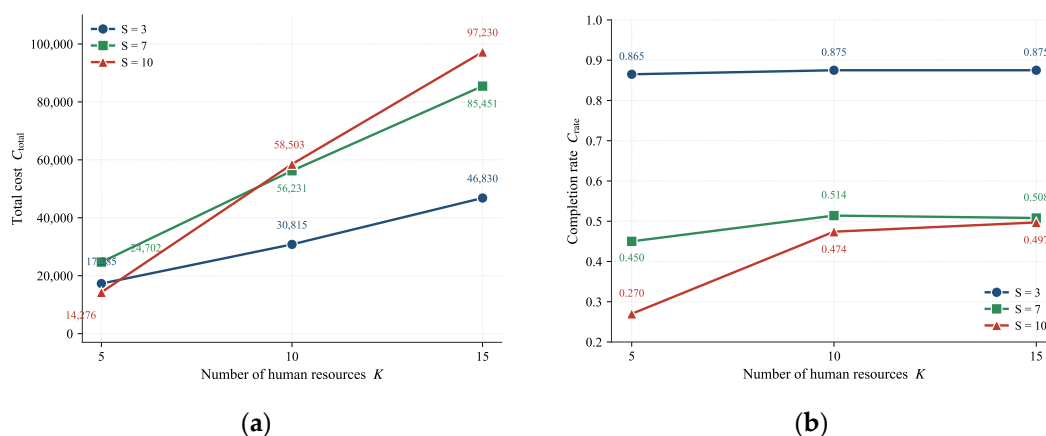


Figure 7. Interaction effects of K and S on total cost and completion rate: (a) Cost response to K ; (b) Completion response to K .

The above results yield three concrete managerial guidelines. First, when the skill portfolio is narrow, the workforce should be sized to demand rather than expanded defensively: at $S = 3$, five employees already achieve a completion rate of 0.865, and tripling

the headcount raises the cost by 171% (from 17,285 to 46,830) without any gain in completion. Second, when the skill portfolio is broad, headcount expansion helps only up to a saturation point: at $S = 7$, moving from five to ten employees lifts the completion rate by 6.4 percentage points, whereas moving from ten to fifteen yields no further gain while the cost rises by another 52%; beyond this point, managers should invest in cross-training that widens the skill coverage of existing employees instead of hiring. Third, the cost efficiency $C_{efficiency}$ provides a practical stopping signal for capacity expansion, since it declines monotonically in K for every skill level in our experiments; once $C_{efficiency}$ falls below an organization-specific threshold, additional headcount converts into idle or poorly matched capacity, and the configuration of headcount and skill coverage should be re-balanced jointly rather than expanded along a single dimension.

We further examine the robustness of our ADP algorithm to the project arrival pattern. The stationary Bernoulli process with $p = 0.5$ used so far can be viewed as the discrete-time analogue of a Poisson arrival stream observed at unit-stage granularity. We now fix the Scenario B configuration ($L = 20$, $K = 10$, $S = 7$, $N = 500$) and vary the per-stage arrival probability p_t over seven patterns: the stationary pattern ($p_t = 0.5$), the low-load and high-load patterns ($p_t = 0.3$ and 0.7), the increasing and decreasing patterns in which p_t rises or falls linearly around an average of 0.5, and the front-loaded and back-loaded patterns that concentrate the arrival mass in the first or second half of the horizon while keeping the average intensity at 0.5. Each pattern is run on ten instances, and Table 15 reports the average results.

Table 15. Performance of our ADP algorithm under different project arrival patterns.

Arrival Pattern	C_{total}	$C_{efficiency}$	K_{rate}	TDS	Total Projects	Completed	Not Started	In Progress	C_{rate}
Stationary ($p_t = 0.5$)	56,231	0.00008680	0.574	44.9	9.6	4.8	4.2	0.6	0.514
Low-load ($p_t = 0.3$)	43,534	0.00010921	0.442	12.9	6.0	4.2	1.3	0.5	0.742
High-load ($p_t = 0.7$)	60,610	0.00008803	0.633	80.8	13.7	5.2	7.7	0.8	0.380
Increasing	44,449	0.00009863	0.499	34.2	9.8	4.0	5.0	0.8	0.422
Decreasing	60,616	0.00009737	0.631	60.2	10.3	5.7	3.7	0.9	0.565
Front-loaded	63,791	0.00008621	0.642	57.8	10.1	5.3	4.0	0.8	0.533
Back-loaded	46,652	0.00010348	0.510	32.2	9.6	4.5	4.4	0.7	0.480

Table 15 shows that the offered workload, rather than the shape of the arrival stream, is the primary performance driver. Under the low pattern the completion rate reaches 0.742 with a total delay of 12.9 stages, whereas under the high pattern it falls to 0.380 with a delay of 80.8 stages. Among the four non-stationary patterns with the same average intensity, early-concentrated streams complete more projects at higher cost, because early arrivals can still be finished within the horizon, and late-concentrated streams show the opposite profile. Moreover, the cost per completed project stays within a narrow band (10,365 to 12,036, a spread of about 16%) across all seven patterns. The algorithm therefore preserves a stable cost-effectiveness profile under non-stationary arrivals without any re-tuning.

5.6. Extension to Penalty-Augmented Objectives

The objective of the MDP model prices only resource usage, which is appropriate when unscheduled projects are carried over to the next planning horizon without contractual damage, as described in Section 3.1. When delays do carry economic consequences, the model accommodates them with a minor modification. Let $C_w \geq 0$ denote the waiting cost charged for each project that remains unscheduled at the end of a stage. The single-stage cost function in Equation (7) is then replaced by

$$\tilde{f}(S_t, a_t, S_{t+1}) = f(S_t, a_t, S_{t+1}) + C_w \times |P'_{t+1}|, \quad (12)$$

and all other elements of the model are unchanged. Because the penalty accrues one unit per waiting project per stage, the resulting objective equals $C_{total} + C_w \times TDS$, and setting $C_w = 0$ recovers the original model. On the algorithmic side, only the immediate cost evaluations in Algorithms 1 and 2 need to be replaced by Equation (12). A larger C_w shifts the policy toward the completion-oriented corner of the trade-off. To verify this effect, we rerun both algorithms on the Scenario B configuration ($L = 20$, $K = 10$, $S = 7$, $N = 500$, 20,000 training episodes) with $C_w \in \{0, 500, 1000, 2000\}$ on the same ten instances. Table 16 reports the results.

Table 16. Pilot study of the penalty-augmented model on the Scenario B configuration, 10 instances per setting.

C_w	Algorithm	C_{total}	TDS	$C_w \times TDS$	Penalized Objective	C_{rate}
0	Rollout-based ADP	56,231	44.9	0	56,231	0.514
0	Q-learning	58,022	63.0	0	58,022	0.362
500	Rollout-based ADP	56,543	34.9	17,450	73,993	0.536
500	Q-learning	62,481	48.1	24,050	86,531	0.496
1000	Rollout-based ADP	67,392	21.8	21,800	89,192	0.642
1000	Q-learning	62,090	59.7	59,700	121,790	0.398
2000	Rollout-based ADP	77,548	13.3	26,600	104,148	0.719
2000	Q-learning	103,692	19.2	38,400	142,092	0.747

As C_w grows from 0 to 2000, our ADP algorithm raises its completion rate from 0.514 to 0.719 and cuts the total delay stages by 70.4%, while its cost rises by 37.9%; the transition is monotone in C_w , so a manager can steer the policy toward higher completion by calibrating a single coefficient. Our ADP algorithm also attains the lower penalized objective at every C_w , with margins of 3.1% to 26.8% over Q-learning. Q-learning responds non-monotonically to the penalty: at $C_w = 1000$ its completion rate falls back to 0.398, which mirrors the oscillation under state aggregation observed in Section 5.4.1.

6. Discussion

Based on the above computational results, this section discusses the theoretical contributions, the practical implications, and the limitations of this study.

6.1. Theoretical Contributions and Implications

First, we propose and formalize the multi-skilled multi-project dynamic scheduling problem with random project arrivals and uncertain durations. Prior dynamic multi-project models with random arrivals assume a single resource type [14,17,18]; prior multi-

skilled models under uncertainty fix the project set in advance [36,37] or address shop-floor task assignment rather than project portfolios [38,39]; and recent DRL approaches likewise concentrate on shop-floor structures or anonymous capacities [41,42]. To the best of our knowledge, our MDP model is the first to combine random project arrivals, uncertain durations, and the sharing of multi-skilled human resources in a single sequential decision framework. It extends dynamic project scheduling theory to the multi-skilled setting and provides a formal basis for richer cost structures, such as the penalty-augmented objective of Section 5.6 and skill-dependent cost rates.

Second, we contribute to the ADP methodology for dynamic scheduling. The finding in Section 5.4.2 that the rollout policy improves upon every base policy it embeds extends the empirical evidence for the sequential improvement property of rollout algorithms [16,47,48] to a multi-skilled, multi-project environment, and the quantified cost gap of about 25% in Section 5.4.3 clarifies the price of restricting the action space, a question rarely measured in the ADP scheduling literature.

Third, our results refine the understanding of when lookahead-based methods pay off. The finding that our ADP algorithm outperforms Q-learning and fixed priority rules on medium- and large-scale instances echoes Li and Womer [15] and Satic et al. [14,18], who report that lookahead-based methods dominate myopic rules as the problem grows. The behavior on small instances, however, has no direct counterpart in these studies: because the objective prices only resource usage, a policy can lower its nominal cost by declining to schedule projects, and Q-learning exploits this possibility in Scenario A. This cost-completion trade-off would remain hidden under the makespan or throughput objectives that dominate the prior ADP literature. In addition, the interaction between workforce size and skill breadth identified in Section 5.5 gives quantitative form, in a dynamic setting, to the observation that multi-skilling and staffing levels must be configured jointly [6,27].

6.2. Practical Implications

The findings of this study offer several implications for managers of knowledge-intensive projects. First, workforce capacity and skill breadth should be planned jointly. The sensitivity analysis in Section 5.5 shows that when skill requirements are simple, a small workforce already completes the vast majority of projects and further hiring only inflates cost, whereas when skill requirements are complex, adding employees alone yields rapidly diminishing returns.

Second, cost figures should be read jointly with completion indicators. A low nominal cost may simply reflect that few projects are scheduled, as observed for Q-learning on the small-scale instances. Monitoring the completion rate and the delay alongside the cost prevents such passive scheduling from being mistaken for efficiency.

Third, the proposed algorithm is usable as an online scheduling tool whenever the decision interval accommodates its per-decision simulation budget. It requires no offline training, adapts to non-stationary arrival patterns without re-tuning, and is markedly less sensitive to its parameter than Q-learning is to its training budget. When project delays carry contractual consequences, the manager only needs to calibrate a single waiting-cost coefficient to steer the policy toward higher completion, as shown in Section 5.6.

Finally, the choice between our ADP algorithm and a trained learning method is itself a managerial decision. Q-learning is preferable when the environment is stationary and its training cost can be amortized over many decisions, whereas our rollout-based algorithm is preferable when responsiveness to environmental change outweighs decision speed.

6.3. Limitations and Future Research Directions

Several limitations of this study should be acknowledged. First, the action space is restricted to ten representative policies, and Section 5.4.3 shows that this restriction carries a measured cost gap of about 25% relative to an exact full-action benchmark at $L = 5$ and to a capped full-action benchmark at $L = 6$; the reported solutions are therefore approximations, and the gap on larger instances, although not measurable exactly, is unlikely to vanish. Second, the model admits at most one project arrival per stage and the experiments instantiate variable costs at the employee level; both are simplifications whose relaxations, batch arrivals and fully skill-dependent costs, are structurally supported by the model but were not tested empirically. Third, each configuration is evaluated on ten instances; this permits matched-instance paired tests but still limits the precision of the reported means, as reflected by the standard deviations in Table 7. Fourth, the learning baseline is a tabular Q-learning algorithm with state aggregation; it is a natural like-for-like benchmark for an online MDP policy, but it does not represent the current frontier of deep reinforcement learning, and a comparison against DRL methods with function approximation [41,42] remains open. Finally, all instances are generated from PSPLIB-based distributions, and validation on organizational data is still required before the managerial implications of Section 5.5 can be applied quantitatively.

The limitations above translate into specific research directions. A first direction is to extend the pilot of Section 5.6 into a systematic study of the penalty-augmented model of Equation (12), characterizing how the optimal policy migrates along the cost and completion frontier as C_w varies and how C_w should be estimated from contractual data. A second direction is to allow batch arrivals and correlated arrival processes, which mainly affects the per-stage decision enumeration and would benefit from column-generation-style pruning of the allocation plans. A third direction is methodological: replacing the Monte Carlo estimate of the cost-to-go function with a learned value function, for example a neural approximation trained offline and corrected online by short rollouts, would combine the adaptivity of rollout with the speed of trained policies. Finally, applying the framework to field data from an R&D or software organization, including skill-dependent cost rates and empirically fitted duration distributions, would test the external validity of the findings.

7. Conclusions

This study sets out to answer a single managerial question: how should a portfolio of knowledge-intensive projects, staffed by a shared pool of multi-skilled workers, be scheduled when both project arrivals and durations are uncertain? We formulated the multi-skilled multi-project dynamic scheduling problem as a Markov decision process, with the available skill workload of each worker as the state, the resource allocation plan as the decision, and the total cost of human resources as the objective, and solved it with a rollout-based ADP algorithm that approximates the cost-to-go function via Monte Carlo simulation with a randomized base policy while confining the action set to ten representative allocation policies.

The experimental evidence supports a calibrated conclusion. On medium- and large-scale instances, the proposed policy reduced the average total cost by 3.1% to 16.7% relative to Q-learning while completing more projects with shorter delays, and the cost advantage was statistically significant on the largest instances; on small-scale instances, the comparison reduced to a trade-off in which the policy completed substantially more projects at a moderately higher nominal cost. The policy dominated fixed priority rules of the minimum-cost type, improved upon every base policy it embedded, remained stable across simulation budgets and across seven stationary and non-stationary arrival patterns, and its restricted action space was shown on small instances to trade a measured cost gap relative to the full-action benchmarks for a computational effort smaller by more than two

orders of magnitude. Overall, rollout-based ADP offers an effective and practical approach to online scheduling in dynamic multi-skilled multi-project settings, particularly for medium- and large-scale problems, with its limitations and future directions discussed in Section 6.3.

Author Contributions: Conceptualization, H.L. and X.Z.; methodology, H.L. and W.C.; validation, H.L. and W.C.; writing—original draft preparation, H.L. and W.C.; writing—review and editing, H.L., Q.Z., W.C. and Z.W.; visualization, H.L. and W.C.; supervision, H.L.; project administration, H.L.; funding acquisition, H.L. and X.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Soft Science Research Project of the Shanghai Municipal High-Level Institution Construction and Operation Plan (grant number 25692107900).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: The authors thank the editor and reviewers for providing valuable suggestions that have improved the quality of this paper.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Li, C.-L.; Hall, N.G. Work package sizing and project performance. *Oper. Res.* **2019**, *67*, 123–142.
- Project Management Institute (PMI). *Ahead of the Curve: Forging a Future-Focused Culture*; Pulse of the Profession; Project Management Institute: Newtown Square, PA, USA, 2020.
- Li, H.; Demeulemeester, E. A genetic algorithm for the robust resource leveling problem. *J. Sched.* **2016**, *19*, 43–60.
- Li, H.; Xiong, L.; Liu, Y.; Li, H. An effective genetic algorithm for the resource levelling problem with generalised precedence relations. *Int. J. Prod. Res.* **2018**, *56*, 2054–2075.
- Wang, Y.; Xu, D.; Zhou, L.; Li, Z. Standard revision project scheduling problem considering coordination degree of standards systems. *Systems* **2025**, *13*, 685.
- De Bruecker, P.; Van den Bergh, J.; Beliën, J.; Demeulemeester, E. Workforce planning incorporating skills: State of the art. *Eur. J. Oper. Res.* **2015**, *243*, 1–16.
- Li, H.; Chen, R.; Kong, M.; Zhang, X. Dynamic centralized public R&D project portfolio selection. *IEEE Trans. Eng. Manag.* **2025**, *72*, 3542–3558.
- Correia, I.; Lourenço, L.L.; Saldanha-da-Gama, F. Project scheduling with flexible resources: Formulation and inequalities. *OR Spectr.* **2012**, *34*, 635–663.
- Tao, S.; Dong, Z.S. Multi-mode resource-constrained project scheduling problem with alternative project structures. *Comput. Ind. Eng.* **2018**, *125*, 333–347.
- Almeida, B.F.; Correia, I.; Saldanha-da-Gama, F. Priority-based heuristics for the multi-skill resource constrained project scheduling problem. *Expert Syst. Appl.* **2016**, *57*, 91–103.
- Heimerl, C.; Kolisch, R. Work assignment to and qualification of multi-skilled human resources under knowledge depreciation and company skill level targets. *Int. J. Prod. Res.* **2010**, *48*, 3759–3781.
- Snauwaert, J.; Vanhoucke, M. A new algorithm for resource-constrained project scheduling with breadth and depth of skills. *Eur. J. Oper. Res.* **2021**, *292*, 43–59.
- Herroelen, W.; Leus, R. Project scheduling under uncertainty: Survey and research potentials. *Eur. J. Oper. Res.* **2005**, *165*, 289–306.
- Satic, U.; Jacko, P.; Kirkbride, C. A simulation-based approximate dynamic programming approach to dynamic and stochastic resource-constrained multi-project scheduling problem. *Eur. J. Oper. Res.* **2024**, *315*, 454–469.

15. Li, H.; Womer, K. Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming. *Eur. J. Oper. Res.* **2015**, *246*, 20–33.
16. Xie, F.; Li, H.; Xu, Z. An approximate dynamic programming approach to project scheduling with uncertain resource availabilities. *Appl. Math. Model.* **2021**, *97*, 226–243.
17. Melchior, P. *Dynamic and Stochastic Multi-Project Planning*; Lecture Notes in Economics and Mathematical Systems; Springer: Cham, Switzerland, 2015; Volume 673.
18. Satic, U.; Jacko, P.; Kirkbride, C. Performance evaluation of scheduling policies for the dynamic and stochastic resource-constrained multi-project scheduling problem. *Int. J. Prod. Res.* **2022**, *60*, 1411–1423.
19. Choi, J.; Realff, M.J.; Lee, J.H. Dynamic programming in a heuristically confined state space: A stochastic resource-constrained project scheduling application. *Comput. Chem. Eng.* **2004**, *28*, 1039–1058.
20. Li, H.; Zheng, L.; Chen, R.; Zhang, X. Stochastic resource leveling in projects with flexible structures. *Comput. Oper. Res.* **2024**, *169*, 106753.
21. Li, H.; Yuan, Z.; Yang, M.; Yuan, Z.; Zhang, X. Project portfolio selection and scheduling considering net present value and delayed delivery. *J. Oper. Res. Soc.* **2025**, 1–17. <https://doi.org/10.1080/01605682.2025.2572996>.
22. Creemers, S. Maximizing the expected net present value of a project with phase-type distributed activity durations: An efficient globally optimal solution procedure. *Eur. J. Oper. Res.* **2018**, *267*, 16–22.
23. Rostami, S.; Creemers, S.; Leus, R. New strategies for stochastic resource-constrained project scheduling. *J. Sched.* **2018**, *21*, 349–365.
24. Lamas, P.; Demeulemeester, E. A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *J. Sched.* **2016**, *19*, 409–428.
25. Powell, W.B. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd ed.; John Wiley & Sons: Hoboken, NJ, USA, 2011.
26. Bertsekas, D.P. *Reinforcement Learning and Optimal Control*; Athena Scientific: Belmont, MA, USA, 2019.
27. Snauwaert, J.; Vanhoucke, M. A classification and new benchmark instances for the multi-skilled resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **2023**, *307*, 1–19.
28. Almeida, B.F.; Correia, I.; Saldanha-da-Gama, F. Modeling frameworks for the multi-skill resource-constrained project scheduling problem: A theoretical and empirical comparison. *Int. Trans. Oper. Res.* **2019**, *26*, 946–967.
29. Correia, I.; Saldanha-da-Gama, F. The impact of fixed and variable costs in a multi-skill project scheduling problem: An empirical study. *Comput. Ind. Eng.* **2014**, *72*, 230–238.
30. Li, H.; Womer, K. Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm. *J. Sched.* **2009**, *12*, 281–298.
31. Cai, X.; Li, K.N. A genetic algorithm for scheduling staff of mixed skills under multi-criteria. *Eur. J. Oper. Res.* **2000**, *125*, 359–369.
32. Attia, E.A.; Duquenne, P.; Le-Lann, J.M. Considering skills evolutions in multi-skilled workforce allocation with flexible working hours. *Int. J. Prod. Res.* **2014**, *52*, 4548–4573.
33. Torba, R.; Dauzère-Pérès, S.; Yugma, C.; Gallais, C.; Pouzet, J. Solving a real-life multi-skill resource-constrained multi-project scheduling problem. *Ann. Oper. Res.* **2024**, *338*, 69–114.
34. Snauwaert, J.; Vanhoucke, M. A solution framework for multi-skilled project scheduling problems with hierarchical skills. *J. Sched.* **2025**, *28*, 289–310.
35. Türk, A.; Yılmaz, Ö.F.; Özkök, M. Examining a multi-skilled project scheduling problem for shipyard industry through operational and tactical perspectives. *Comput. Oper. Res.* **2025**, *183*, 107150.
36. Li, H.; Zhu, H.; Zheng, L.; Xie, F. Software project scheduling under activity duration uncertainty. *Ann. Oper. Res.* **2023**, *338*, 477–512.
37. Liu, Y.; Ding, R.; Liu, S.; Wang, L. Multi-skilled project scheduling for high-end equipment development considering newcomer cultivation and duration uncertainty. *Systems* **2025**, *13*, 448.
38. Chen, X.; Thomas, B.W.; Hewitt, M. Multi-period technician scheduling with experience-based service times and stochastic customers. *Comput. Oper. Res.* **2017**, *82*, 1–14.
39. Annear, L.M.; Akhavan-Tabatabaei, R.; Schmid, V. Dynamic assignment of a multi-skilled workforce in job shops: An approximate dynamic programming approach. *Eur. J. Oper. Res.* **2023**, *306*, 1109–1125.
40. Bouajaja, S.; Dridi, N. A survey on human resource allocation problem and its applications. *Oper. Res.* **2017**, *17*, 339–369.

41. Wang, X.; Lu, S.; Qian, X.; Hu, C.; Liu, X. Dynamic scheduling of decentralized high-end equipment R&D projects via deep reinforcement learning. *Comput. Ind. Eng.* **2024**, *190*, 110018.
42. Yu, H.; Gu, W.; Tang, N.; Guo, Z. A deep reinforcement learning approach for dynamic job-shop scheduling problem considering time variable and new job arrivals. *Comput. Oper. Res.* **2026**, *185*, 107263.
43. Yang, H.; Wang, Z.; Gao, Y.; Zhou, W. Bi-objective multi-mode resource-constrained multi-project scheduling using combined NSGA II and Q-learning algorithm. *Appl. Soft Comput.* **2024**, *152*, 111201.
44. Xu, M.; Mei, Y.; Zhang, F.; Zhang, M. Learn to optimise for job shop scheduling: A survey with comparison between genetic programming and reinforcement learning. *Artif. Intell. Rev.* **2025**, *58*, 160.
45. Bertsekas, D.P.; Tsitsiklis, J.N. *Neuro-Dynamic Programming*; Athena Scientific: Belmont, MA, USA, 1996.
46. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2018.
47. Bertsekas, D.P.; Tsitsiklis, J.N.; Wu, C. Rollout algorithms for combinatorial optimization. *J. Heuristics* **1997**, *3*, 245–262.
48. Bertsekas, D.P.; Castanon, D.A. Rollout algorithms for stochastic scheduling problems. *J. Heuristics* **1999**, *5*, 89–108.
49. Kolisch, R.; Sprecher, A. PSPLIB—A project scheduling problem library. *Eur. J. Oper. Res.* **1996**, *96*, 205–216.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.