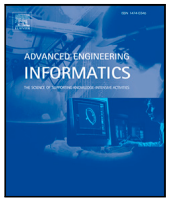
Advanced Engineering Informatics 58 (2023) 102209

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei



Full length article

When architecture meets RL+EA: A hybrid intelligent optimization approach for selecting combat system-of-systems architecture

Yang Huang [a], Aimin Luo [a,*], Tao Chen [a], Mengmeng Zhang [a], Bangbang Ren [a], Yanjie Song [b]

[a] Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha, China
[b] College of Systems Engineering, National University of Defense Technology, Changsha, China

ARTICLE INFO

Keywords:
SoSs
Architecture
Rule-based
Mission planning
Genetic algorithm
Task oriented
DQN

ABSTRACT

In recent decades, various domains such as military, aerospace and supply chain are becoming increasingly complex, forming system-of-systems (SoSs). The effectiveness of SoSs is mainly determined by their architecture, thus selecting proper architecture for SoSs is crucial to maximize their effectiveness. While most studies focus on specific issues, such as mission planning and SoSs meta-architecture selection, few papers deal with the two issues together. This paper introduces the task oriented SoSs architecture selection problem (TSASP) and proposes an combination of reinforcement learning (RL) and evolutionary algorithm (EA) (CRE) to address this problem. CRE comprises two layers: the inner layer utilizes a DQN-based algorithm to solve the mission planning problem, and the outer layer with improved genetic algorithm optimizes the SoSs architecture selection based on the output of inner one. A synthetic air and missile defense (AMD) example is conducted to test the effectiveness of proposed method, and results show that CRE can improve the mission planning effect by select a feasible architecture.

1. Introduction

A System-of-systems (SoSs) is defined as a network consisting of various independent constituent systems that collaborate to provide capabilities beyond what the individual systems could achieve alone [1]. Examples of typical SoSs in real life include air transportation SoSs, cyber–physical SoSs, spacecraft SoSs, combat SoSs, and C4ISR SoSs [2–4]. The Combat System-of-systems (CSoSs) is a typical SoSs comprised of various monolithic weapon platforms [5,6].

Inspired by civil engineering, the concept of architecture has been introduced to the domain of System of Systems (SoS) engineering. U.S. Department of Defense Architecture Framework (DoDAF) is widely used to design SoS architecture [7]. Some researchers also encode a set of decisions to represent architecture [8].

Traditional system design methods do not directly consider the impact of resource combinations on mission planning, but in fact the integration of resources will directly affect the effectiveness of mission planning, this is because different resources will be subject to the limitations of the communication capacity or the scope of the role of the conditions of the mission cannot be synergized, which leads to the system of the combat effectiveness of the situation is not ideal. The research problem of this paper focuses on how to select the optimal combination of resources to make the mission planning program the most effective.

In the actual use of cases, for example, air defense and anti-missile combat system in the number of information-guided signaling channels is limited, only the first signaling channel to the resources of the docking situation can be arranged before mission planning, so how to allocate these signaling channels is a system-level problem.

Designers must consider the mission planning effect as the goal of architecture selection [9]. Regardless the representation of architecture, there are many possible solutions for architecture design. Adopting a proper CSoS architecture can improve the performance of the CSoS. CSoS architecture selection problem concerns selecting appropriate combat units to execute given mission with best efficiency.

Numerous research papers have addressed individual problems in SoSs design, such as SoSs architecture design, command and control organizational structure optimization, and mission planning optimization et al. [5,10,11], but few have focused on the combined optimization

* Corresponding author.
E-mail addresses: huangyangnudt@163.com (Y. Huang), amluo@nudt.edu.cn (A. Luo), kd_chentao@163.com (T. Chen), 18670381635@163.com (M. Zhang), renbangbang11@nudt.edu.cn (B. Ren), songyj_2017@163.com (Y. Song).

https://doi.org/10.1016/j.aei.2023.102209
Received 6 June 2023; Received in revised form 7 September 2023; Accepted 2 October 2023

of the above problems. This paper aims to find a joint optimal CSoS architecture, which break down the task oriented SoSs architecture selection problem (TSASP) into two sub-problems: one is to find the best task schedule for mission execution, and the other is to select optimal portfolio of combat units. Both of them are NP-hard, and require much computation to find optimal solution. Since the mission planning problem is already NP-hard [12], this problem is also NP-hard.

Previous studies tried to solve the sub-problems separately sequentially due to the limitation of the solution efficiency [10,11,13]. With the application of reinforcement learning in the field of scheduling, the process of scheduling can be greatly accelerated, and thus make nested joint optimization possible. To decrease the computational complexity of the problem, in the first place, it is essential to properly arrange the hierarchy of mission planning problems and the System-of-systems (SoSs) architecture selection problems. And efficient algorithms should be designed to solve the mission planning problems, followed by using the resulting plan to guide the optimization of the SoSs architecture [2, 14].

This paper presents a novel hybrid CSoS architecture selection method that addresses two key challenges in this domain. To achieve this, we propose a two-layer nested optimization algorithm that considers the joint optimization of multiple objectives. Additionally, we leverage the power of Deep Reinforcement Learning (DRL) to solve the mission planning problem, thereby reducing the time cost of searching for the optimal solution. By moving the search process to the training stage, our approach enables the trained model to produce results in a much smaller time, significantly improving search efficiency.

Contributions of the paper can be summarized as follows:

- Firstly, it proposes an optimization model for the System-of-systems (SoSs) architecture selection problem. By integrating the SoSs architecture selection process with the mission planning process, the proposed model offers a practical solution to the task oriented SoSs architecture selection problem.
- Secondly, this research introduces a genetic algorithm that is specifically designed through an encoding scheme, crossover operator, variation operator, and fitness function tailored for the proposed problem.
- Lastly, this study proposes a mission planning algorithm based on Deep Q-Networks (DQN). This algorithm offline learns task selection rules and applies the acquired knowledge to expedite the online mission planning process while ensuring solution effectiveness.

Overall, this research contributes to the field of SoSs architecture selection by proposing an integrated model that considers system portfolio constraints, a genetic algorithm that is customized to the problem characteristics, and a mission planning algorithm that utilizes DQN to speed up the scheduling process.

The rest of paper is organized as follows: Section 2 introduces the related work. Section 3 describes the SoSs architecture selection model. Section 4 proposes the task oriented SoSs architecture selection algorithm. Section 5 experimentally verifies the effectiveness of the proposed algorithm. Section 6 summarizes the content of the paper and gives future research directions.

2. Related work

Most definitions of SoSs originate from Maier's seminal work [1], where they described SoSs as a set of components that can be considered individually as systems, these components are independently operated and managed, and the components are usually able to emerge through interactions between them with capabilities that are not available to the individual component members.

2.1. SoSs architecture selection method

The selection of SoSs architecture methods can be categorized into several aspects, such as those based on meta-SoSs architecture theory and model-based approaches. The summary of SoSs architecture selection method is shown in Table 1.

In situations where constraints are relatively scarce, SoSs designers typically opt for an optimal meta-architecture initially, and subsequently modify the specific SoSs architecture as additional constraints arise. Lesinski et al. [15] have proposed a fuzzy genetic-based method that facilitates the generation and evaluation of firepower capability meta-SoSs architecture for non-linear sites. This methodology generates, evaluates, and selects the system meta-SoSs architecture in the system through coupled executable models. Dagli's research group [16] utilized a genetic algorithm (GA) to select an SoSs meta-SoSs architecture based on fuzzy evaluation of performance, flexibility, and robustness. This approach enabled the development of an SoSs architecture tool that can be used to define, formulate, and solve a wide range of socio-technical problems [23–25].

When domain models are used to model architecture, there are numerous optimization methods available for selecting the appropriate System of Systems (SoSs) architecture. Purohit et al. [17] proposed a model-based approach for SoSs architecture and integration that considers system-level dependencies. This approach utilizes inter-layer and intra-layer dependency matrices and combines stochastic optimization with heuristics to align modules in different layers. Problematic interactions that may cause issues during system integration are identified, and integration is achieved accordingly. However, uncertainties in individual interconnected systems can lead to significant risks of cascading failure modes. To address this issue, Davendralingam et al. [18] proposed a robust optimization approach that models the hierarchy of operationally interdependent systems as nodes on a cohesive network. Wang et al. [19] presented an optimal search algorithm for the SoSs architecture space of equipment with uncertain capabilities from a capability perspective. Sanduka et al. [20] investigated a model-based system development approach with real-time requirements and SoSs architecture model, timing requirements, and extensions to the DoDAF and MODAF (UPDM) unified outline files extensions. They used mixed integer linear programming (MILP) for optimization to meet real-time requirements and injected the optimization results back into the UPDM model. Kalawsky et al. [21] proposed an SoSs architecture selection approach called Concise Modeling, which is used in conjunction with SoSs architecture models to speed up the exploration and analysis of SoSs architectures. However, the metrics optimized by these research species SoSs architecture selection models are static nonfunctional or functional metrics, with little evaluation of task execution effectiveness. Accurate task-oriented performance measurement of the system remains a challenge.

Organizational design serves as an intermediate link between System of Systems (SoSs) architecture and mission planning and is a crucial aspect of architectural design. Meirina et al. [22] have proposed a normative approach and computational methods to evaluate the effectiveness and efficiency of command and control (C2) organizations. Their evaluation process is based on a normative model of team and individual decision-making in a C2 environment, which is quantitatively represented by the organization, component systems, and tasks. This approach provides valuable insights into the design of task oriented SoSs architectures.

Levchuk et al. [13] have proposed a method for designing task-based policies and heterogeneous organizational structures by exploring information/command delivery and processing in organizations. Their approach models the organization as a coupled 2-network structure, where the distribution and routing of information and command processing determine the organizational policy, and the information and command networks (including their topology, communication costs, and capacity constraints) determine the organizational structure.

Table 1
Summary of SoSs architecture selection method.

Reference	SoSs model	Optimization objective	Algorithm	C2 consideration
Lesinski, et al. [15]	Meta-SoSs architecture	Firepower capability	Fuzzy genetic-based method	N
AI-Amin, et al. [16]	Meta-SoSs architecture	Performance, flexibility, robustness	Genetic algorithm	N
Purohit, et al. [17]	Model-based SoSs architecture	Inter-layer and intra-layer dependency matrices	Stochastic optimization with heuristics	N
Davendralingam, et al. [18]	Network model	Robustness	Robust optimization approach	N
Wang, et al. [19]	Super-network model	Maximized expected rewards and minimized cumulative costs	Greedy search algorithm	Y
Sanduka, et al. [20]	UML and SysML model	Real-time requirements	Mixed integer linear programming	N
Kalawsky, et al. [21]	SysML model	Expedite architecture exploration and analysis	Concise Modeling optimization method	N
Meirina, et al. [22]	Normative model	Effectiveness and efficiency	/	Y
Levchuk, et al. [13]	Network model	Topology, Communication costs, and capacity constraints	Exploring information transfer and processing in organizations	Y
Levchuk, et al. [10]	Network model	Minimize the network construction cost	Heuristic solution	Y

This method provides valuable insights into the flow and landing points of SoSs architecture selection.

In addition, Levchuk et al. [10] have considered complex, unpredictable, and highly interdependent tasks that would overload the DMs responsible for supervision. They propose a method for designing a networked organizational structure that allows for horizontal authorization, communication, and control flow between layers, and sharing of component systems. This approach provides valuable insights into the design of SoSs architectures to cope with complex task scenarios.

Overall, these studies demonstrate the importance of organizational design in the development of effective and efficient SoSs architectures. By utilizing normative approaches, computational methods, and network structures, designers can gain valuable insights into the design of task oriented and complex SoSs architectures [26–30].

2.2. Mission planning method

The summary of mission planning method is shown in Table 2. In the domain of mission planning, Levchuk et al. [11] employed the multidimensional dynamic list planning (MDLS) algorithm to tackle the mission planning problem. Cheng et al. [18] addressed a more practical scenario and integrated the MDLS algorithm with influence networks to devise a novel priority rule list. Belfares et al. [31] utilized a multi-objective tabu algorithm to resolve uncertain military mission planning issues. Yu et al. [32] investigated the mission planning problem within a holonic organization, considering the existing System of Systems (SoSs) architecture situation. The proposed nested genetic algorithm was able to cope with the military mission planning problem that is subject to uncertainty interference. These studies have contributed to the advancement of mission planning by proposing innovative algorithms and considering various practical scenarios.

There are various types of research in reinforcement learning aimed at solving scheduling problems. The key aspect common to these studies is the proposal of a suitable state representation and corresponding actions, often utilizing rules as actions. For instance, Zhao et al. [33] proposed a dynamic job-shop scheduling algorithm based on Deep Q Network (DQN) to enhance the performance of adaptive scheduling algorithms in dynamic manufacturing environments. Similarly, Luo et al. [34] proposed a two-level deep Q network (THDQN) for online rescheduling, optimizing total weighted delay and average machine utilization. The higher-order DQN serves as a controller that determines a temporary optimization goal for the lower-order DQN. The lower DQN acts as an executor, selecting a suitable scheduling rule to achieve the given goal. Tasse et al. [35] designed a meaningful and compact state representation scheme and a novel reward function for the job-shop scheduling problem, closely related to the sparse make-span minimization criterion used by the combinatorial optimization problem approach. Li et al. [36] proposed a hybrid deep Q-network for the dynamic flexible job shop scheduling problem, using a shop flow state model, decision points, generic state features, genetic programming-based action spaces, and reward functions. Zeng et al. [37] addressed the dynamic job-shop scheduling problem and proposed a flexible hybrid framework, using an attention mechanism as a graph representation learning module for state feature extraction and a deep Q-network with double-decision replay priority and noise network. Du et al. [38] studied the flexible job shop scheduling problem with time-of-use tariff constraints and proposed a hybrid distribution estimation algorithm with a deep Q-network for multi-objective optimization, selecting state features to describe the scheduling situation in the deep Q-network component, defining nine knowledge-based action refinement scheduling schemes, and designing rewards based on these two objectives.

As this paper endeavors to optimize both the System of Systems (SoSs) architecture and the mission planning problem, it is recommended that a reinforcement learning (RL) algorithm be employed for the mission planning process, while a standard Evolutionary Computation (EC) algorithm has the potential to effectively address the architecture selection problem.

3. Problem formulation

This paper presents a novel approach to the task-oriented selection of System of Systems (SoSs) architecture, referred to as the Task-Oriented SoSs Architecture Selection Problem (TSASP). The TSASP aims to design the optimal SoSs architecture that maximizes the mission planning performance while adhering to the cost constraints of SoSs architecture construction.

The initial subsection of this section outlines the methodology for representing the SoSs architecture, while the subsequent subsection presents a precise mathematical formulation for addressing the TSASP problem.

Table 2
Summary of mission planning method.

Reference	Research problem	Optimization objective	Method	Problem scenario
Levchuk, et al. [11]	Mission planning	Mission completion time	MDLS	Static
Cheng, et al. [18]	Flexible job-shop scheduling problem	Mission completion time	MDLS with influence networks	Static
Belfares, et al. [31]	Component systems allocation	Maximize the usage of component system	Tabu Search	Static & uncertainty
Yu, et al. [32]	Mission planning problem under a holonic organization	Mission completion time	Nested genetic algorithm	Static & uncertainty
Zhao, et al. [33]	Dynamic job-shop scheduling	Minimize tardiness	DQN	Dynamic
Luo, et al. [34]	Online rescheduling	Total weighted delay and average machine utilization	Two-level deep Q network (THDQN)	Dynamic
Tasse, et al. [35]	Job-shop scheduling problem	Minimize make-span	Reinforcement learning	Static
Li, et al. [36]	Dynamic flexible job shop scheduling problem with insufficient transportation component systems	Minimize the make-span and total energy consumption	Hybrid deep Q network	Dynamic
Zeng, et al. [37]	Dynamic job-shop scheduling	Minimize make-span	DQN with an attention mechanism	Dynamic
Du, et al. [38]	Flexible job shop scheduling problem	Maximize completion time and minimize total electricity price	Hybrid multi-objective optimization algorithm of estimation of distribution algorithm and DQN	Static

3.1. SoSs architecture representation

This paper discusses the system of systems (SoSs) architecture as a portfolio of component systems that work together to complete a mission. The main constraint is the cost of selecting these component systems, while the goal is to minimize the make span of mission execution. To provide a visual representation of a task oriented SoSs architecture, Fig. 1 presents a schematic diagram. Drawing inspiration from the information age combat model, the SoSs architecture consists of three basic elements: sensor (S), decider (D), and influencer (I), with each component system representing one of these elements and providing relevant capabilities to complete the mission.

To successfully complete a mission, it is necessary to decompose it into a series of tasks, each of which is completed through a sequence of operations. Each operation requires a specific component system that meets the requirements of that operation. For example, in Fig. 1 the 'S' element is suitable for executing operation 1 of task N, while the 'I' element cannot be chosen to finish this operation due to its lack of 'S' capability. The mission execution effect is heavily influenced by the execution logic of tasks, which are executed by different component systems but must adhere to the task execution logical order. Therefore, different component system portfolios will lead to different combat mission effects.

As the mission planning process is closely related to the SoSs architecture, the goal is to design an optimal SoSs architecture that fits the mission planning process well. Based on the above analysis, the variables of SoSs architecture selection should be the evolution of component systems, with corresponding costs associated with each chosen component system. The objective of SoSs architecture selection is to minimize the task execution time of mission planning while adhering to the constraint of SoSs architecture construction cost.

From the analysis presented above, it can be inferred that the problem at hand can be transformed into a two-layer optimization problem. The inner layer of this problem deals with optimizing the task plan arrangement, while the outer layer deals with optimizing the design of the System of Systems (SoSs) architecture. The outer layer is dependent on the inner layer optimization results, owing to the coupling of the SoSs architecture selection effect and the task execution effect. If the SoSs architecture is optimized in parallel with the mission planning process, instead of following a layered strategy, it may lead to inconsistent optimization goals. Therefore, the SoSs architecture selection problem is modeled as a two-layer optimization problem, which can reflect the improvement of the task execution effect by the SoSs architecture selection and optimize the system-level performance indicators, making the mission planning and SoSs architecture selection processes interrelated and independent.

3.2. Mathematical formulation for architecture selection

In this section, we present the mathematical model for the SoSs architecture selection problem. To ensure scientific rigor, we first outline the hypothesis conditions that transform the complex practical problem into a well-defined scientific problem. We then provide a comprehensive list of all parameters and variables involved in the SoSs architecture selection model. Table 3 contains the mathematical symbols and descriptions of the SoSs architecture selection model.

The TSASP problem aims to optimize the composition of the system in the SoSs architecture in order to achieve the highest possible effectiveness in completing tasks. To account for the relationship between SoSs architecture and mission planning,

While many external conditions can affect the result of the SoSs architecture selection, the model is based on several assumptions to avoid the influence of perturbations. These assumptions include:

(1) Knowledge of all operations for each task;
(2) Independence of tasks from each other;
(3) Non-interruptibility of tasks once started;
(4) Exclusion of component system failure;
(5) No distinction between component systems that can perform the same operation;
(6) Component systems can immediately execute another operation after completing the current operation.

In typical SoSs architecture, multiple tasks are addressed, each corresponding to a set of operations that are completed by component systems. Component systems can execute certain operations based on their capability type, and each operation specifies a particular type of component system to execute. Multiple component systems are required to execute a task in sequence to achieve the task's established goal.

Suppose there are n mutually independent tasks that constitute a set of tasks $T = \{T_1, T_2, \ldots, T_n\}$. Each task is represented by a sequence of

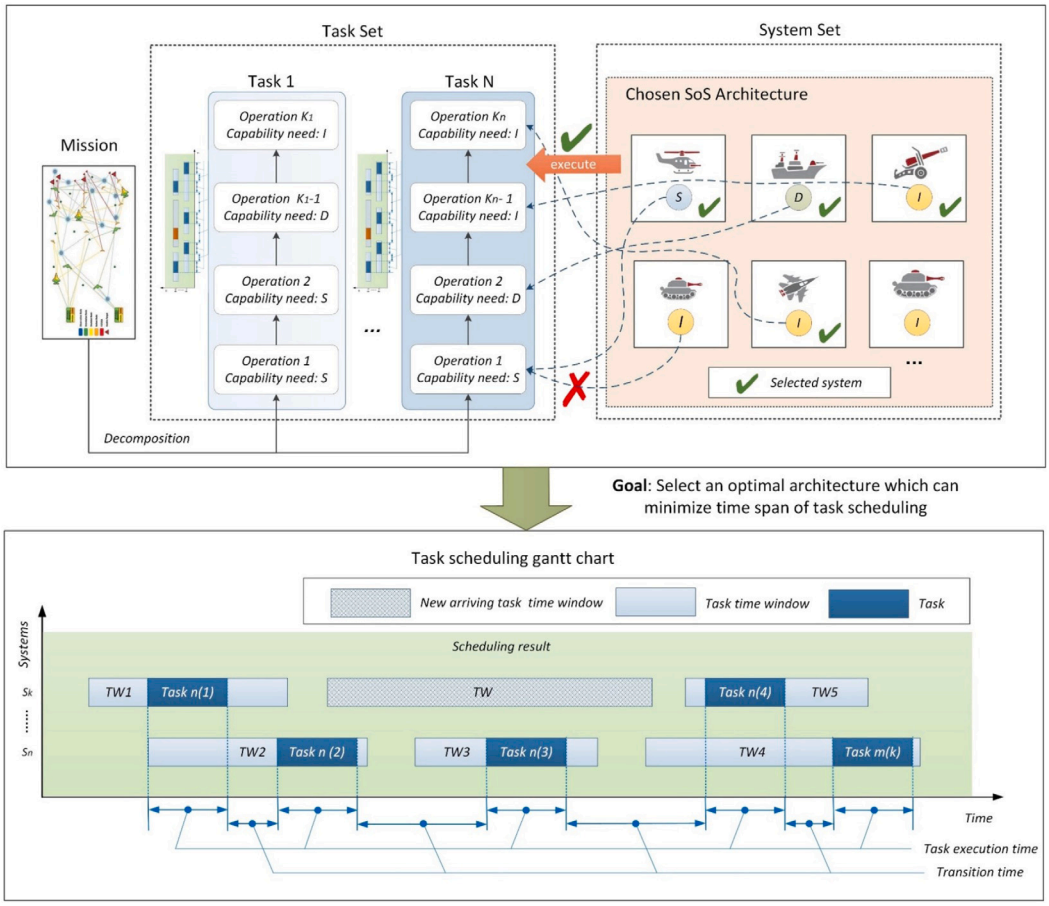


Fig. 1. Schematic diagram of the task oriented SoSs architecture.

Table 3
Symbols and variables.

Parameters	Description
T	Task set
S	Component system set
i	Task serial number
j	Component system serial number
$S_{i,a}$	The component system set which can execute the ath operation of the ith task
n_i	Number of operations for the ith task
$r_{i,a,k}$	Release time for the ath operation of the ith task by component system k
$d_{i,a,k}$	Termination time for the ath operation of the ith task by component system k
$w_{i,a,k}$	Waiting time on component systems for the ath operation of the ith task
$p_{i,a,k}$	Execution time on the component systems for the ath operation of the ith task
N_T	Number of tasks
N_S	Number of component system
$t_{i,a,k}$	Time required to perform operation a of task i by component system
f_k	Cost for chosen component system k
F	Total cost constrains
Variables	**Description**
$c_{i,a,k}$	Completion time on the component system for the ath operation of the ith task
$y_{i,a,k}$	The number of operation that component system k executes
H	Set of all the chosen component systems
Decision variables	**Description**
h_k	Decision variable for indicate component system k in the SoSs
$x_{i,a,k}$	Decision variable for indicate component system selected to perform the ath operation of the ith task
$s_{i,a,k}$	Start time on component system for the ath operation of the ith task

n_i operations, each of which has a release time $r_{i,a,k}$ and a termination time $d_{i,a,k}$, where i refer to ith task, a refer to the ath operation, k refer to kth component system. The operations within a task T_i must be completed within the specific release and termination times. The operations must be executed in order from first to last, with no overlap between them. Some of operations may have a waiting period $w_{i,a,k}$ before the operation can be started, following the completion of the previous operation. Each operation must be completed on a specific type of component system. The objective of our scheduling approach is to determine an appropriate execution plan for these tasks.

The component systems selected by the System of Systems (SoSs) are represented by the following equation:

$$h_k = \begin{cases} 1, if\ component system\ k\ is\ choosen\ by\ SoSs \\ 0, otherwise \end{cases} \tag{1}$$

For the problem of mission planning, our focus is solely on the impact of task completion time, also known as make span. To this end, we have developed a corresponding mathematical model that considers this crucial factor.

The objective function of our model is as follows:

$$\begin{aligned} &\Psi(H) = \min\left\{\max\left(c_{i,a,k}\right)\right\}, \\ &\forall i \in T, k \in S, a \in \left\{1, \ldots, n_i\right\}, h_k = 1 \end{aligned} \tag{2}$$

In the mission planning problem, the primary objective is to minimize the task execution time, also known as the make span. This is achieved by minimizing the largest task completion time through the objective function. Specifically, i refer to ith task, a refer to the ath operation, k refer to kth component system, $c_{i,a,k}$ means the completion time of the ath operation in the ith task. The goal is to identify an optimized system-of-systems (SoSs) architecture that can significantly reduce the mission execution time.

To achieve this objective, there are certain constraints that must be considered. These constraints are critical in ensuring that the proposed SoSs architecture is feasible and practical.

(1) Operation a must be executed and completed within the designated execution window for task i.

$$\begin{aligned} &r_{i,a,k} \le s_{i,a,k}, c_{i,a,k} \le d_{i,a,k}, \\ &\forall i \in T, k \in S_{i,a}, a \in \left\{1, \ldots, n_i\right\}, h_k = 1 \end{aligned} \tag{3}$$

If component system k is unable to execute operation a, the waiting and execution times will be considered infinite.

$$\begin{aligned} &w_{i,a,k} = +\infty, p_{i,a,k} = +\infty \\ &if\ resource\ j\ cannot\ do\ operation\ a, \\ &\forall i \in T, a \in \left\{1, \ldots, n_i\right\} \end{aligned} \tag{4}$$

Additionally, the operation must be completed after the preceding operation to initiate the execution process.

$$\begin{aligned} &c_{i,a,k} = s_{i,a,k} + p_{i,a,k} + w_{i,a,k}, \\ &\forall i \in T, k \in S_{i,a}, a \in \left\{1, \ldots, n_i\right\}, h_k = 1 \end{aligned} \tag{5}$$

(2) To guarantee that subsequent activities are not initiated until the completion of the previous activity, we have:

$$\begin{aligned} &c_{i,a,k} \le s_{i,a+1,k'}, \\ &\forall i \in T, a \in \left\{1, \ldots, n_i\right\}, k \in S_{i,a}, k' \in S_{i,a+1} \end{aligned} \tag{6}$$

(3) In order to ensure that each activity is executed by a single component system, we have implemented a strict protocol.

$$\sum_{k \in S} x_{i,a,k} = 1, \forall i \in T, k \in S_{i,a}, a \in \left\{1, \ldots, n_i\right\} \tag{7}$$

(4) To prevent component systems from performing multiple activities simultaneously, we have implemented a system that enforces a one-activity-at-a-time rule.

$$\begin{aligned} &x_{i,a,k} \cdot c_{i,a,k} \le x_{i',a',k} \cdot s_{i',a',k}, \\ &\forall i \in T, i' \in T, k \in S_{i,a}, k \in S_{i',a'}, a \in \left\{1, \ldots, n_i\right\}, \\ &a' \in \left\{1, \ldots, n_{i'}\right\}, h_k = 1, x_{i,a,k} = 1, x_{i',a',k} = 1, \\ &y_{i,a,k} = m, y_{i',a',k} = m + 1, m \in N^+ \end{aligned} \tag{8}$$

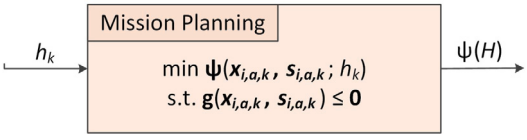


Fig. 2. Inner layer mission planning problem.

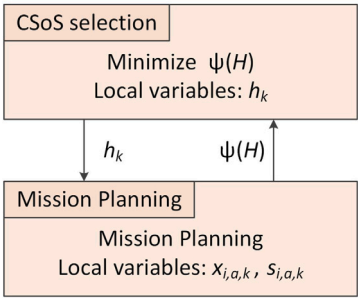


Fig. 3. Outer layer SoSs architecture selection problem.

(5) To ensure that the cost constraint is not breached during the selection process for the SoSs architecture, we have implemented rigorous measures.

$$\sum_{k=1}^{N_S} h_k \cdot f_k \le F, h_k \in \{0,1\}, \forall k \in S \tag{9}$$

(6) The value range of the decision variables.

$$h_k \in \{0,1\}, k \in S_{i,a} \tag{10}$$

$$x_{i,a,k} \in \{0,1\}, \forall i \in T, k \in S_{i,a}, a \in \left\{1, \ldots, n_i\right\} \tag{11}$$

$$y_{i,a,k} \in \{0,1\}, \forall i \in T, k \in S_{i,a}, a \in \left\{1, \ldots, n_i\right\} \tag{12}$$

$$s_{i,a,k} \in N, \forall i \in T, k \in S_{i,a}, a \in \left\{1, \ldots, n_i\right\} \tag{13}$$

4. Solution approach for SoSs architecture selection

This paper transformed the TSASP problem into a two-layer optimization problem containing two sub-problems: mission planning problem and SoSs architecture selection problem, which in essence is a Parallel machine scheduling problem (PMP) with constraints and a Set Covering Problem (SCP), both of which are NP-hard. As a result, the TSASP problem is also an NP-hard problem. To address this issue, this paper proposes an SoSs architecture selection optimization method based on a learning-based mission planning approach. The proposed method consists of two parts: an outer layer SoSs architecture selection algorithm and an inner layer DQN-based mission planning algorithm. The schematic diagram of the solution framework are shown in Fig. 4. The outer layer employs genetic algorithm to optimize the SoSs architecture solution, while the inner layer uses a reinforcement algorithm to solve the mission planning problem. The two layers of algorithms work cooperatively to solve the TSASP problem.

4.1. Two-layer optimization model transformation of the problem

As Fig. 2 shows, the inner layer optimization problem is essentially a mission planning problem, where the inputs are the task list and the current component system list. The goal of the mission planning problem is to find the task scheduling scheme that satisfies the constraint of maximum task execution effectiveness. Each component system has a set of attributes, such as time window and capabilities.

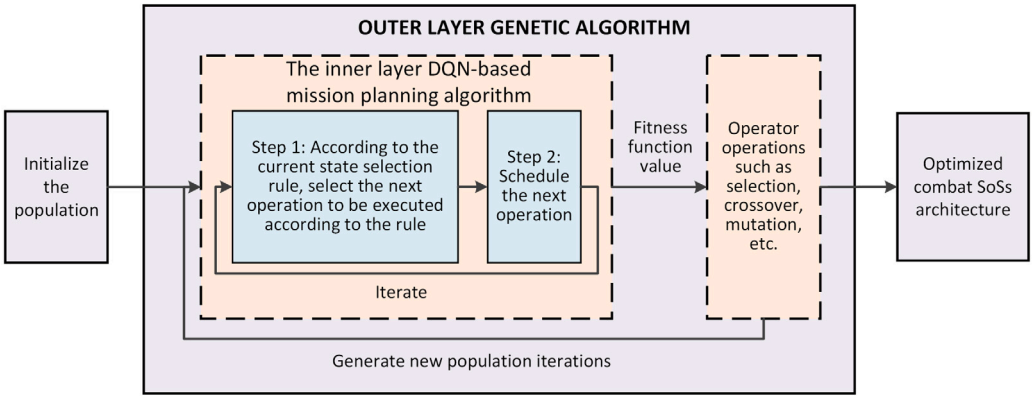


Fig. 4. Schematic diagram of the solution framework.

The optimization problem aims to arrange the component systems for each operation to be executed, which needs to satisfy a series of constraints. The sequence of operations to be executed needs to satisfy the timing relationship in the operation. Only when the previous operation has been executed can the current operation be executed, and the component systems can only execute one operation at the same time.

Fig. 3 shows the outer layer of the optimization problem: the SoSs architecture selection problem, where the input is the cost constraint of the SoSs architecture construction, and the goal is to maximize the task execution effectiveness, which means the SoSs architecture effectiveness is maximized.

Fig. 4 shows the proposed solution framework, which comprises two layers. The outer layer is used to solve the SoSs architecture selection optimization problem, while the inner layer is used to solve the mission planning problem based on the current SoSs architecture. This approach allows for quick feedback for the outer layer.

For small-scale problems, dynamic programming and exact solution algorithms are solvable. However, as the problem size increases, the computational cost becomes prohibitively high. Therefore, heuristic algorithms and evolutionary algorithms are more adaptable to large-scale task situations. Reinforcement learning is used for the inner optimization problem as it speeds up the solution and makes nested optimization acceptable in terms of time consumption.

The RL-based method proposed in this paper is interpretable, thereby increasing the stability of the algorithm solution. For the outer layer optimization problem, genetic algorithm are chosen as the exact solution method cannot be used and reinforcement learning would lead to excessive training costs. However, the effectiveness of the heuristic algorithm depends largely on the selected heuristic strategy.

4.2. Outer layer SoSs architecture selection algorithm

The focus of this study is on addressing the design problem of the System of Systems (SoSs) architecture structure. To this end, the authors propose a novel outer layer evolution algorithm, referred to as the SoSs architecture Evolution Algorithm (SAEA), which is designed to optimize the SoSs architecture. The SAEA algorithm is evaluated through multiple iterations, with each iteration requiring the determination of the adaptation value for each chromosome. This value is obtained by executing the mission planning algorithm based on the Deep Q-Network (DQN) algorithm. Essentially, the adaptation degree for each chromosome is a measure of the task execution effectiveness under the corresponding SoSs architecture configuration.

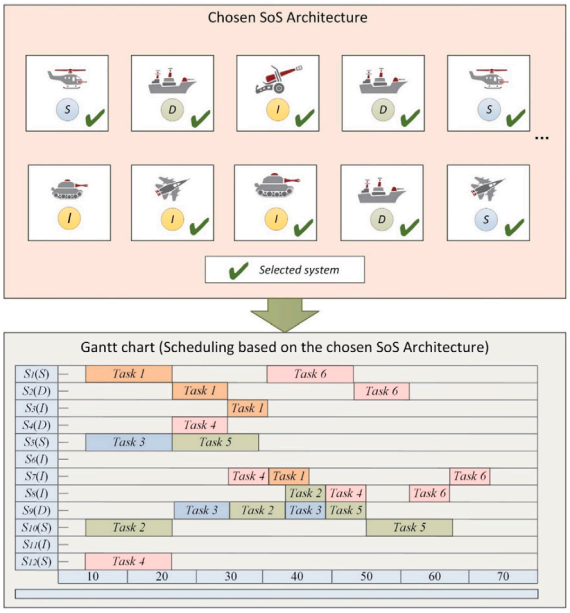


Fig. 5. Mission planning based on chosen SoSs architecture.

4.3. Inner layer DQN-based mission planning algorithm

The mission planning problem poses a significant challenge as it requires simultaneous decision-making on execution component systems and start execution time for each operation. Additionally, the problem's complexity is compounded by the presence of waiting time. The mission planning problem based on the chosen SoSs architecture are shown in Fig. 5.

The inner layer algorithm of the SoSs architecture selection algorithm which is shown in Algorithm 1 addresses the mission planning problem, with inputs including a list of tasks and component systems. The latter refers to the architecture corresponding to the chromosomes in the population of the outer layer genetic algorithm, which can be decoded. The fitness function of the outer layer algorithm is determined by the mission planning scheme, but the time required for solving the problem with the genetic algorithm remains significant. To expedite the solution process, this section proposes a mission planning algorithm based on DQN, which leverages offline learning to enhance solution efficiency without compromising effectiveness [39,40].

Algorithm 1: SoSs Architecture Evolution Algorithm (SAEA)

Input: number of genetic generations N, number of populations NP, trained DQN network, crossover probability α, mutation probability β, list of tasks T, component systems S

Output: Optimal SoSs architecture

Initialization of algorithm parameters;
Initializing the population;
Load the trained DQN network;
Generate initial population P_0;
for $t = 1, ..., N$ **do**
  if $rand() \leq \alpha$ **then**
    $P(t+1) \leftarrow Crossover(P(t), L, \alpha)$
  if $rand() \leq \beta$ **then**
    $P(t+1) \leftarrow Mutation(P(t+1), \beta)$
  Execute the improved DQN algorithm for each individual in the population to obtain the planning scheme;
  Calculate the fitness function value $Fit(t+1)$ for each planning scenario;
  $P(t+1) \leftarrow roulettewheel\left(P(t+1), Fit(t+1), N_P\right)$;
  $local\_best\_fit, local\_best\_indi \leftarrow Fit(t+1)$;
  if $local\_best\_fit > global\_best\_fit$ **then**
    $global\_best\_fit, global\_best\_indi \leftarrow local\_best\_fit, local\_best\_indi$

To improve efficiency while maintaining solution quality, this paper proposes a mission planning algorithm based on DQN. The DQN-based mission planning algorithm addresses two key aspects. First, as tasks are continuously added, the mission planning scheme's hidden state information becomes nearly infinite, which can lead to the "dimensional explosion" problem. This issue is resolved by using the DQN method, which fits the neural network to the Q values. Second, to avoid the neural network from learning redundant information, feature vectors are extracted from the mission planning scheme, and the Q values corresponding to all operation selection rules in the current mission planning scheme state can be obtained after the neural network. This approach completes the evaluation of the state action once, ensuring that the algorithm is efficient and effective.

4.3.1. Interaction design of mission planning process in MDP

Markov models require that the problems being modeled adhere to the principle of Markovianity, which stipulates that if all historical information is incorporated into a state at a given moment, the decision made at the subsequent moment is solely dependent on the current state. When modeling the mission planning process as a scheduling operation at each decision step, the state at any given point is the planning solution for the previously scheduled operation, thereby ensuring that the decision solution is exclusively related to the current state. Consequently, the states of this mission planning process are Markovian, allowing for Markovian modeling.

A Markov process consists of a five-tuple $M = \langle S, A, P, R, \gamma \rangle$: where S represents the set of states of the environment, A represents the set of actions of the intelligence, P represents the state transfer probability, R represents the reward function, and γ is the discount factor.

The mission planning problem's MDP model is presented in Fig. 6. The Q-network is trained with a substantial amount of data, and the subsequent state is derived by selecting the appropriate action at each decision step. The component system allocation algorithm is utilized to assign component systems to the task sequence in that state, and the fitness obtained serves as the reward value for updating the Q-network. To prevent the training results from falling into local optima, the stored historical state transfer dataset is sampled using empirical replay. This enables the sampled data to be employed for network learning. Moreover, the update strategy of the target value function and the state value function is distinguished. The state value function is updated at each step of the state transfer, while the target value function is updated after a certain time step by assigning the parameters of the state value function to the target value function. This approach enhances the stability of the training process.

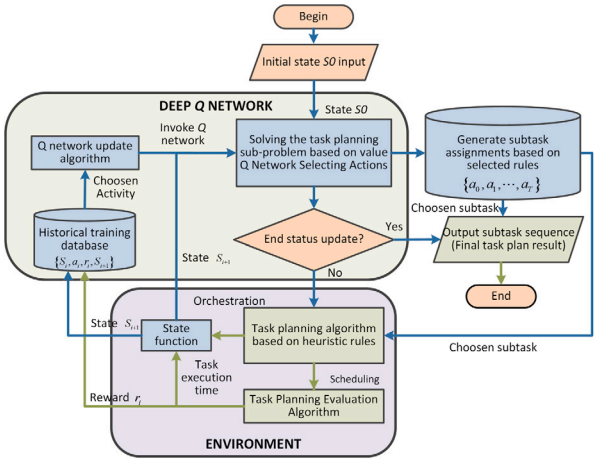


Fig. 6. MDP model for mission planning problem.

4.3.2. State space design based on the task execution state

The design of the state space is crucial in ensuring the reliability and effectiveness of the Q-network. A well-defined state space should express sufficient state information while minimizing the amount of information required. This allows the algorithm to converge and select action values that are consistent with the current actual state. To achieve this, state features that reflect the overall mission planning equilibrium should be selected. Additionally, the state space design should be able to generalize the characteristics of different types of mission planning problems.

In particular, the task execution efficiency of a component system is dependent on the state of the entire system. Therefore, the state space should be designed to capture the relevant characteristics of the component system and the overall system. By doing so, the trained neural network can effectively cope with growing tasks. In summary, the careful selection and design of the state space is essential in ensuring the Q-network's reliability and effectiveness in solving mission planning problems. The states are calculated and described in Table 4.

To calculate the states in Table 4, some auxiliary quantity should be calculate by the following equations. For example, the task execution

Table 4
Equations for calculating state.

State calculation equation	Variable description
$URS_{ave}(t) = \frac{\sum_{j=1}^{N_S} URS_j(t)}{N_S}$	Average utilization of component system s
$URS_{std}(t) = \sqrt{\frac{\sum_{j=1}^{N_S}(URS_j(t) - URS_{ave}(t))^2}{N_S}}$	The standard deviation of component system utilization
$CRT_{ave}(t) = \frac{\sum_{i=1}^{N_S} OCN_i(t)}{\sum_{i=1}^{N_S} n_i}$	Average completion rate of operations
$CRT_{ave}(t) = \frac{\sum_{i=1}^{N_S} CRT_i(t)}{N_S}$	Average task completion rate
$CRT_{std}(t) = \sqrt{\frac{\sum_{i=1}^{N_S}(CRT_i(t) - CRT_{ave}(t))^2}{N_S}}$	Standard deviation of task completion rate
$SGTl(t) = t - \min SLT_j$	The maximum value of component system idle time
$SGT_{ave}(t) = \frac{\sum_{j=1}^{N_S} SLT_j(t)}{N_S}$	Average of component system idle time
$SGT_{std}(t) = \sqrt{\frac{\sum_{j=1}^{N_S}(SLT_j(t) - SGT_{ave}(t))^2}{N_S}}$	The standard deviation of component system idle time
$SGTs(t) = t - \max SLT_j$	The minimum difference in component system idle time
$SGTla(t) = \frac{SGT_{ave}(t)}{SGTl(t)}$	Component system idle time average to a maximum ratio
$SGTsa(t) = \frac{SGTs(t)}{SGT_{ave}(t)}$	Minimum to average ratio of component system idle time
$SGTla_j(t) = \frac{SGTs(t)}{SGTl(t)}$	The ratio of minimum to maximum component system idle time

efficiency (utilization rate of component system) of component system j is:

$$URS_j(t) = \frac{\sum_{i=1}^{N_T}\sum_{a=1}^{OCN_i(t)} x_{i,a,j} \cdot p_{i,a,j}}{MCT_j(t)} \tag{14}$$

The completion rate of task i (completion rate of task) is:

$$CRT_i(t) = \frac{OCN_i(t)}{n_i} \tag{15}$$

In this study, the state space comprises variables that are associated with the utilization of the component system and the rate of task completion. These state features serve as an evaluation of the feasibility of the current planning solution. The definition of the action space is closely linked to the setting of the state space, and the two work together to facilitate the neural network in learning a more effective strategy. For instance, if the current average utilization of the component systems is low, the neural network should select an action that can enhance the average utilization of the component systems in the subsequent decision period. The analysis indicates that the action should be chosen in a way that the tasks to be executed in the next decision period are distributed more uniformly among the scheduled sequence of operations, thereby minimizing the average idle time between the execution of component systems. Similarly, if the current average task completion rate is low, the neural network should choose an action that can improve the average completion rate of the tasks corresponding to the mission planning solution in the next decision period. The analysis suggests that an action should be selected to execute the next operation corresponding to the task that has already been executed but has the lowest number of executed operations in the next decision period.

4.3.3. Action space design based on the task execution state

Each stage of mission planning is associated with a specific state, and for each state, a corresponding action must be designed. The design of the action space can be approached from two perspectives: first, by designing actions that enhance the average utilization of component systems, and second, by designing actions that improve the average completion rate of tasks.

Table 5
Single rule.

Rule serial number	Rule description
1	Select the operation that will result in the shortest wait time for component system s
2	Select the operation that can be executed at the earliest in the next step
3	Select the operation with the shortest execution time for the next step

This paper proposes a set of five rules, consisting of one single rule and four compound rules, aimed at selecting the optimal strategy for the next execution operation. The choreography of the mission planning scheme is optimized by learning these rules, which can be categorized as single rules or compound rules comprising multiple single rules (as presented in Table 5). The starting point for the design of these rules is the selection of the operation that can be executed as early as possible, depending on the situation.

Rule 1: Prioritizes the selection of the operation that minimizes the component systems' waiting time, and if the waiting time is the same, the one that can be executed earliest in the next step.

Rule 2: Selects the operation with the earliest executable time for the next step.

Rule 3: Chooses the operation with the shortest execution time in the next step, and if the execution time is the same, the one with the earliest executable time in the next step.

Rule 4: Prioritizes the operation with the earliest executable time for the next step, and if the executable times are the same, the one that minimizes the component systems' waiting time.

Rule 5: Selects the operation with the earliest executable time in the next step, and if the executable time is the same, the one with the shortest execution time.

In this model, the action space is defined based on the selection of rules in the current task execution state, and only one task is chosen at each decision step according to the selected rules. This approach optimizes the choreography of the mission planning scheme and ensures the efficient execution of the component systems.

4.3.4. Component system scheduling algorithm

After selecting an appropriate action for the current state, an operation is chosen for execution based on the rules corresponding to the selected action. The execution of this operation requires scheduling of the corresponding execution component systems and determining the execution time. To solve this problem, a component system scheduling algorithm has been designed in this section.

The scheduling of component systems involves selecting the appropriate systems and processing time for each operation to optimize the objective function value. To accomplish this, we propose a mission planning algorithm that utilizes a single operation scheduling approach. This approach schedules the next operation in the sequence after a suitable execution time has been determined for the previous operation. During the scheduling process, all constraints are checked, and a suitable execution time is determined only when the operation satisfies all constraints.

The component system scheduling algorithm (CSSA) is presented in Algorithm 2. The input of the algorithm is the operation corresponding to the action selected in the current state, and the output is the specific component system scheduling scheme for all operations of the current task.

The scheduling algorithm for the component system involves a constrained judgment process. Initially, the algorithm evaluates the available component systems that can execute a given operation. The operation can only be executed on component systems that match the

Algorithm 2: Component System Scheduling Algorithm(CSSA)

Input: current SoSs architecture, component system execution record Rec, task list T, current scheduling scheme $Plan$, component system list R

Output: Specific component system programming $plan^*$ for all operations in the next mission planning phase

```
Select the corresponding operation according to the rules corresponding to the selected action;
Set the initial execution moment t = inf, set the component system r_choosen = 0 for the initial execution operation;
for r = 1, ..., N_R do
    According to the task list T to get the current operation r corresponding to the large task;
    if Rec(task) ≤ t and Match component systems to the type of capability of the operation then
        Update operation execution time t = Rec(task);
        Update component systems for implementation operations r_choosen = r;
    else
        Go to the next component system for the operation;
    Update component system execution records Rec(task);
Add the operation to the Gantt chart according to the scheduling scheme to get the new scheduling scheme Plan*;
```

type required for completion. Subsequently, the algorithm schedules the operation at an appropriate location after ensuring that the previous operation has been completed. This necessitates an evaluation of whether the scheduled operation execution location has already completed the previous operation.

4.3.5. Incremental revenue-oriented reward design

After scheduling the operation using a component system, the reward can be calculated based on the new mission planning scenario generated and the scenario from the previous phase. In this paper's SoSs architecture, the main concern in the component system allocation problem is task completion time. Therefore, the reward value for each decision step is determined by the task completion time. The timely reward reflects the short-term effect of the action on the scheduling scheme, while the cumulative reward value reflects the long-term effect. By prioritizing the cumulative reward value, the reinforcement learning algorithm can learn the action selection strategy that yields the highest impact on task completion results.

The minimized maximum completion time of the task is $MT = \min\left\{\max\left(c_i\right)\right\}$. Therefore, the design reward value is $r_t = \alpha \cdot C_i + (1-\alpha) \cdot MT$. To reasonably evaluate the timely rewards of the actions, define the component system effectiveness under the condition (state k) that the current number of operations is:

$$Q(k) = \frac{1}{N_M(k)} \sum_{h=1}^{N_M} \frac{\sum_{i=1}^{N_J(k,i)} \sum_{j}^{N_{O_i}} T_{ijh}}{MT(k)} = \frac{T}{N_M(k) \cdot MT(k)} \tag{16}$$

where $Q(k)$ denotes the average efficiency of component system utilization, $N_m(k)$ denotes the number of component systems used at state k, $N_J(k,i)$ denotes the set of tasks scheduled at state k, N_{O_i} denotes the number of operations corresponding to task i, $MT(k)$ denotes the total time of task completion at state k, and T denotes the total working time of component systems.

The prompt reward corresponding to the action of the state k is:

$$r_k = Q(k) - Q(k-1) \tag{17}$$

The cumulative award value can be calculated according to the following equation:

$$\begin{aligned} R &= \sum_{k}^{K} r_k = \sum_{k=1}^{K} (Q(k) - Q(k-1)) \\ &= Q(1) - Q(0) + Q(2) - Q(1) \\ &+ \cdots + Q(k) - Q(k-1) \\ &= Q(k) - Q(0) = Q(k) \\ &= \frac{T}{N_M(k) \cdot MT(k)} \end{aligned} \tag{18}$$

4.3.6. Value function design for large-scale state

As the value function is dependent on the reward value, it can be designed after the design of the reward. The value function plays a crucial role in reinforcement learning algorithms, as it guides the MDP model in selecting the most appropriate action. It is a function that is determined by both states and actions and is used to assess the expected long-term gain from selecting different actions in a given state. In accordance with the Bellman equation, the value function can be expressed as:

$$q(s,a) = E\left(r_{t+1} + \gamma \max_{a'} q\left(s_{t+1}, a'\right) | s_{t+1} = s, a_t = a\right) \tag{19}$$

In this problem, with the increasing number of tasks, the state space grows exponentially, which can be interpreted as the state space tends to be continuous, so it is difficult to reflect the relationship between the state, action, and long-term value by taking the matrix data structure of Q table, so it is necessary to express the state by approximating the value function, and a parameterized continuous function $Q(s,a;\theta)$ can be used to approximate the state action-value function. In this case, the update operation of the value θ of the continuous function $Q(s,a;\theta)$ is equivalent to the update operation of the Q function.

Value function approximation is a fundamental technique in reinforcement learning that can be divided into linear and nonlinear methods. Linear methods rely on linear combinations of basis functions and parameters, which limits their ability to approximate complex functions. In contrast, nonlinear methods utilizing neural networks can theoretically approximate arbitrarily complex value functions. However, directly incorporating Q networks into reinforcement learning algorithms can lead to unstable training due to the strong correlation of the training data. To address this issue, Mnih et al. [28] proposed the DQN algorithm, which defines a target network and utilizes an experience pool structure to ensure stable convergence.

In this paper, we describe the value function using a fully connected artificial neural network, as illustrated in Fig. 7. The fully connected network is the most fundamental form of such networks, consisting of an input layer, an implicit layer, and an output layer. At each time step t, the state parameter serves as input to the network, which outputs a vector representing the expected long-term value for each possible action in state s. The network is trained using a learning algorithm that eventually converges. Ideally, in this problem, the network training process converges when the output vector represents the maximum long-term value achievable by taking different actions in any state s.

4.3.7. Network model training algorithm

In this study, we employ a Markov model to determine the preferred order of task scheduling based on action selection. However, the exact order varies slightly during the generation of initial solutions for population search and neighborhood search. During action selection, a task

Table 6
The candidate systems and their parameters.

Candidate systems	Fcn type	Cost (m$)	Earliest release time (h)	Latest working time (h)
EW aircraft I (S)	S	280	20	420
EW aircraft I (D)	D	280	50	300
EW aircraft II (S)	S	320	30	900
EW aircraft II (D)	D	320	20	150
Fighter I (S)	S	220	50	200
Fighter I (I)	I	220	100	300
Fighter II (S)	S	250	80	450
Fighter II (I)	I	250	60	200
Destroyer I (S)	S	1200	100	800
Destroyer I (D)	D	1200	120	800
Destroyer I (I)	I	1200	20	820
Destroyer II (S)	S	1500	50	300
Destroyer II (D)	D	1500	30	900
Destroyer II (I)	I	1500	20	650
UAV I (S)	S	50	50	500
UAV I (I)	I	100	100	900
UAV II (S)	S	80	30	150
UAV II (I)	I	120	40	200
USV I (S)	S	55	100	500
USV I (I)	I	80	120	400
USV II (S)	S	88	30	300
USV II (I)	I	125	100	500

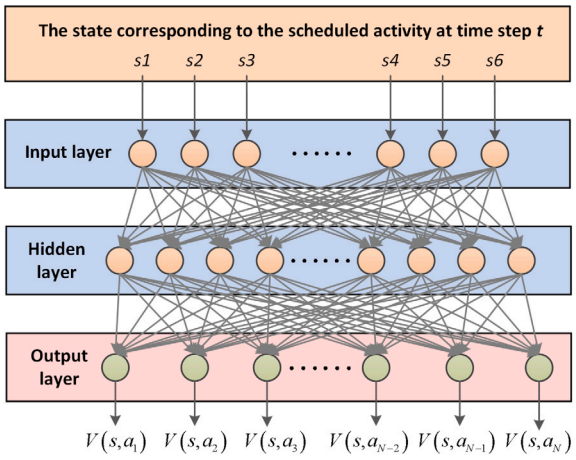


Fig. 7. Schematic representation of value function approximation based on a fully connected network.

is identified and placed after the previous task. In neighborhood search, the selected task is also rejoined to the task sequence according to a certain strategy to obtain a new neighborhood structure. Balancing exploration and mining are crucial in action selection. Exploration focuses on the global search capability of reinforcement learning methods, while mining focuses on the local search capability of reinforcement learning methods.

To select a task, a greedy approach is adopted, where the probability of selecting each selectable task in the current state is obtained, and the action with the highest probability value is selected. Alternatively, a random task can be selected from the unselected tasks. It is important to note that the set of optional tasks must be generated before selecting the action, and the selected task should remain unselected for the action selection to be valid.

The improved deep reinforcement learning algorithm is presented in Algorithm 3. The outer layer represents the number of learning rounds, and the inner layer represents traversing through the current task list and selecting an action at each time step. The state is updated by selecting an unselected task in the current task list, and the state is the list of attributes of all selected tasks at the current time step. To address the problem of poor stability in the training process, we adopt the concept of replay memory and target network. DQN stores the state transition data in the replay memory and updates the network with the sampled state transition history data. The training effect is stabilized by updating the target network weights every fixed step. The target network directly calculates the Q value of each action in the current state, and DQN calculates the reward and updates the state by selecting the action with the largest Q value. Finally, it is important to note that DQN is an off-policy reinforcement learning method, meaning that the target action value function and the action value function are not consistent. In the DQN algorithm, the target action value function and the action value function are consistent only when the target network weights $\theta' = \theta$ are updated at fixed steps.

Following data training, the DQN network can provide a decision solution for each step in the mission planning process. At the initialization stage, the DQN network selects the appropriate action selection rule based on the current state, chooses the next operation to be executed, employs the component system scheduling algorithm to arrange the suitable component systems for the operation, updates the state, and repeats the above steps until the final mission planning solution is obtained.

5. Experiment

To evaluate the efficacy of the models and algorithms presented in this paper, task scenarios were devised to test the effectiveness of the SoSs architecture selection algorithm. Both the inner layer DQN-based mission planning algorithm and the outer layer SoSs architecture selection algorithm was subjected to verification. The experimental setup is described in the first subsection of this section, followed by an analysis of the experimental results in the second subsection. These tests serve to validate the proposed models and algorithms.

5.1. Experimental settings

In this study, experiments were conducted on a desktop computer with an Intel(R) Core (TM) i9-9980HK CPU, 2.4 GHz, and 32 GB memory, running the Windows 10 operating system. The coding environment used was Python 3.9.7. The algorithms were tested using the same system configuration, and the data and code have been made available in a GitHub repository.

To assess the effectiveness of the DQN-based mission planning algorithm, we conducted experiments using ten random scenarios, each consisting of 120 operations and 21 component systems of four different types. The attributes of the component systems included their

Algorithm 3: Training Algorithm of DQN

Input: SoSs architecture A, task list T, component system list R, number of training rounds M
Output: trained network parameters θ

Initialize the action value function Q using random network parameters θ;
Initialize the target action value function $\hat{Q}$ with the same parameters $\theta' = \theta$;
Set the amount of data that can be held in the initialized relay buffer is N;
for $eposide = 1, ..., M$ **do**
  Initialize the first state and transform the state into features through the network as input $\phi(s_t)$;
  for $t = 1, ..., T$ **do**
    Select an action a_t at random with a probability ε;
    If the small probability event does not occur then the greedy strategy is used to select the action from the unselected actions;
    $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$, mark it as the selected action for the current moment;
    Execute the component system scheduling algorithm for the selected action to obtain the reward value r_t and the next state s_{t+1};
    Eigenvalue conversion for the next state using;
    Storage of status transition data in relay buffer $\phi(s_{t+1})$;
    Sampling a random set of state transfer sample data $\left(\phi(s_k), a_k, r_k, \phi(s_{k+1})\right)$ from the relay buffer;
    Execute the gradient descent algorithm and update the network parameters θ of the action value function;
    Update the target network weights every fixed step by $\theta' = \theta$;

runnable time window and type, while the attributes of the operations included their type, execution time, and waiting time. The values of each attribute were generated from a uniform distribution.

We designed four cases, each with 120 operations, labeled as "120-N", where N represents the case number. Each algorithm runs 30 times on one example, with make span used as the basis for evaluation. We statistically obtain the maximum (Best), mean (Mean), and standard deviation (Std. Dev.) of the results for evaluation benchmark metrics. We selected six experimental comparison algorithms, including the artificial bee colony algorithm (ABC) [41], constructive heuristic Algorithm 1 (HA1), heuristic Algorithm 2 (HA2), local search algorithm (LS), adaptive large domain search algorithm (ALNS) [42], and genetic algorithm (GA) [43]. The training dataset for the inner layer DQN network is set to 10, with each dataset trained for 30 epochs. The other comparison algorithm parameters are consistent with those in the literature.

To test the effectiveness of SAEA, this paper employs a synthetic air and missile defense (AMD) example which mimics the U.S. Naval Integrated Fire Control Counter Air (NIFC-CA) system. Small-unit systems and low-cost unmanned systems which known as component systems will be added to this combat SoSs.

The task set in this paper is supposed to be generated randomly. And the candidate systems and part of input parameters are listed in Table 6. The EW aircraft, fighter, destroyer, UAV and USV can provide multiple functions. Assume that each type of system has two variants with different technology advancement levels: cost and released time window. The SAEA algorithm parameters include a crossover probability of 0.9 for the outer layer genetic algorithm, a variance probability of 0.05, a population size of 10, 150 iterative search generations, a cost constraint of 8000.

5.2. Experimental results

5.2.1. Performance of DQN-based mission planning algorithm

Fig. 8 shows the utilization of the component system by the DQN at each training step. The curve represents the average utilization of component systems during the DQN's training process, with the horizontal axis indicating the training step and the vertical axis representing the average utilization of component systems. The highest average utilization of component systems occurs at the beginning of the training process when there are fewer training steps. This is because, during the initial state, operations are evenly distributed across the component systems for execution, leading to the highest level of utilization. As the training step length increases, the complexity of the operation arrangement grows, resulting in idle time for the component systems during execution and a decrease in component system utilization. However, as the training step length continues to increase, the component system utilization gradually stabilizes at a constant value.

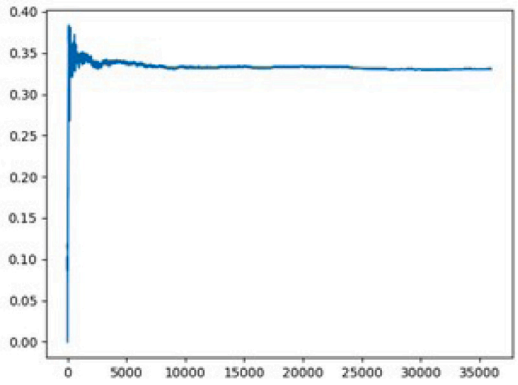


Fig. 8. Component system utilization by the DQN at each training step. X axis: number of training step, Y axis: component system utilization.

The Gantt chart depicting the mission planning process is presented in Fig. 9, which showcases the efficacy of the algorithm through the evenly distributed operations resulting in reduced execution time. The high-resolution diagrams have been replaced in the article, where the numbers on the Gantt chart represent the task serial number and the activity serial number, and the same color for the progress bar corresponding to the same task. Table 7 provides a comparative analysis of the makespan accrued by each algorithm, with our proposed DQN-based mission planning algorithm exhibiting the highest task execution gain in most cases. The superiority of the proposed algorithm over its counterparts is evident, with the mean and standard deviation indicating consistent and stable performance across multiple runs (see Fig. 11).

The box plots representing the gains achieved by each algorithm are presented in Fig. 10. Analysis of the box plots for the four case scenarios indicates that the DQN-based mission planning algorithm outperforms other algorithms in terms of gain, gain average, and convergence. These results suggest that the proposed DQN-based mission planning algorithm is capable of effectively addressing the system mission planning problem and holds significant potential for practical applications.

Fig. 11 shows the solution time of each algorithm under different task scale. It can be seen from the figure that the solution time of

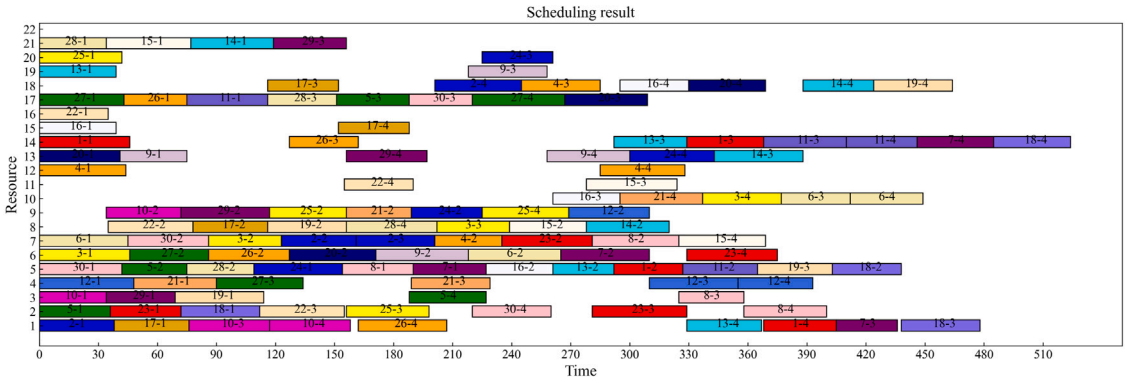


Fig. 9. Mission planning Gantt Chart.

Table 7
Comparison of makespan of the algorithm.

Instance	DQN			ABC			ALNS			GA			LS		
	min	avg	std	min	avg(WR)	std	min	avg(WR)	std	min	avg(WR)	std	min	avg(WR)	std
120-1	**428**	513.3	61.5	674	946.8-	150.3	875	967.3-	89.3	765	884.7-	88.3	921	1063.2-	129.1
120-2	**514**	620.4	56.6	744	1002.5-	153.6	869	1016.5-	89.6	809	959.2-	91.7	785	1060.5-	160.1
120-3	**476**	509.7	25.1	765	907.2-	87.6	799	918.2-	63.6	760	889.4-	104.7	777	939.7-	112.8
120-4	**487**	592.8	90.8	796	925.9-	78.5	809	966-	81.3	625	841.8-	179.4	750	922.9-	100.5

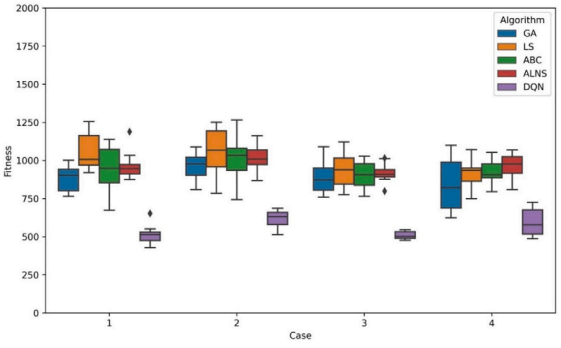


Fig. 10. Box plot of fitness for each algorithm.

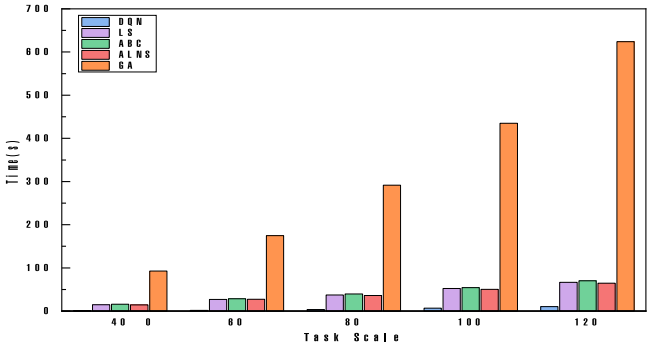


Fig. 11. The solution time of each algorithm under different task scale.

the DQN algorithm is significantly shorter than the other compared algorithms for different task sizes. This is because the number of populations and iterations for the evolution of the algorithm are specified to ensure that the number of individuals involved in the evolution of the evolutionary algorithm and the size of the iterations are the same. In the limited population and the number of iterations, the search effect of the algorithm is not very good, at the same time, when using the evolutionary algorithms, generally set up a specific coding method for a specific problem, design and improve the crossover, mutation and other operators, so as to adapt to a particular problem, in this paper, we use the comparison of algorithms belongs to the more basic algorithms, there is no algorithmic improvement of the problem proposed in this paper, so these comparison algorithms do compare with the proposed algorithm in this paper, the algorithms are not as effective as the algorithms. Algorithms are indeed much less effective compared to the algorithms proposed in this paper.

5.2.2. *Case study of the SoSs architecture evolution algorithm*

The convergence curve of the population-based System of Systems (SoSs) architecture solution process is illustrated in Fig. 12. It is evident that the population-based search strategy exhibits convergence after approximately 124 generations. The best component systems portfolio selected is: EW aircraft (S), EW aircraft (D), Fighter I (S), Destroyer I (S), Destroyer I (I), Destroyer II (S), Destroyer II (D), UAV II (S), UAV II (I), USV I (I).

Notably, the population-based strategy can discover relatively satisfactory solutions in the initial generation, owing to the large search space of the population and the reinforcement learning mission planning algorithm-based crossover and mutation operations utilized to evaluate the SoSs architecture solution. As a result, the population-based strategy can efficiently explore and identify optimal solutions within a shorter time frame.

6. Conclusion

A learning-based method for optimizing the selection of System of Systems (SoSs) architectures is proposed in this paper. Specifically, a task-oriented SoSs architecture selection optimization model is presented, which can be transformed into a two-layer nested optimization model. The outer layer optimization model focuses on modeling the SoSs architecture, while the inner layer optimization model models the mission planning problem. These two layers of optimization models are iteratively updated to generate an optimal SoSs architecture that ensures optimal mission planning. To improve the efficiency of mission planning, a DQN-based mission planning algorithm is proposed, which defines task states, uses task selection rules as actions, and learns effective rules from interaction with the environment. Experimental

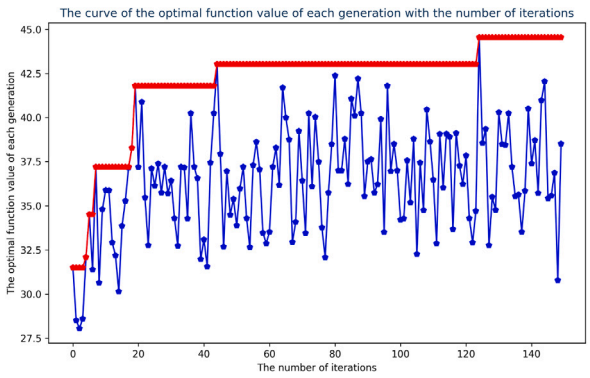


Fig. 12. Convergence curve of SoSs architecture-solving process.

results demonstrate that this algorithm not only improves the solution efficiency of mission planning, but also achieves better make span than other comparative algorithms.

The task-oriented selection optimization approach for System of Systems (SoSs) architecture has several advantages. Firstly, it boasts a clear and logical framework that highlights both the mission planning process and the optimization process of the SoSs architecture. Secondly, it is highly flexible and can easily adapt to different types of future tasks. The algorithm can be modified by the objective function of the inner layer or outer layer optimization algorithm. Although the proposed algorithm is effective, further mechanisms and operators can be designed to adapt to more complex task scenarios.

For reinforcement learning-based resource scheduling algorithms, resource integration for combat systems with a large number of tasks is significantly better than traditional optimization algorithms within a limited number of iterations. The algorithm possesses fastness as well as interpretability because many heuristics are utilized, but this method is equivalent to a combination of multiple heuristics and a combination of heuristics corresponding to the characteristics of the task learned through reinforcement learning algorithms, and thus can adapt to the scheduling of that task scenario, can perform better, and can have a higher interpretability.

Currently, it can only be used in the system design problem where the scheduling objective is used as the evaluation, and the current algorithm only considers the efficiency of the task execution, but as long as the objective function is modified, the method can optimize other objectives as well. In addition, this paper considers the case where all resources can communicate with each other, and does not discuss the case where the communication coverage is limited. However, the methodology of the solution framework can be extended to various system design problems, and the algorithm proposed in this paper can be used as a reference idea.

In future research, the authors plan to investigate SoSs architecture selection in dynamic task scenarios and analyze the mechanism of SoSs architecture construction in different scenarios. They aim to provide solutions for SoSs architecture selection and evolution problems in dynamic task scenarios.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] Mark W. Maier, Architecting principles for systems-of-systems, Syst. Eng. 1 (4) (1998) 267–284.
[2] Gautam Marwaha, Michael Kokkolaras, System-of-systems approach to air transportation design using nested optimization and direct search, Struct. Multidiscip. Optim. 51 (2014) 885–901.
[3] Joseph Tribble, Kevin MacG. Adams, Using system of systems engineering to strengthen carrier strike group C4ISR readiness evaluation, Int. J. Syst. Syst. Eng. 2 (2011) 98–111.
[4] Yong-Jun Shin, Lingjun Liu, Sang Hyun, Doo-Hwan Bae, Platooning LEGOs: An open physical exemplar for engineering self-adaptive cyber-physical systems-of-systems, in: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2021, pp. 231–237.
[5] Bingfeng Ge, Keith W. Hipel, Liping Fang, Kewei Yang, Yingwu Chen, An interactive portfolio decision analysis approach for system-of-systems architecting using the graph model for conflict resolution, IEEE Trans. Syst., Man, Cybern.: Syst. 44 (2014) 1328–1346.
[6] D.S. Jasper, Mst Hollingshead, Mosaic warfare networks can serve naval expeditionary forces, Signals 5 (2022) 76.
[7] S. Mittal, Extending DoDAF to allow integrated DEVS-based modeling and simulation, J. Def. Model. Simul. 2 (2) (2006) 95–123.
[8] Zhemei Fang, System-of-systems architecture selection: A survey of issues, methods, and opportunities, IEEE Syst. J. 16 (2022) 4768–4779.
[9] G. Myers, Effects-based operations, Armed Forces J. Int. 140 (11) (2003) 47–49.
[10] Georgiy M. Levchuk, Feili Yu, Krishna R. Pattipati, Yuri N. Levchuk, From hierarchies to heterarchies : Application of network optimization to design of organizational structures *, in: 8th International Command and Control Symposium, 2003.
[11] Georgiy M. Levchuk, Yuri N. Levchuk, Jie Luo, Krishna R. Pattipati, David L. Kleinman, Normative design of organizations. I. Mission planning, IEEE Trans. Syst. Man Cybern. A 32 (2002) 346–359.
[12] Yong Zhang, Changjiu Li, Xichao Su, Rongwei Cui, Bing Wan, A baseline-reactive scheduling method for carrier-based aircraft maintenance tasks, Complex Intell. Syst. (2022) 1–31.
[13] Georgiy M. Levchuk, Feili Yu, Yuri N. Levchuk, Krishna R. Pattipati, Networks of decision-making and communicating agents: A new methodology for design and evaluation of organizational strategies and heterarchical structures, in: Command and Control Research Program, 2005.
[14] Y. Sun, Z. Fang, Research on projection gray target model based on FANP-QFD for weapon system of systems capability evaluation, IEEE Syst. J. PP (99) (2020) 1–11.
[15] G. Lesinski, S.M. Corns, C.H. Dagli, A fuzzy genetic algorithm approach to generate and assess meta-architectures for non-line of site fires battlefield capability, in: 2016 IEEE Congress on Evolutionary Computation (CEC), 2016.
[16] M. Al-Amin, C.H. Dagli, A tool for architecting socio-technical problems: SoS explorer, in: International Symposium on Systems Engineering, 2019.
[17] Shatad Purohit, Azad M. Madni, A model-based systems architecting and integration approach using interlevel and intralevel dependency matrix, IEEE Syst. J. 16 (2022) 747–754.
[18] N. Davendralingam, D. Delaurentis, A robust optimization framework to architecting system of systems, Procedia Comput. Sci. 16 (16) (2013) 255–264.
[19] T. Wang, X. Zhou, W. Wang, Y. Zhu, T. Jing, An optimal searching algorithm for the equipment system-of-systems architecture space with uncertain capabilities, IEEE Access PP (99) (2020) 1.
[20] Imad Sanduka, Roman Obermaisser, Model-based development of Systems-of-Systems with real-time requirements, in: 2014 12th IEEE International Conference on Industrial Informatics (INDIN), 2014, pp. 188–194.
[21] I. Sanduka, R.S. Kalawsky, Y. Tian, D. Joannou, M. Masin, Incorporating architecture patterns in a SoS optimization framework, in: IEEE International Conference on Systems, 2013.
[22] Candra Meirina, Georgiy M. Levchuk, Sui Ruan, Krishna R. Pattipati, Robert L. Popp, Normative framework and computational models for simulating and assessing command and control processes, Simul. Model. Pract. Theory 14 (2006) 454–479.
[23] S. Vanfossan, C.H. Dal, B. Kwasa, A system-of-systems meta-architecting approach for seru production system design, in: 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE), 2020.
[24] Alex T. Price, Nels Knutson, Cihan H. Dagli, Using system-of-systems optimization for healthcare: A use case in radiation oncology, in: 2022 IEEE International Systems Conference (SysCon), 2022, pp. 1–5.
[25] M.M. Karim, C.H. Dagli, SoS meta-architecture selection for infrastructure inspection system using aerial drones, in: 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE), 2020.
[26] Yongliang Yuan, Qianlong Shen, Shuo Wang, Jianji Ren, Donghao Yang, Qingkang Yang, Junkai Fan, Xiaokai Mu, Coronavirus mask protection algorithm: A new bio-inspired optimization algorithm and its applications, J. Bionic Eng. (2023) 1–19.

[27] Yongliang Yuan, Qingkang Yang, Jianji Ren, Junkai Fan, Qianlong Shen, Xiaobang Wang, Yong Zhao, Learning-imitation strategy-assisted alpine skiing optimization for the boom of offshore drilling platform, Ocean Eng. 278 (2023) 114317.

[28] Yongliang Yuan, Jianji Ren, Shuo Wang, Zhenxi Wang, Xiaokai Mu, Wu Zhao, Alpine skiing optimization: A new bio-inspired optimization algorithm, Adv. Eng. Softw. (2022).

[29] Yongliang Yuan, Qianlong Shen, Wenhui Xi, Shuo Wang, Jianji Ren, Jiangong Yu, Qingkang Yang, Multidisciplinary design optimization of dynamic positioning system for semi-submersible platform, Ocean Eng. 285 (2023) 115426.

[30] Yongliang Yuan, Xiaokai Mu, Xiangyu Shao, Jianji Ren, Yong Zhao, Zhenxi Wang, Optimization of an auto drum fashioned brake using the elite opposition-based learning and chaotic k-best gravitational search strategy based grey wolf optimizer algorithm, Appl. Soft Comput. 123 (2022) 108947.

[31] Lamia Belfares, Walid Klibi, Nassirou Lo, Adel Guitouni, Multi-objectives Tabu Search based algorithm for progressive resource allocation, European J. Oper. Res. 177 (2007) 1779–1799.

[32] F. Yu, F. Tu, K.R. Pattipati, Integration of a holonic organizational control architecture and multiobjective evolutionary algorithm for flexible distributed scheduling, IEEE Trans. Syst., Man, Cybern. A 38 (5) (2008) 1001–1017.

[33] Yejia Zhao, Yanhong Wang, Yuanyuan Tan, Jun Zhang, Hong-Xia Yu, Dynamic jobshop scheduling algorithm based on deep Q network, IEEE Access 9 (2021) 122995–123011.

[34] Shu Luo, Linxuan Zhang, Yushun Fan, Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning, Comput. Ind. Eng. 159 (2021) 107489.

[35] Pierre Tassel, M. Gebser, Konstantin Schekotihin, A reinforcement learning environment for job-shop scheduling, 2021, ArXiv abs/2104.03760.

[36] Yuxin Li, Wenbin Gu, Minghai Yuan, Yaming Tang, Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep Q network, Robot. Comput. Integr. Manuf. 74 (2022) 102283.

[37] Yu jian Zeng, Zijun Liao, Yingying Dai, Rong Wang, Xiu Li, Bo Yuan, Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism, 2022, ArXiv abs/2201.00548.

[38] Yu Du, Jun-qing Li, Xiao-long Chen, Pei-yong Duan, Quan-ke Pan, Knowledge-based reinforcement learning and estimation of distribution algorithm for flexible job shop scheduling problem, IEEE Trans. Emerg. Top. Comput. Intell. (2022) 1–15.

[39] Richard S. Sutton, Andrew G. Barto, Reinforcement learning: An introduction, IEEE Trans. Neural Netw. 16 (2005) 285–286.

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, Demis Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.

[41] Bahriye Akay, Derviş Karaboğa, A modified Artificial Bee Colony algorithm for real-parameter optimization, Inform. Sci. 192 (2012) 120–142.

[42] Yingguo Chen, Yanjie Song, Yonghao Du, Mengyuan Wang, Ran Zong, Cheng Gong, A knowledge-based scheduling method for multi-satellite range system, in: Knowledge Science, Engineering and Management, 2020.

[43] Qing Wang, Haiwei Luo, Jian Xiong, Yanjie Song, Zhongshan Zhang, Evolutionary algorithm for aerospace shell product digital production line scheduling problem, Symmetry 11 (2019) 849.