applied sciences

MDPI

Article

A Two-Stage Framework for Static Task–Channel Allocation and Low-Cost Dynamic Reconfiguration Under Temporal-Frequency Constraints

Shengtao Wang [1], Han Du [2,3] and Jiafeng Zhang [1,*]

[1] School of Mechanical and Electrical Engineering, Xidian University, Xi'an 710071, China; 24041212762@stu.xidian.edu.cn
[2] Xi'an Institute of Electromechanical Information Technology, Xi'an 710065, China; han.du.work@foxmail.com
[3] National Key Laboratory of Proximity Detection and Control, Xi'an 710065, China
* Correspondence: zhangjiafeng@xidian.edu.cn

Abstract

Efficient task–channel allocation in satellite communication networks becomes particularly challenging when tasks are subject to both time and frequency constraints, and when resource failures or environmental changes invalidate an initially feasible allocation. Existing studies often treat static allocation and dynamic adaptation separately, lacking a unified framework that ensures both a low resource fragmentation rate and low reconfiguration cost. This paper proposes a two-stage approach that integrates static task–channel allocation with dynamic reconfiguration. In the static stage, a greedy algorithm is developed to assign tasks to channels under time-window, bandwidth, and conflict-free constraints, aiming to achieve as low a resource fragmentation rate as possible within the heuristic search. When channel failures occur, a heuristic search-based reconfiguration algorithm is proposed to generate a sequence of reconfiguration events that transitions the initial static allocation strategy step by step to a feasible target static allocation strategy, while maintaining constraint satisfaction and an acceptable resource fragmentation rate throughout the process. Comparative experiments on both small-scale and large-scale datasets demonstrate that the unified framework effectively balances allocation quality, low-cost and compact dynamic reconfiguration, and adaptability in dynamic network environments.

Keywords: resource allocation; dynamic reconfiguration; heuristic algorithm; satellite communication

Check for updates

Academic Editor: Hyokyung Bahn

Received: 11 May 2026
Revised: 22 June 2026
Accepted: 23 June 2026
Published: 25 June 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) license.

1. Introduction

Task–channel allocation aims to assign limited channel resources to multiple tasks in a way that satisfies task requirements while improving overall system efficiency and resource utilization. In satellite communication systems, channel resource allocation has received sustained attention because of fluctuating service demand, limited onboard resources, and strict operational constraints. Existing studies have addressed beam-hopping scheduling, satellite bandwidth management, and dynamic resource management by means of multi-objective evolutionary optimization, gradient-based improvement, and metaheuristic constraint-handling strategies [1–3]. In hybrid GEO-LEO satellite networks and short-message satellite systems, resource allocation has also been combined with task offloading, load balancing, and long-term utility optimization [4–6]. These studies demonstrate that

resource allocation in satellite-related systems is strongly affected by dynamic demand, resource coupling, and operational constraints.

Many task–channel allocation problems with temporal and frequency constraints can be naturally cast as two-dimensional bin packing problems (2D-BPP) [7–9]. Specifically, the time dimension and the frequency dimension jointly define a rectangular resource region on each channel, and each task, characterized by its required time interval and frequency bandwidth, corresponds to a rectangle that must be placed without overlap within the channel's available region. This abstraction is not merely a mathematical convenience; it brings the problem into a well-studied domain with rich theoretical results and practical heuristics. 2D-BPP and their variants have been widely applied in logistics, manufacturing, memory management, and scheduling, demonstrating both practical versatility and significant theoretical richness. By recognizing the underlying 2D-BPP, we can leverage existing packing principles—such as envelope-based placement, minimal fragmentation strategies, and shelf-based ordering—to design efficient allocation algorithms while retaining the ability to incorporate domain-specific constraints.

In wireless communication networks, especially OFDMA, femtocell, and HetNet-based systems, resource allocation has been extensively investigated under cross-tier interference, heterogeneous services, imperfect spectrum sensing, channel uncertainty, and quality-of-service requirements [10–13]. These studies have shown that convex approximation, decomposition, distributed optimization, game-theoretic modeling, and learning-assisted methods can effectively improve throughput, capacity, or network profit in specific scenarios [11,14,15]. However, most of these methods are designed for power allocation, subchannel assignment, caching, offloading, or user association, and their optimization variables and objective settings differ from task–channel allocation problems with coupled temporal-frequency constraints. Moreover, although learning-based methods have shown strong adaptability in dynamic environments [16–20], they usually emphasize long-term policy learning and often do not explicitly characterize reconfiguration feasibility, reconfiguration sequence, or transition cost under strict structural constraints.

For general constrained allocation problems, exact optimization is often difficult because of combinatorial complexity, compatibility requirements, and multi-resource coupling. Accordingly, many researchers have turned to heuristic, metaheuristic, and evolutionary methods. Representative studies include hybrid greedy–metaheuristic methods for combinatorial allocation [21], pruning-enhanced backtracking for allocation with compatibility and exclusivity constraints [22], and genetic or swarm-based methods for NP-hard multi-objective allocation problems [23,24]. In parallel, distributed and dynamic optimization methods have been developed for resource allocation under dynamic constraints, event-triggered communication, and local convex-set constraints [25–27]. These efforts indicate that the structure of constraints largely determines the choice of solution method.

In practical systems, resource failures, resource unavailability, or environmental changes may cause an originally feasible allocation scheme to become invalid. Under such circumstances, it is not sufficient to only compute a feasible target allocation; it is also necessary to determine how to move from the initial allocation state to the target state with low reconfiguration cost and acceptable execution efficiency. Nevertheless, existing studies mostly focus on static allocation optimization or dynamic adaptation separately, while relatively limited attention has been paid to the coordinated treatment of static task–channel allocation and subsequent dynamic reconfiguration, especially when both temporal and frequency constraints must be satisfied simultaneously.

To address this issue, this paper proposes a two-stage approach that integrates static task–channel allocation with dynamic reconfiguration. First, a static allocation model is established to obtain feasible task–channel allocation strategies under resource constraints.

The effectiveness of this static allocation is demonstrated through both small-scale and large-scale experiments, showing its capability to generate feasible and efficient assignments across different problem sizes. On this basis, a dynamic reconfiguration model is further constructed to generate a reconfiguration scheme when the original allocation becomes infeasible due to resource failure. The performance of the dynamic reconfiguration is evaluated on both small- and large-scale scenarios. Experimental results show that the proposed reconfiguration algorithm can generate feasible, compact, and low-cost reconfiguration schemes.

The remainder of this paper is organized as follows. Section 2 formalizes the static task–channel allocation problem and describes the greedy allocation algorithm with its complexity analysis. Section 3 defines the dynamic reconfiguration problem and proposes a heuristic search-based reconfiguration algorithm that ensures feasibility and minimizes transition cost. Section 4 provides experimental results on both small-scale and large-scale scenarios, comparing the proposed methods with several baseline approaches. Finally, Section 5 concludes the paper and discusses future research directions.

2. Static Task–Channel Allocation Problem

The goal of static task–channel allocation in a satellite communication system is to assign as many tasks as possible to suitable channels in a conflict-free manner, while achieving a low resource fragmentation rate. This section provides a formal definition of the considered static task–channel allocation problem.

2.1. Problem Description

Consider a task–channel allocation system involving n to-be-assigned tasks and m channels, where the set of to-be-assigned tasks is denoted by T and the set of channels is denoted by C.

A task $\tau_i \in T$ ($i \in \{1, 2, \ldots, n\}$) is defined by

$$\tau_i = (t_i^s, t_i^e, \Delta f_i) \tag{1}$$

where t_i^s and t_i^e denote the start and end times of task τ_i, and Δf_i represents its bandwidth requirement. The frequency allocation interval (f_i^s, f_i^e) for each to-be-assigned task is the decision variable to be determined, subject to the constraint

$$f_i^e - f_i^s = \Delta f_i \tag{2}$$

The resource demand of task τ_i, to be denoted by st_i, is calculated by the product of its time duration and bandwidth requirement:

$$st_i = (t_i^e - t_i^s) \cdot \Delta f_i \tag{3}$$

Accordingly, the total resource demand of all to-be-assigned tasks is

$$S_T = \sum_{i=1}^{n} st_i \tag{4}$$

A channel $c_j \in C$ ($j \in \{1, 2, \ldots, m\}$) is defined as

$$c_j = (b_j^s, b_j^e, d_j^s, d_j^e) \tag{5}$$

where b_j^s (b_j^e) denotes the starting (ending) frequency of channel c_j and d_j^s (d_j^e) denotes the

starting (ending) time of channel c_j. The capacity of channel c_j, to be denoted by sc_j, is given by

$$sc_j = (b_j^e - b_j^s) \cdot (d_j^e - d_j^s) \tag{6}$$

As a consequence, the amount capacity of all channels is

$$S_C = \sum_{j=1}^{m} sc_j \tag{7}$$

To represent a static task–channel allocation strategy precisely, a task–channel allocation matrix $X \in \mathbb{R}^{n \times m}$ is introduced, where

$$X(i,j) = \begin{cases} f_i^s, & \text{if } \tau_i \text{ is assigned to } c_j \text{ with starting frequency } f_i^s \\ -1, & \text{otherwise} \end{cases} \tag{8}$$

Let O_j denote the set of initial occupied tasks on channel c_j. An occupied task $o_{jk} \in O_j$ ($k = \{1, 2, \ldots, |O_j|\}$) on channel c_j is defined by

$$o_{jk} = (t_{jk}^s, t_{jk}^e, f_{jk}^s, f_{jk}^e) \tag{9}$$

where o_{jk} denotes the k-th initial occupied task on channel c_j, characterized by the time interval (t_{jk}^s, t_{jk}^e) and the frequency range (f_{jk}^s, f_{jk}^e).

Taking channel c_j as an example, as illustrated in Figure 1, there are q initially occupied tasks and g newly to-be-assigned tasks on this channel. Let $\tau_k = (t_k^s, t_k^e, f_k^s, f_k^e)$ ($k = 1, 2, \ldots, q+g$) denote a task assigned to this channel. The envelope of the tasks on channel c_j, defined as the minimal bounding region that fully contains all $q+g$ tasks, is determined by Equation (10), where α_j, β_j, γ_j, δ_j are computed as described in Equation (11).

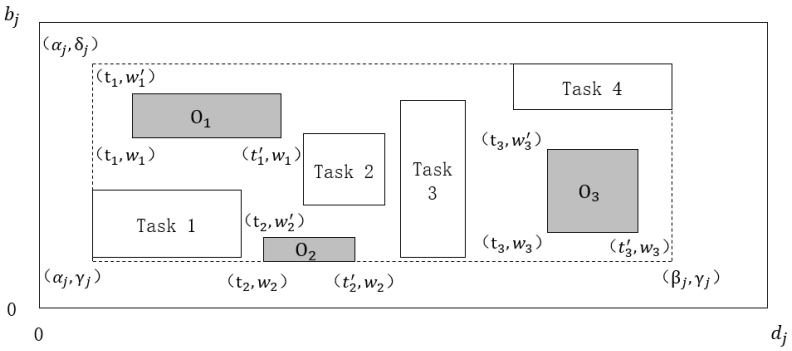


Figure 1. Envelope of tasks on channel c_j.

$$Env(c_j) = (\beta_j - \alpha_j) \cdot (\delta_j - \gamma_j) \tag{10}$$

$$\begin{cases} \alpha_j = \min_{k \in [1, q+g]} t_k^s \\ \beta_j = \max_{k \in [1, q+g]} t_k^e \\ \gamma_j = \min_{k \in [1, q+g]} f_k^s \\ \delta_j = \max_{k \in [1, q+g]} f_k^e \end{cases} \tag{11}$$

The envelope of tasks of a channel reflects the resource block used in a channel. The resource fragmentation rate (RFR) of a task–channel allocation strategy X is defined by

$$\Phi(X) = \frac{\sum_{j=1}^{m} Env(c_j)}{S_C} \tag{12}$$

Two tasks $\tau_i = (t_i^s, t_i^e, f_i^s, f_i^e)$ and $\tau_j = (t_j^s, t_j^e, f_j^s, f_j^e)$ on a shared channel are conflict-free, to be denoted by $\tau_i || \tau_j$, if the following condition holds:

$$[\min\{t_i^e, t_j^e\} \leq \max\{t_i^s, t_j^s\}] \vee [\min\{f_i^e, f_j^e\} \leq \max\{f_i^s, f_j^s\}] \tag{13}$$

the first condition indicates that the two tasks are conflict-free in the time domain, while the second condition indicates that they are conflict-free in the frequency domain. Therefore, a conflict occurs only when the two tasks overlap in both the time and frequency domains simultaneously.

Assume that task τ_i will be assigned to channel c_j in a final obtained task–channel allocation strategy X. Let A be a flag matrix of a task–channel allocation strategy X, which is defined by Equation (14).

$$A(i,j) = \begin{cases} 1, & \text{if } X(i,j) \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

The objective and constraints of the studied static task–channel allocation problem is given by Equation (15), where the constraints are described as follows:

$$\begin{aligned} \min \quad & \Phi(X) \\ \text{s.t.} \quad & \begin{cases} A(i,j) \neq 0 : b_j^s \leq f_i^s < f_i^e \leq b_j^e \ \& \ d_j^s \leq t_i^s < t_i^e \leq d_j^e & \text{(a)} \\ \Sigma_{j=1}^{m} A(i,j) \leq 1 & \text{(b)} \\ A(i,j) \neq 0 \ \& \ A(i',j) \neq 0 : \tau_i || \tau_{i'} & \text{(c)} \\ A(i,j) \neq 0, \forall o_k \in O_j : \tau_i || o_k & \text{(d)} \\ A(i,j) \neq 0 : X(i,j) \geq 0 & \text{(e)} \end{cases} \end{aligned} \tag{15}$$

- Constraint (a) indicates that the frequency range and time window of τ_i should confine within the bandwidth and the time window of c_j.
- Constraint (b) guarantees that each task is assigned to only one channel.
- Constraint (c) ensures that any two to-be-assigned tasks cannot conflict with each other if they are assigned to a same channel.
- Constraint (d) ensures that there is no conflict between τ_i and any initial occupied task in O_j if τ_i will be assigned to c_j.
- Constraint (e) indicates that the start frequency of task τ_i should be a non-negative real number.

2.2. Static Task–Channel Allocation Strategy

To address the static task–channel allocation problem, a sorting-based greedy algorithm (Algorithm 1) is employed to derive a feasible allocation strategy that satisfies all constraints in Equation (15) and keeps the RFR at a reasonably low level.

This static task–channel allocation algorithm aims to assign a set of tasks T to a set of channels C, considering initial occupied tasks $O_1, O_2, \ldots, O_{|C|}$, to produce a task–channel allocation matrix X and the corresponding RFR $\Phi(X)$. It begins by sorting tasks in ascending order of their time duration requirements. For each task τ_i in this sorted order, the algorithm iterates through all channels to identify the best candidate channel. If channel c_j satisfies constraint Equation (15) (a), it tentatively determines a feasible starting frequency for τ_i on c_j. When c_j is idle, the starting frequency is tentatively set as $f_i^s = b_j^s$ according to the minimal frequency principle. Otherwise, the starting frequency f_i^s is obtained by a sorted-interval scan such that Equation (15) (c) and Equation (15) (d) are satisfied. Specifically, only the tasks that overlap with τ_i in the time domain are considered. Their occupied frequency intervals are scanned in ascending order, starting from the lower boundary b_j^s,

and τ_i is tentatively placed at the lowest feasible starting frequency. After a feasible starting frequency is found, the algorithm computes the envelope of channel c_j by temporarily adding τ_i to its occupied task set. The channel yielding the smallest envelope after this tentative placement is recorded as the best channel for τ_i, and the corresponding starting frequency is also recorded. If such a channel exists, the task is finally allocated to the recorded best channel with the recorded starting frequency, the channel's occupied set is updated, and the corresponding minimum envelope value is recorded as min_envelope; otherwise, the task is marked as unassigned. This process continues until all tasks are processed.

Algorithm 1 Static task–channel allocation algorithm

Input: Set of to-be-assigned tasks T, set of channels C, and sets of initial occupied tasks of each channel $O_1, O_2, \ldots, O_{|C|}$.
Output: Task–channel allocation matrix X, RFR $\Phi(X)$.
Sort tasks in T by time duration requirements in ascending order;
Initialize the allocation matrix X;
for all task τ_i in the sorted set T **do**
  Initialize min_envelope = $+\infty$ and best_channel_τ_i = null;
  for all channel c_j in C; **do**
    if Equation (15) (a) holds **then**
      if $O_j = \emptyset$ **then**
        Tentatively set the starting frequency of τ_i on c_j as $f_i^s = b_j^s$;
      else
        Find the lowest feasible starting frequency by the sorted-interval scan and set it as the starting frequency f_i^s of task τ_i, such that Equation (15) (c,d) hold;
      end if
      if a feasible starting frequency f_i^s of task τ_i is found **then**
        Compute the envelope of c_j after temporarily adding τ_i to O_j by using Equation (10);
        if $Env(c_j) <$ min_envelope **then**
          min_envelope = $Env(c_j)$;
          best_channel_τ_i = c_j;
          Record the current f_i^s as the starting frequency of τ_i on best_channel_τ_i;
        end if
      end if
    end if
  end for
  if best_channel_$\tau_i \neq$ null **then**
    Assign τ_i to best_channel_τ_i with the recorded starting frequency;
    Update O_j by adding τ_i;
  else
    Mark task τ_i as an unassigned task;
  end if
end for

The computational complexity of Algorithm 1 can be analyzed as follows. Let $m = |T|$ denote the number of tasks and $n = |C|$ denote the number of channels. The initial sorting of tasks requires $O(m \cdot \log m)$ time. Then, for each task, the algorithm scans all n channels to search for the best feasible assignment. For each channel, feasibility checking involves verifying the frequency and time-window constraints in Equation (15) (a), as well as the compatibility conditions in Equation (15) (c) and Equation (15) (d) with respect to the tasks already occupying that channel. Let q denote the maximum number of occupied intervals associated with a channel during the allocation process. For each candidate channel, the placement feasibility of τ_i is checked by scanning the relevant occupied intervals. If these intervals are already maintained in sorted order, the per-channel placement check requires $O(q)$ time. Otherwise, an additional sorting step is required, and the per-channel cost becomes $O(q \cdot \log q)$. Therefore, in the worst case without sorted-interval maintenance,

the overall complexity of the channel-selection phase is $O(m \cdot n \cdot q \cdot \log q)$. Combining the sorting phase and the allocation phase, the total worst-case time complexity of the algorithm is $O(m \cdot \log m + m \cdot n \cdot q \cdot \log q)$. In the worst case, since q can be as large as m, the complexity can be further written as $O(m \cdot \log m + n \cdot m^2 \cdot \log m)$. Hence, the proposed greedy algorithm runs in polynomial time and is suitable for static task–channel allocation with moderate problem sizes. If the envelope value of a channel can be incrementally maintained, the practical runtime can be lower than the above worst-case bound.

3. Dynamic Reconfiguration of Allocation Strategy

To ensure robust task execution despite potential partial channel failures, this section investigates the corresponding resource allocation challenge. Extending the static allocation scheme, we focus on this dynamic scenario, which we refer to as the dynamic reconfiguration problem of task–channel allocation strategies.

3.1. Problem Description

In the current work, a reconfiguration event r changes a static task–channel allocation strategy X to another strategy X', where the two strategies differ in the allocation of exactly one task. This implies that a reconfiguration event is defined as a reallocation event of a task. Let τ_i be the task operated by the reconfiguration event r. The following cases are permitted:

- τ_i remains on the same channel c_j but its start frequency is changed, i.e., $(X(i,j) \neq X'(i,j) \neq -1)$;
- τ_i is moved from channel c_j to a different channel $c_{j'}$, i.e., $(X(i,j) \neq -1,\ X'(i,j) = -1$ and $X(i,j') = -1,\ X'(i,j') \neq -1)$;
- τ_i is removed from its allocated channel c_j, i.e., $(X(i,j) \neq -1,\ X'(i,j) = -1)$;
- τ_i is newly allocated to a channel c_j, i.e., $(X(i,j) = -1,\ X'(i,j) \neq -1)$.

In all cases, all other entries remain unchanged.

Given the initial static task–channel allocation strategy X_0 before channel failures and the objective fault-tolerant static task–channel allocation strategy X_d, a reconfiguration scheme R is defined as a finite sequence of reconfiguration events:

$$R = r_1, r_2, \ldots, r_d \tag{16}$$

The corresponding sequence of intermediate strategies of R is denoted by $R_X = X_0, X_1, X_2 \ldots, X_d$ and we have $X_0 \xrightarrow{r_1} X_1 \xrightarrow{r_2} X_2 \ldots \xrightarrow{r_d} X_d$.

Let τ_j ($j \in \{1, 2, \ldots, n\}$) be the task that processed by the reconfiguration event r_i ($i \in \{1, 2, \ldots, d\}$). The implementation cost of r_i is assumed to be proportional to the resource demand of the processed task τ_j, which is computed by Equation (3), i.e.,

$$Cost(r_i) = \alpha \times st_j \tag{17}$$

Here, weighting coefficient α is a positive real number to scale the normalized resource-dependent reconfiguration cost. Generally, α is related to factors such as task urgency, channel switching overhead, and resource preemption cost. Accordingly, the cost of a reconfiguration scheme $R = r_1, r_2, \ldots, r_d$ is defined by Equation (18).

$$Cost(R) = \Sigma_{i=1}^{d} Cost(r_i) \tag{18}$$

The dynamic reconfiguration problem for task–channel allocation strategies is formulated in Equation (19), with the objective of finding a reconfiguration scheme that minimizes the total cost. The first constraint describes the relationship between consecutive strategies.

The second constraint requires that no task conflict exists in any intermediate strategy. Finally, the last constraint specifies that the RFR of every intermediate strategy must not exceed β, where $0 < \beta < 1$.

$$\begin{aligned} \min \quad & Cost(R) \\ \text{s.t.} \quad & \begin{cases} X_i \xrightarrow{r_i} X_{i+1} & \forall i = \{0, 1, \ldots, d-1\} \\ X_i \text{ satisfies all constraints in Equation (15)} & \forall i = \{0, 1, \ldots, d\} \\ \Phi(X_i) \leq \beta & \forall i = \{0, 1, \ldots, d\} \end{cases} \end{aligned} \tag{19}$$

3.2. Dynamic Reconfiguration Algorithm

Solving the dynamic reconfiguration problem is highly complex. For each task operation, except for task deletion, all other operations require computing a suitable placement position for the task, i.e., a new starting frequency. However, the starting frequency is actually continuous. Moreover, although the set of tasks that need to be reconfigured is known, the operations performed on them still lead to differences in the intermediate allocation strategies during the reconfiguration process, especially because these intermediate strategies must satisfy all constraints in Equation (15). Furthermore, the RFR of the intermediate strategies is also subject to certain constraints. Different sequences of reconfiguration events can result in different total numbers of steps in the reconfiguration process. For example, a task can either be moved directly to its target position or first deleted and then re-added, which means that the operation on this task may take either one step or two steps. Since each additional step increases the reconfiguration cost, we expect the total reconfiguration cost to be as small as possible.

The current work adopts an A^*-inspired heuristic best-first search algorithm to find a feasible low-cost reconfiguration scheme R from the initial task–channel allocation matrix X_0 to the target task–channel allocation matrix X_d, where X_0 and X_d are obtained by Algorithm 1. In the proposed method, each node in the search space represents a possible intermediate task–channel allocation strategy, and each edge corresponds to a reconfiguration event. The intermediate strategy X_k is evaluated by using the evaluation function:

$$f(X_k) = g(X_k) + h(X_k) \tag{20}$$

where $g(X_k)$, as defined in Equation (21), is the actual cumulative reconfiguration cost from X_0 to X_k and $h(X_k)$, as defined in Equation (22), is a heuristic guidance term that measures the envelope-area difference between the intermediate strategy X_k and the target strategy X_d. Specifically, $Env(c_j)^k$ ($i \in \{1, 2, \ldots, d\}$ and $j \in \{1, 2, \ldots, m\}$) denotes the envelope of channel c_j under the allocation strategy X_k. Since $h(X_k)$ is designed to guide the search according to the envelope difference rather than to provide a strict lower bound on the remaining reconfiguration cost, the proposed search does not claim the global optimality guarantee of classical A^*. Instead, it is used as a heuristic best-first search method to efficiently obtain a feasible reconfiguration scheme with low implementation cost.

$$g(X_k) = \sum_{i=1}^{k} Cost(r_i) \tag{21}$$

$$h(X_k) = \sum_{j=1}^{m} |Env(c_j)^k - Env(c_j)^d| \tag{22}$$

A detailed description of the dynamic reconfiguration algorithm is provided in Algorithm 2. It initializes Open and Close tables, adds X_0 to the Open table, and iterates while the Open table is not empty: at each step, the strategy X_k with the minimum evalua-

tion is selected from the Open table. If X_k equals X_d, the algorithm backtracks from X_d to X_0 to output R; otherwise, X_k is moved to the Close table, compared with X_d, and the set N_{k+1} of possible next strategies satisfying constraints in Equation (19) is computed. For each strategy X_{k+1} in N_{k+1}, if it is in the Close table, it is skipped; otherwise, the actual cumulative reconfiguration cost $g(X_{k+1})$ and heuristic cost $h(X_{k+1})$ are calculated. If it is already in the Open table but the newly obtained $g(X_{k+1})$ is smaller, its cost value and predecessor are updated.

Algorithm 2 Dynamic reconfiguration algorithm

Input: Initial resource allocation strategy X_0 and target resource allocation strategy X_d.
Output: Reconfiguration scheme R.
Initialize Open table and Close table;
Add X_0 to the Open table which stores candidate strategies to be evaluated;
while Open table is not empty **do**
Take a strategy X_k that satisfies all constraints in Equation (15) and is with the minimum evaluation from the Open table;
if X_k equals X_d **then**
Backtrack from X_d to X_0;
Return reconfiguration scheme R;
else
Add X_k to the Close table which stores evaluated strategies to avoid re-processing;
Compare X_k with X_d to obtain the set of tasks that need to be reconfigured;
Compute the set of possible next strategies of X_k satisfying constraints in Equation (19), to be denoted by N_{k+1};
end if
for all possible next strategies in N_{k+1} **do**
if X_{k+1} is in the Close table **then**
Skip;
else
Calculate $g(X_{k+1})$ and $h(X_{k+1})$;
end if
if X_{k+1} is not in the Open table **then**
Add strategy X_{k+1} to the Open table;
else if the newly obtained $g(X_{k+1})$ is smaller than the recorded one **then**
Update $g(X_{k+1})$,$f(X_{k+1})$ in the Open table;
end if
end for
end while

This algorithm has a worst-case time complexity of $O(V[m \cdot n + b(m \cdot n + K + \log V)])$, where m is the number of tasks, n is the number of channels, K is the total number of occupied intervals in all channels, V is the number of visited states, and b is the average branching factor. For each expanded state, comparing X_k with X_d requires $O(m \cdot n)$ time, since the allocation matrix contains $m \cdot n$ entries. For each candidate successor, feasibility checking, conflict detection, and cost evaluation require $O(m \cdot n + K)$ time, while maintaining the Open table implemented as a min-heap requires $O(\log V)$ time. Therefore, the complexity can be simplified as $O(V \cdot b(m \cdot n + K + \log V))$. Let s denote the number of adjusted tasks. The number of expanded states V depends on s, the branching factor, and the conflict-induced evictions during reconfiguration. In the worst case, V may grow exponentially with s, i.e., $V = O(b^s)$. Therefore, the time complexity can also be expressed as $O(b^s[m \cdot n + b(m \cdot n + K + \log V)])$, The space complexity is $O(V(m \cdot n + K))$, because the Open and Close tables store visited allocation matrices and the corresponding channel occupation states.

4. Case Study

This paper validates the proposed static task–channel allocation method and allocation scheme dynamic reconfiguration method through several sets of data. In addition, the performance of these methods is compared with related algorithms. The experiments are conducted on a Legion Y9000P laptop with Windows 11, an Intel Core Ultra 9 processor, and 32 GB of memory. The algorithms were implemented in Python 3.13.5 and developed using PyCharm 2024.1.7 Professional Edition.

In the static task–channel allocation experiments, we first conducted experiments using a small-scale dataset. The small-scale datasets are used to illustrate the allocation process under different task sizes and initial channel-occupation conditions. Each small-scale group consists of three channels, and the number of tasks varies from 10 to 25. The time range of the channels is [0, 90], and the bandwidth ranges of the three channels are [0, 36], [0, 54], and [0, 72], respectively. Although the channel bandwidth settings are the same across the small-scale groups, the number of tasks and the initially occupied tasks on each channel are different, resulting in different resource-occupation patterns.

The following algorithms: dynamic programming algorithm, Hungarian algorithm, genetic algorithm, and randomized algorithm are selected as baseline methods for comparison with Algorithm 1. For a fair comparison, all these methods are implemented under the same task and channel constraints. The dynamic programming algorithm formulates the bandwidth allocation problem as a two-dimensional bin-packing variant and derives the final allocation scheme by applying a staged allocation process to multiple candidate task sequences under the channel-capacity and non-overlapping constraints. The Hungarian algorithm formulates the channel–task allocation problem as a constrained assignment problem, constructs a cost matrix according to the feasibility and bandwidth-occupation change of each task–channel assignment, and iteratively updates the task set and channel-occupation state to obtain the final allocation scheme. The genetic algorithm encodes each allocation scheme as a chromosome consisting of task-ordering and channel-selection genes. During decoding, tasks are assigned according to the encoded order and selected channels. If a task cannot be feasibly placed on the encoded channel, it is repaired by searching other feasible channels; otherwise, it is marked as unassigned. In the experiments, the genetic algorithm uses a population size of 50 and runs for 50 generations in small-scale cases, while it uses a population size of 500 and runs for 100 generations in large-scale cases. The maximum number of generations is used as the stopping criterion. The randomized algorithm processes tasks according to their input order and places each task by sequentially checking the available channels and feasible positions under the non-overlapping constraint, ensuring a consistent allocation process; if no feasible placement is found, the task is discarded and the algorithm proceeds to the next task. All methods are evaluated using the same RFR definition and computational environment.

As shown in Table 1, the compared methods are evaluated using two metrics: RFR and computational time. RFR is reported as a percentage, and the running time is reported in seconds using the format mean $\pm$ standard deviation over 30 repeated runs. Algorithm 1 performs well on both metrics. In terms of RFR, Algorithm 1 has the lowest value among all the compared methods in most experiments. For example, it remains at a low value in the small-scale dataset, far lower than the generally high levels of other methods, which proves that the algorithm model indicates better solution quality under the considered objective. In terms of computational efficiency, the running time of this method is also significantly lower than that of these selected methods. In the small-scale static task–channel allocation experiments, Algorithm 1 even achieves the shortest runtime among all compared methods and has the fastest running speed. In addition, Algorithm 1 achieves both the lowest RFR and the shortest computational time in almost all experimental scenarios, leading

comprehensively in these two core dimensions, which fully proves the superiority and practicality of this method.

Table 1. Comparison of small-scale data for static task–channel allocation experiments.

Group	Algorithm 1		Dynamic Programming		Hungarian Algorithm		Genetic Algorithm		Randomized Algorithm	
	RFR	Running Time (s)	RFR	Running Time (s)	RFR	Running Time (s)	RFR	Running Time (s)	RFR	Running Time (s)
1	35.61%	0.00 ± 0.02	63.46%	0.72 ± 0.09	70.41%	0.00 ± 0.02	63.24%	3.25 ± 0.17	53.55%	0.72 ± 0.07
2	77.04%	0.00 ± 0.03	79.73%	0.80 ± 0.12	84.57%	0.10 ± 0.03	77.92%	3.47 ± 0.15	74.82%	0.81 ± 0.12
3	66.09%	0.00 ± 0.01	80.68%	1.16 ± 0.11	77.65%	0.02 ± 0.01	77.85%	3.73 ± 0.23	84.69%	0.88 ± 0.13
4	68.42%	0.00 ± 0.02	83.40%	1.98 ± 0.12	76.79%	0.00 ± 0.02	75.62%	3.23 ± 0.14	69.44%	0.83 ± 0.09
5	69.86%	0.00 ± 0.02	71.81%	1.04 ± 0.08	77.53%	0.00 ± 0.02	74.46%	3.43 ± 0.15	72.67%	0.82 ± 0.10
6	66.32%	0.00 ± 0.03	87.54%	0.82 ± 0.10	87.54%	0.01 ± 0.03	90.14%	3.93 ± 0.25	77.54%	0.92 ± 0.12
7	72.71%	0.00 ± 0.03	82.10%	0.70 ± 0.09	90.53%	0.01 ± 0.03	82.89%	3.93 ± 0.25	75.01%	0.92 ± 0.12
8	72.35%	0.00 ± 0.01	69.71%	0.76 ± 0.10	85.47%	0.01 ± 0.01	78.50%	3.67 ± 0.20	69.60%	0.89 ± 0.13

Following the completion of the static task–channel allocation experiments, we further evaluated the performance of dynamic reconfiguration under channel failure conditions. Table 2 reports the damage scenario of the eight small-scale datasets, while Table 3 compares Algorithm 2, the rule-based algorithm, and the randomized algorithm in terms of scheme length, reconfiguration cost, and running time on the eight small-scale datasets. The initial allocation strategy X_0 is generated by Algorithm 1 before channel failure, while the target allocation strategy X_d is obtained by re-running Algorithm 1 after a single-channel failure. For consistency, the failed channel is set to the third channel in all datasets, whose bandwidth range is [0, 72]. Since this channel has the largest bandwidth among the three channels, its failure represents a relatively severe single-channel failure scenario. Note that Algorithm 2 takes X_0 and X_d as inputs and is not restricted to a specific failed channel; if another channel fails, the corresponding X_d can be generated in the same way and the same reconfiguration procedure can be applied.

Table 2. Damage scenario of dynamic reconfiguration experiments on small-scale data.

Metric	1	2	3	4	5	6	7	8
Total number of tasks	10	15	20	10	15	20	15	24
Matrix differences	16	12	21	10	15	14	15	15
Number of adjusted tasks	8	9	10	7	9	7	10	10

Table 3. Results of dynamic reconfiguration algorithms on small-scale data.

Algorithm	Index	1	2	3	4	5	6	7	8
Algorithm 2	Scheme length	11	11	17	8	11	10	14	14
	Cost	33,150	34,200	43,710	23,800	39,530	34,460	38,320	41,160
	Running time (s)	0.01 ± 0.02	0.01 ± 0.02	0.06 ± 0.03	0.00 ± 0.02	0.56 ± 0.12	0.11 ± 0.01	0.00 ± 0.01	0.53 ± 0.13
Rule-based algorithm	Scheme length	16	18	20	14	18	14	20	20
	Cost	39,460	40,400	45,120	31,600	41,140	37,270	42,230	44,170
	Running time (s)	0.40 ± 0.11	0.45 ± 0.12	0.53 ± 0.13	0.42 ± 0.10	0.83 ± 0.21	0.51 ± 0.14	0.52 ± 0.15	0.84 ± 0.22
Randomized algorithm	Scheme length	14	15	19	12	17	13	18	19
	Cost	37,450	37,950	44,460	27,800	40,970	36,710	40,700	43,890
	Running time (s)	0.62 ± 0.17	0.67 ± 0.19	0.71 ± 0.16	0.65 ± 0.15	0.98 ± 0.20	0.71 ± 0.19	0.64 ± 0.18	0.97 ± 0.21

As shown in Table 2, the first row denotes the total number of tasks included in the experiment, the second row denotes the number of different entries between the initial allocation matrix and the target allocation matrix, and the third low denotes the number of tasks whose allocation states need to be changed. The rule-based algorithm first extracts these adjusted tasks from the initial strategy and then places them into the target channels one by one according to X_d. As shown in Table 3, Algorithm 2 achieves the shortest scheme

length in all datasets and the lowest reconfiguration cost in all datasets. It also has the lowest running time on the small-scale cases, indicating that it can efficiently generate compact and low-cost reconfiguration schemes when the problem size is limited. In contrast, the rule-based algorithm obtains the longest scheme length and the highest reconfiguration cost, because it removes the affected tasks one by one and then reallocates them one by one during the reconfiguration process, which increases both the number of reconfiguration events and the accumulated implementation cost. The randomized algorithm generally produces longer schemes and higher costs due to its random task-selection process. These results demonstrate the superiority of Algorithm 2 in generating efficient, compact, and low-cost reconfiguration schemes under small-scale dynamic reconfiguration scenarios. The running time is reported in seconds using the format mean ± standard deviation over 30 repeated runs. For the randomized algorithm, the random selection sequence is fixed within each experimental group, so repeated runs produce the same reconfiguration result and only the running time varies.

The reconfiguration process generated by Algorithm 2 is represented as a sequence of reconfiguration events, where each event denotes the allocation-state change of one task. Since moving one task may conflict with other tasks, some tasks can be temporarily removed and reallocated in later steps. To illustrate this dynamic process, the second small-scale experimental set is used as an example, the initial static allocation strategy is $X_0 = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & 0 & 19 & -1 & -1 & -1 & -1 & -1 & 13 & 0 \\ -1 & -1 & -1 & -1 & 0 & 18 & -1 & -1 & -1 & -1 & 33 & 22 & 0 & -1 & -1 \\ 49 & -1 & -1 & 40 & -1 & -1 & -1 & -1 & 17 & 0 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$, the target static allocation strategy is $X_d = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & -1 & 19 & -1 & -1 & -1 & 13 & 0 \\ -1 & -1 & -1 & -1 & 0 & 33 & -1 & 18 & -1 & -1 & -1 & 33 & 0 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$, and its reconfiguration sequence is shown in Figure 2. The obtained sequence is $r_1, r_4, r_8, r_{10}, r_9, r_{12}, r_6, r_8, r_{11}, r_{12}, r_6$. In Figure 2, each sub-matrix represents an intermediate allocation state, with rows corresponding to channels and columns corresponding to tasks. Each sub-matrix has a size of 3×15, representing the allocation relationship between 3 channels and 15 tasks. The colored blocks highlight the changes between consecutive states: blue denotes the original task placement, yellow denotes task addition, red denotes task deletion, and green denotes task movement. This example shows how Algorithm 2 adjusts tasks step by step until the target allocation state is reached.

After the small-scale experiments, large-scale experiments are conducted to further evaluate the scalability of the proposed methods. Compared with the small-scale setting, the large-scale setting uses more diverse channel bandwidth ranges, selected from [0, 36], [0, 54], [0, 72], and [0, 300]. The datasets cover four task-density settings, namely low-load, high-load, overload, and high-occupancy cases. For each setting, both uniform and Gaussian task distributions are considered. Specifically, Groups 1–2 are low-load cases, Groups 3–4 are high-load cases, Groups 5–6 are overload cases, and Groups 7–8 are high-occupancy cases, where the two groups in each pair correspond to uniform and Gaussian distributions, respectively. These datasets are used to examine algorithm performance under different resource-demand intensities, task-distribution patterns, and channel bandwidth configurations.

As shown in Table 4, Algorithm 1 maintains good performance on the large-scale datasets, where each dataset contains 500 tasks and 400 channels. The evaluation metrics are the same as those used in Table 1, including RFR and running time. Similar to the small-scale results, Algorithm 1 achieves competitive RFR values across the eight experimental groups, indicating stable allocation quality under different task-density and task-distribution settings. In terms of computational efficiency, Algorithm 1 shows a clear advantage. Its running time is only about 3–5 s in all large-scale cases, whereas the com-

pared methods require tens or even hundreds of seconds. These results demonstrate that Algorithm 1 can maintain stable allocation performance and high computational efficiency as the problem scale increases.

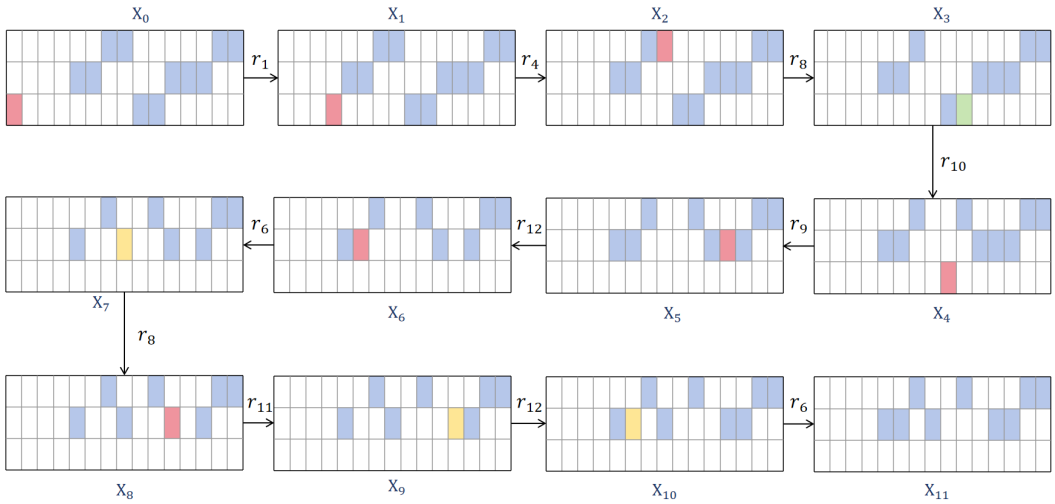


Figure 2. Reconfiguration procedure diagram. Blue, yellow, red, and green indicate the original task placement, task addition, task deletion, and task movement, respectively.

Table 4. Comparison of large-scale data for static task–channel allocation experiments.

Group	Algorithm 1		Dynamic Programming		Hungarian Algorithm		Genetic Algorithm		Randomized Algorithm	
	RFR	Running Time (s)	RFR	Running Time (s)	RFR	Running Time (s)	RFR	Running Time (s)	RFR	Running Time (s)
1	35.23%	5.43 ± 0.23	40.97%	2.33 ± 0.18	56.63%	57.51 ± 1.12	45.27%	21.60 ± 0.96	42.16%	0.84 ± 0.11
2	35.04%	5.56 ± 0.25	41.18%	2.04 ± 0.16	56.79%	57.78 ± 1.18	44.46%	29.51 ± 1.13	42.19%	0.96 ± 0.15
3	52.80%	3.24 ± 0.15	59.87%	6.58 ± 0.20	66.60%	36.32 ± 1.05	58.31%	154.22 ± 2.87	60.54%	0.97 ± 0.16
4	53.75%	3.30 ± 0.18	59.53%	6.34 ± 0.25	65.89%	46.92 ± 1.22	59.59%	169.48 ± 2.31	60.17%	0.91 ± 0.17
5	57.89%	3.14 ± 0.20	65.30%	8.39 ± 0.21	67.32%	32.30 ± 1.33	65.88%	47.60 ± 1.35	64.72%	1.00 ± 0.20
6	57.66%	3.08 ± 0.13	64.65%	8.42 ± 0.22	66.94%	32.52 ± 1.02	65.27%	47.68 ± 1.33	63.46%	0.93 ± 0.16
7	89.06%	5.37 ± 0.22	88.80%	27.82 ± 1.04	91.15%	98.32 ± 1.97	89.41%	125.28 ± 2.57	88.93%	0.95 ± 0.18
8	89.18%	5.21 ± 0.29	88.78%	52.41 ± 1.86	91.15%	100.96 ± 2.68	89.52%	232.89 ± 3.79	89.03%	0.95 ± 0.17

After evaluating Algorithm 1 on large-scale static allocation cases, we further examine the large-scale dynamic reconfiguration performance of Algorithm 2. Since Algorithm 2 is limited by the reconfiguration search space, the large-scale dynamic experiments use a reduced setting with 350 tasks and 200 channels compared with the large-scale static allocation experiments. Meanwhile, the number of adjusted tasks is controlled around 20 to ensure that the reconfiguration search can be completed. In each group, the same damaged channel ids are used for all compared algorithms to ensure consistency. Table 5 reports the damage scenario of the eight large-scale reconfiguration groups, including the number of damaged channels, matrix differences, and the number of adjusted tasks. Compared with the small-scale results, Table 5 additionally reports the number of damaged channels to describe the failure severity. Table 6 compares Algorithm 2, the rule-based algorithm, and the randomized algorithm in terms of scheme length, reconfiguration cost, and running time. As shown in Table 6, Algorithm 2 achieves the shortest scheme length and the lowest reconfiguration cost in all large-scale groups. These results show that, when the number of adjusted tasks is controlled, Algorithm 2 can still generate compact reconfiguration schemes with low reconfiguration cost under larger problem settings. Meanwhile, the running time increases noticeably compared with the small-scale cases, since the enlarged allocation matrix and reconfiguration search space require more intermediate states to be generated and evaluated. This indicates that the proposed method provides a favorable balance

between reconfiguration quality and computational effort, with improved compactness and lower reconfiguration cost obtained at the cost of longer running time.

Table 5. Damage scenario of dynamic reconfiguration experiments on large-scale data.

Metric	1	2	3	4	5	6	7	8
Number of damaged channels	8	10	7	8	9	6	7	9
Matrix differences	36	50	30	30	30	28	30	30
Number of adjusted tasks	20	23	15	14	15	12	14	15

Table 6. Results of dynamic reconfiguration algorithms on large-scale data.

Algorithm	Index	1	2	3	4	5	6	7	8
Algorithm 2	Scheme length	22	26	20	20	21	18	19	22
	Cost	68,167.56	60,727.41	63,479.22	62,466.28	76,339.53	64,813.26	62,495.81	63,878.87
	Running time (s)	309.65 ± 2.21	248.61 ± 1.58	139.22 ± 1.87	197.41 ± 2.02	274.48 ± 1.94	236.38 ± 2.13	79.06 ± 1.01	146.86 ± 1.43
Rule-based algorithm	Scheme length	40	46	30	28	30	24	28	30
	Cost	107,347.56	89,533.49	73,582.98	70,358.49	94,872.89	72,049.46	80,650.61	71,563.76
	Running time (s)	0.65 ± 0.17	0.66 ± 0.16	0.78 ± 0.18	0.80 ± 0.19	0.70 ± 0.15	0.73 ± 0.16	0.73 ± 0.15	0.73 ± 0.16
Randomized algorithm	Scheme length	30	30	22	23	24	22	22	25
	Cost	77,276.84	63,087.43	65,535.79	68,518.02	80,982.83	66,264.57	65,855.97	68,773.83
	Running time (s)	0.70 ± 0.17	0.72 ± 0.16	0.69 ± 0.15	0.75 ± 0.16	0.80 ± 0.18	0.71 ± 0.17	0.78 ± 0.17	0.77 ± 0.16

The parameter α for all experiments is set to 10 and the parameter β is set to 0.9. Here, α is used to weight the accumulated resource-dependent reconfiguration cost, while β defines the upper bound of RFR for intermediate allocation strategies. The same parameter settings are used for all datasets to avoid dataset-specific tuning and ensure a fair comparison. To examine the influence of α, sensitivity tests are conducted on the first small-scale dynamic reconfiguration case. As summarized in Table 7, the results show that increasing α from 1 to 10 helps Algorithm 2 obtain a shorter scheme length and a lower unscaled reconfiguration cost, while further increasing α to 15 and 20 does not improve the solution quality but significantly increases the running time. Therefore, $\alpha = 10$ is used as the default setting, providing a reasonable balance between reconfiguration quality and computational efficiency. In addition, $\beta = 0.9$ is determined according to the experimental setting to allow feasible intermediate allocation strategies while keeping the RFR within an acceptable range.

Table 7. Sensitivity analysis of α for Algorithm 2 on the first small-scale dynamic reconfiguration case.

Metric	$\alpha = 1$	$\alpha = 5$	$\alpha = 10$	$\alpha = 15$	$\alpha = 20$
Scheme Length	12	12	11	11	11
Cost/α	3865	3865	3315	3315	3315
Running time (s)	0.00 ± 0.01	0.00 ± 0.01	0.01 ± 0.01	26.09 ± 0.86	65.44 ± 1.07

5. Conclusions

This paper addressed the problem of task–channel allocation in satellite communication systems under both static and dynamic conditions, with a particular focus on tasks constrained by time windows and frequency requirements. A two-stage framework was proposed: a greedy static allocation algorithm that minimizes resource fragmentation while satisfying all conflict and feasibility constraints, followed by a heuristic search-based dynamic reconfiguration algorithm that generates low-cost strategy reconfiguration event sequences when resource failures occur. Experimental validation on small- and large-scale datasets demonstrated that the static allocation algorithm consistently outperforms baseline methods—including dynamic programming, Hungarian, genetic, and randomized algorithms—in terms of both resource fragmentation rate and computational runtime. In the

dynamic reconfiguration experiments, the proposed method generated feasible reconfiguration sequences and achieved shorter scheme lengths and lower reconfiguration costs than the compared methods in both small- and large-scale cases. For the large-scale dynamic cases, the experiments were conducted with a controlled number of adjusted tasks, which allowed the reconfiguration search to remain tractable while still reflecting larger allocation matrices and more complex channel conditions. The results show that the proposed method is particularly effective in producing compact and low-cost reconfiguration schemes, while its computational time increases as the allocation scale and search space grow.

Despite these advantages, the current method faces computational challenges in very large-scale problems, where the complexity of the reconfiguration search space may become a bottleneck. Future research will focus on integrating metaheuristic techniques such as genetic algorithms or simulated annealing, as well as learning-based methods, to further improve scalability.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/app16136354/s1.

Author Contributions: Methodology, H.D.; investigation, H.D. and S.W.; writing—original draft preparation, S.W.; writing—review and editing, J.Z.; supervision, J.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Fundamental Research Funds for the Central Universities, grant number QTZX26066. The APC was funded by the Fundamental Research Funds for the Central Universities.

Data Availability Statement: The data presented in this study are available in the article and its Supplementary Material. The algorithm descriptions and related data supporting the findings of this study are included in the main text and Supplementary Material. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Martínez Zamacola, S.; Luna Valero, F.; Martínez Rodríguez Osorio, R. Hybrid MOEA with problem-specific operators for beam-hopping based resource allocation in multi-beam LEO satellites. *Swarm Evol. Comput.* **2025**, *99*, 102174. [CrossRef]
2. Zhao, D.; Xiong, W.; Shi, J. Exploration of the application of optimisation algorithm using stochastic gradient descent method in satellite resource allocation. *Appl. Math. Nonlinear Sci.* **2024**, *9*, 1–18. [CrossRef]
3. Gao, W.; Wang, L.; Qu, L. Research on Joint Resource Allocation for Multibeam Satellite Based on Metaheuristic Algorithms. *Entropy* **2022**, *24*, 1536. [CrossRef] [PubMed]
4. Xia, K.; Feng, J.; Yan, C.; Duan, C. BeiDou Short-Message Satellite Resource Allocation Algorithm Based on Deep Reinforcement Learning. *Entropy* **2021**, *23*, 932. [CrossRef] [PubMed]
5. Liao, X.; Hu, X.; Liu, Z.; Ma, S.; Xu, L.; Wang, W.; Ghannouchi, F.M.; Li, X. Distributed Intelligence: A Verification for Multi-Agent DRL-Based Multibeam Satellite Resource Allocation. *IEEE Commun. Lett.* **2020**, *24*, 2785–2789. [CrossRef]
6. Leng, T.; Duan, P.; Hu, D.; Cui, G.; Wang, W. Cooperative user association and resource allocation for task offloading in hybrid GEO-LEO satellite networks. *Int. J. Satell. Commun. Netw.* **2021**, *40*, 230–243. [CrossRef]
7. Gao, M.; Chen, Y.; Li, J.; Wahab, M.I.M. Hybrid branch-and-price-and-cut algorithm for the two-dimensional vector packing problem with time windows. *Comput. Oper. Res.* **2023**, *157*, 106267. [CrossRef]
8. Rodrigues, C.D.; Cherri, A.C.; de Araujo, S.A. Strip based compact formulation for two-dimensional guillotine cutting problems. *Comput. Oper. Res.* **2023**, *149*, 106044. [CrossRef]
9. Souza, F.; Grimes, D. Combining local search and genetic algorithm for two-dimensional guillotine bin packing problems with partial sequence constraint. *CEUR Workshop Proc.* **2020**, *2771*, 97–108.
10. Zhang, Y.; Wang, S. Resource Allocation for Cognitive Radio-Enabled Femtocell Networks with Imperfect Spectrum Sensing and Channel Uncertainty. *IEEE Trans. Veh. Technol.* **2016**, *65*, 7719–7728. [CrossRef]

11. Cao, X.; Yan, S.; Zhang, H. Resource allocation for network profit maximization in NOMA-based F-RANs: A game-theoretic approach. *Front. Inf. Technol. Electron. Eng.* **2022**, *23*, 1546–1561. [CrossRef]
12. Zhang, H.; Jiang, C.; Beaulieu, N.C.; Chu, X.; Wen, X.; Tao, M. Resource Allocation in Spectrum-Sharing OFDMA Femtocells with Heterogeneous Services. *IEEE Trans. Commun.* **2014**, *62*, 2366–2377. [CrossRef]
13. AlSobhi, W.; Aghvami, A.H. QoS-Aware Resource Allocation of Two-tier HetNet: A Q-learning Approach. In *2019 26th International Conference on Telecommunications (ICT), Hanoi, Vietnam, 8–10 April 2019*; IEEE: Piscataway, NJ, USA, 2019; pp. 330–334.
14. Miao, S.; Song, T.; Zhang, L.; Sari, H.; Gui, G. Multi-Efficiency Based Resource Allocation for Cognitive Radio Networks with Deep Learning. In *2018 IEEE 10th Sensor Array and Multichannel Signal Processing Workshop (SAM), Sheffield, UK, 8–11 July 2018*; IEEE: Piscataway, NJ, USA, 2018; pp. 56–59.
15. Zhang, H.; Wang, T.; Shen, H. A Resource Allocation Algorithm for Ultra-Dense Networks Based on Deep Reinforcement Learning. *Int. J. Comput. Commun. Control* **2021**, *16*, 4189. [CrossRef]
16. He, B.; Meng, Y.; Tang, L. An Off-Policy Reinforcement Learning-Based Adaptive Optimization Method for Dynamic Resource Allocation Problem. *IEEE Trans. Neural Netw. Learn. Syst.* **2025**, *36*, 3504–3518. [CrossRef] [PubMed]
17. Yin, C.; Zhang, Y.; Dong, P.; Zhang, H. Deep reinforcement learning-based joint task offloading and resource allocation in multipath transmission vehicular networks. *Trans. Emerg. Telecommun. Technol.* **2024**, *35*, e4930. [CrossRef]
18. Aghapour, Z.; Sharifian, S.; Taheri, H. Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed AI execution tasks in IoT edge computing environments. *Comput. Netw.* **2023**, *223*, 109577. [CrossRef]
19. Li, K.; Zhao, J.; Hu, J.; Chen, Y. Dynamic energy efficient task offloading and resource allocation for NOMA-enabled IoT in smart buildings and environment. *Build. Environ.* **2022**, *226*, 109513. [CrossRef]
20. Nath, S.; Wu, J. Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems. *Intell. Converg. Netw.* **2020**, *1*, 181–198. [CrossRef]
21. Kamal, A.; Roshdy, H.S.; Said Hassan, N.R.; Mohamed, A.W. An Improved Gaining Sharing Knowledge-Based Algorithm for Solving Resource Allocation Problems. *Int. J. Comput. Intell. Syst.* **2025**, *18*, 197. [CrossRef]
22. Khassiba, A. An efficient backtracking heuristic for the resource allocation problem with compatibility and exclusivity constraints. *J. Heuristics* **2024**, *31*, 4. [CrossRef]
23. Qi, X.; Khattak, S.; Mustafa, S.S.; Khan, I. GA based energy efficiency resource allocation in heterogeneous networks with MRC. *IET Commun.* **2022**, *16*, 369–377. [CrossRef]
24. Yilmaz Acar, Z.; Başçiftçi, F. Solving Multi-Objective Resource Allocation Problem Using Multi-Objective Binary Artificial Bee Colony Algorithm. *Arab. J. Sci. Eng.* **2021**, *46*, 8535–8547. [CrossRef]
25. Su, C.; Chen, Z.; Zhu, Z.; Dai, H.; Chang, J. Solving a class of resource allocation problem under dynamic constraints: A predefined-time distributed optimization scheme. *ISA Trans.* **2025**, *165*, 295–307. [CrossRef] [PubMed]
26. Guo, F.; Chen, X.; Yue, M.; Jiang, H.; Chen, S. Distributed Optimization for Resource Allocation Problem with Dynamic Event-Triggered Strategy. *Entropy* **2023**, *25*, 1019. [CrossRef] [PubMed]
27. Lian, M.; Guo, Z.; Wang, X.; Wen, S.; Huang, T. Adaptive Exact Penalty Design for Optimal Resource Allocation. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *34*, 1430–1438. [CrossRef]

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.