


Article

An Efficient Task Synthesis Method Based on Subspace Differential Patterns for Arrangements of Event Intervals Mining in the Avionics Cloud System Architecture

Xiaoxu Dong ¹, Xin Wang ², Ling Peng ², Miao Wang ^{1,*}  and Guoqing Wang ¹

¹ School of Aeronautics and Astronautics, Shanghai Jiao Tong University, Shanghai 200240, China

² China Ship Development and Design Center, Wuhan 430000, China

* Correspondence: miaowang@sjtu.edu.cn

Abstract: Avionics Cloud is a new multi-platform avionics system architecture that provides dynamic access, resource pooling, intelligent scheduling, on-demand service and other cloud computing features. Using Avionics Cloud to rationalize the order of multi-flight platform task execution and realize multitask synthesis is a challenging problem. In this paper, we propose an Efficient Task Synthesis Method based on Subspace Differential Patterns for Arrangements of Event Intervals Mining-DiMining. For tasks executed in a multi-platform Avionics Cloud system with dynamic characteristics of time intervals, DiMining is proposed. The algorithm mines the differential frequent task execution event interval patterns related to execution efficiency from the scenario dataset with high execution efficiency and the scenario dataset with low execution efficiency in order to identify key task patterns related to execution efficiency and improve the task synthesis design efficiency of the multi-platform Avionics Cloud system. Furthermore, in order to improve the mining efficiency of the algorithm, this algorithm designs a variety of pruning strategies to ensure that two differential time interval patterns with high and low functional execution efficiency are mined at one time without preserving the set of candidate items. The experimental results show that the DiMining algorithm is more efficient than the traditional algorithm on the open dataset. The DiMining algorithm is used to mine the 350-field high-efficiency operation scenario dataset and the 350-field efficiency operation scenario dataset under the constructed typical UAV cluster co-detection task scenarios. Based on the simulation results, the DiMining algorithm is able to effectively support the design of multi-platform Avionics Cloud system task synthesis architecture and improve the efficiency of UAV cluster collaborative detection.

Keywords: Avionics Cloud system; task synthesis; subspace differential patterns; arrangements of event intervals mining



Citation: Dong, X.; Wang, X.; Peng, L.; Wang, M.; Wang, G. An Efficient Task Synthesis Method Based on Subspace Differential Patterns for Arrangements of Event Intervals Mining in the Avionics Cloud System Architecture. *Aerospace* **2023**, *10*, 249. <https://doi.org/10.3390/aerospace10030249>

Academic Editor: Carlos Insaurralde

Received: 19 January 2023

Revised: 3 March 2023

Accepted: 3 March 2023

Published: 5 March 2023



Copyright: © 2023 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 (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Currently, open system architecture ideas have been used in avionics software development to reduce the cost of avionics system development. Open system architecture aims to enhance the portability and reconfigurability of avionics software in avionics systems with different hardware bases, thus supporting the continuous upgrading of avionics systems [1–5]. The Future Airborne Capable Environment (FACE) architecture is an emerging open and reconfigurable avionics system architecture in recent years [6]. Under the FACE architecture, single-platform avionics systems cannot fully meet the increasingly complex systemic task requirements. Today, avionics equipment usually focuses on human resources, security and perceptual impact [7,8]. With the increase in systemic applications, future avionics systems must evolve toward cross-platform synthesis. Avionics systems have undergone four generations of development from discrete platform avionics systems [9]. The existing research on multi-platform avionics systems proposed a three-tier architecture for Avionics Cloud computing and the concept of “avionics cloud”

system [10,11]. The Avionics Cloud computing environment has a set of distributed computing frameworks, including a computing model, a hardware framework, and a software model [12]. Most of the existing studies are based on a cloud computing mechanism to process information, which cannot cover the organization and management of tasks, functions, and resources. It is important to study the Avionics Cloud architecture from the perspective of task organization, function organization, and resource organization.

The Avionics Cloud system provides resource virtualization capabilities that can identify and connect the various flight units in the system. Ultimately, networked collaborative capabilities can be realized to efficiently perform tasks such as long-range communications, route navigation, and disaster prediction. The task allocation of multi-flight platform avionics task system in the avionics system cloud organization architecture is a difficult point to study. The article [13] proposes a three-branch clustering-based scheduling algorithm for cloud task optimization to improve scheduling efficiency by granularizing tasks in the cloud. The article [14] uses a deep reinforcement learning scheduling algorithm based on an action branching architecture improvement to comprehensively sense potential correlations between scheduling jobs in the cloud system. The article [15] describes recent advances in the problem of resource scheduling optimization in cloud-side collaboration and gives a reference scheme for resource scheduling optimization adapted to the characteristics of the scenario. Existing studies on task allocation in cloud architectures have not been analyzed from the perspective of task synthesis. Task synthesis refers to the gradual decomposition of requirements on top of unit resources and functions with corresponding capabilities. To cope with the increasing number and complexity of task patterns in the context of task integration, data mining techniques can be applied to assist in task sequence analysis. The specific application is as follows: Avionics Cloud completes the optimal task sequence with the help of a data mining algorithm based on a historical task database through known task objectives, to improve the efficiency of task completion. To effectively balance the system load and data mining as a means, the study of task allocation relationship based on spatiotemporal data mining for Avionics Cloud architecture becomes a theoretical problem that needs to be solved at present.

Traditional sequential pattern mining methods are constrained based on support or frequency. Because efficiency with sequential pattern mining does not satisfy the downward closure property, it needs to reduce its candidate set or search space. Therefore, some attributes need to be defined to reduce the search space. Fournier [16] et al. combine sequential pattern mining with dimensional pattern mining, time intervals, automatic clustering with valued actions, and closed sequence mining. The SARA algorithm [17] not only mines the relationship between two events, but also reveals the time period in which each event occurs and ends. The HUFTI-SPM [18] algorithm uses any two-time, frequency, or utility constraints to mine sequential patterns with time intervals. The VertTIRP algorithm [19] uses the temporal relationship of pairwise tests for ranking to speed up the mining process. The Z-Miner algorithm [20] proposes an efficient algorithm for mining time-interval data. All the above algorithms focus on the constraint of time interval but are unable to extract the key patterns from the difference samples. The SDC algorithm [21] proposed that the concept of differential support can be used to mine differential frequent function patterns, i.e., a set of functions that are frequent in one dataset but not represented or represented in the opposite way in another dataset, but the algorithm requires high storage space and computational complexity and strong constraints. The above algorithms have different drawbacks and are not applicable to assist Avionics Cloud architecture design for data mining. In this paper, we propose the DiMining algorithm for complete and efficient mining of differential frequent time interval patterns. The algorithm utilizes a memory-efficient data structure and a new link table pruning method for analyzing efficient task-matching patterns between units in a task scenario dataset. The contributions of the DiMining algorithm proposed in this paper that distinguish it from existing frequent algorithms are as follows:

(1) The algorithm mines the differential dataset and can mine the typical patterns that affect the efficiency of task execution.

(2) The algorithm uses a relational pair storage data structure graph to store the original data, ensuring that two kinds of differential time interval patterns with high and low execution efficiency are mined at one time, avoiding losses from secondary mining and improving mining efficiency.

(3) The algorithm mines by linear expansion and uses multiple pruning strategies to mine the differential frequent time interval patterns.

(4) The algorithm is suitable for the application of finding resources suitable for performing tasks in avionics systems.

The remainder of this paper is organized as follows: Section 2 describes the Avionics Cloud architecture. Section 3 describes the underlying definitions. Section 4 describes the proposed DiMining algorithm in detail. Section 5 compares the differences in mining efficiency between this paper's algorithm and existing algorithms, and a typical task scenario is designed for Avionics Cloud optimization before and after effectiveness validation. Conclusions are given in the last section.

2. Avionics Cloud Architecture

2.1. Avionics Cloud Organizational Structure

Avionics Cloud organizational structure consists of an application cloud platform (Software-as-a-Service, SaaS) for flight platform, a function cloud platform (Platform-as-a-Service, PaaS) for common application service, and a resource cloud platform (Infrastructure-as-a-Service, IaaS) for common function operation, as shown in Figure 1. In the IaaS cloud delivery model, the physical resources on each flight platform are virtualized and processed to form virtual resources located in the cloud. The PaaS cloud delivery model includes all the functions involved in the process and provides a pre-configured environment that is used to build and deploy cloud services and solutions. SaaS shared a cloud service delivery model.

(1) Application Cloud Platform (SaaS)

The application cloud platform (SaaS) is located at the top layer. In the application cloud platform (SaaS), task services are established, and service scheduling, service analysis, and service organization are realized through distributed processing.

(2) Function Cloud Platform (PaaS)

The function cloud platform (PaaS) is located in the middle layer. The function cloud platform mainly uses algorithms to data mine the scene dataset to get function-matching patterns, and feeds the mining results to the resource cloud platform to realize resource allocation.

(3) Resource Cloud Platform (IaaS)

The resource cloud platform (IaaS) is located at the bottom layer. In the resource cloud platform, navigation resources, communication resources, etc. are virtualized. Logical virtual spaces are established on each flight platform, and each physical entity in the Avionics Cloud is scheduled based on task requirements to collaborate to complete tasks.

Based on the analysis of the layered view of the cloud, it is clear that how to conduct intelligent scheduling of resources within the Avionics Cloud is a problem that needs to be solved at present.

The practical application of a layered view of the Avionics Cloud is illustrated using a shipboard aircraft as an example, as shown in Figure 2. The shipboard aircraft provides task organization modes such as air interception, air strike, strike against islands, and cyber-attack. The management of shipboard aircraft under the Avionics Cloud architecture is achieved through a virtual formation-based task organization cloud processing.

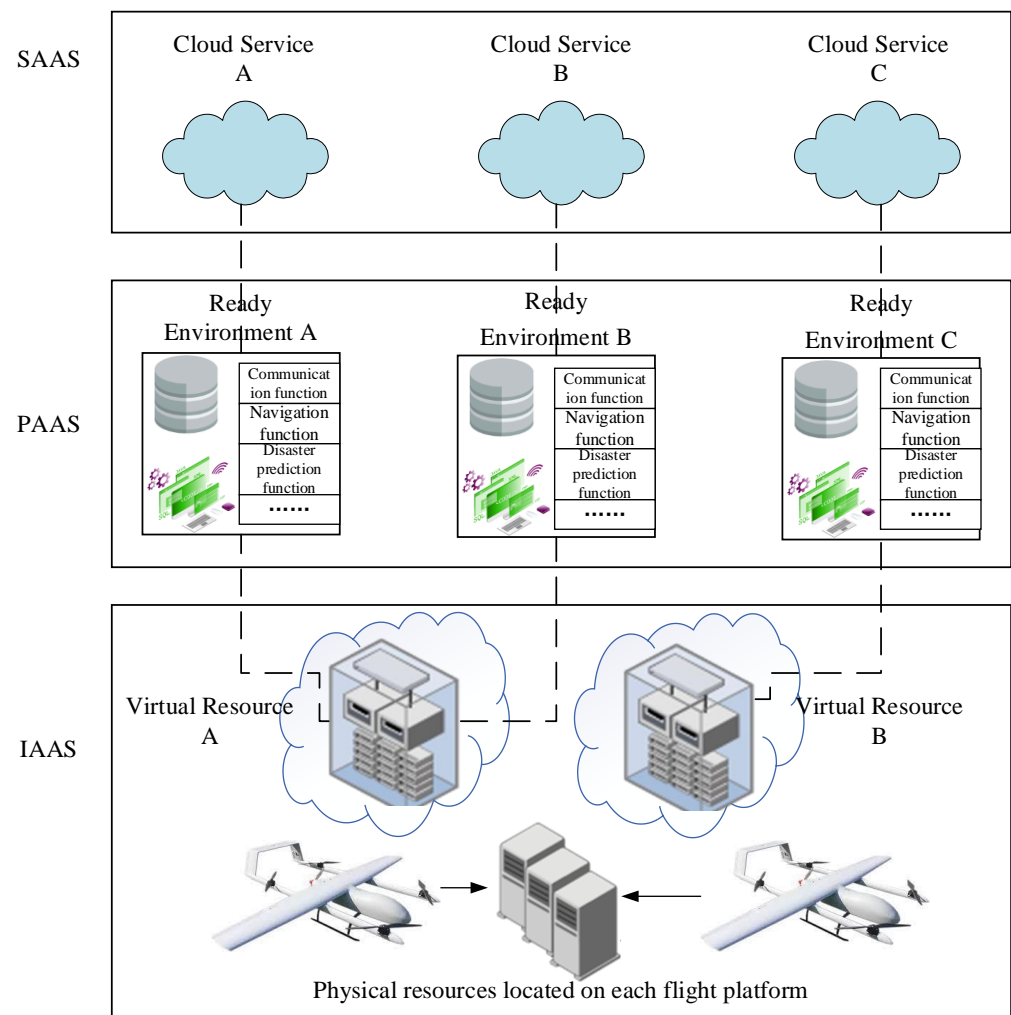


Figure 1. Avionics Cloud IaaS, PaaS, SaaS layered view.

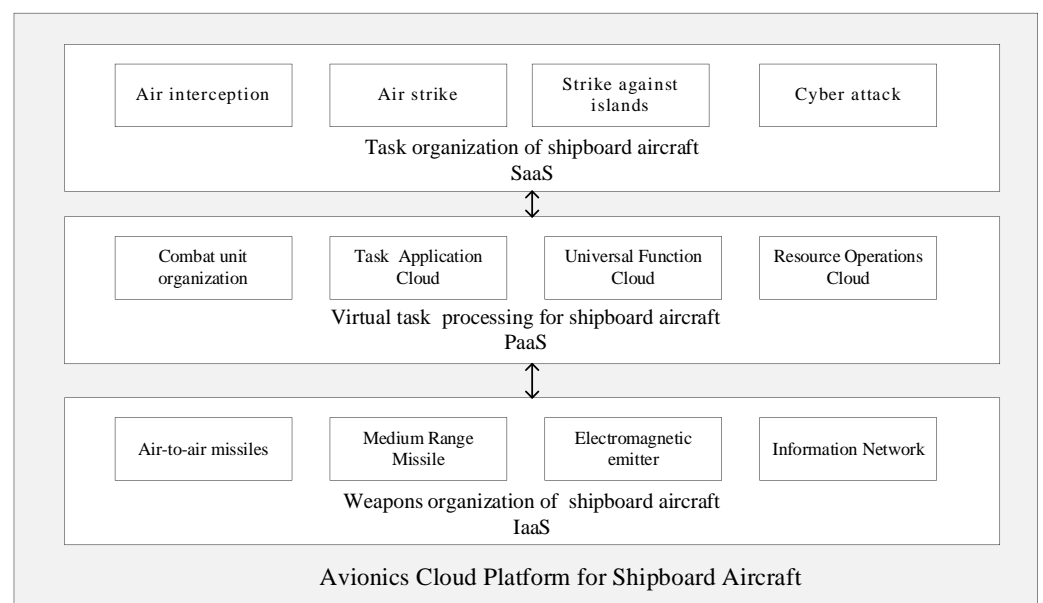


Figure 2. Layered view of Avionics Cloud IaaS, PaaS, and SaaS for shipboard aircraft.

The task operation management of the shipboard aircraft is built on the basis of the management of the Avionics Cloud. It realizes task organization and operation processing based on task sensing and information acquisition of the shipboard aircraft. It completes the weapon control and delivery of the shipboard aircraft through the task output and weapon management of the shipboard aircraft, as shown in Figure 3.

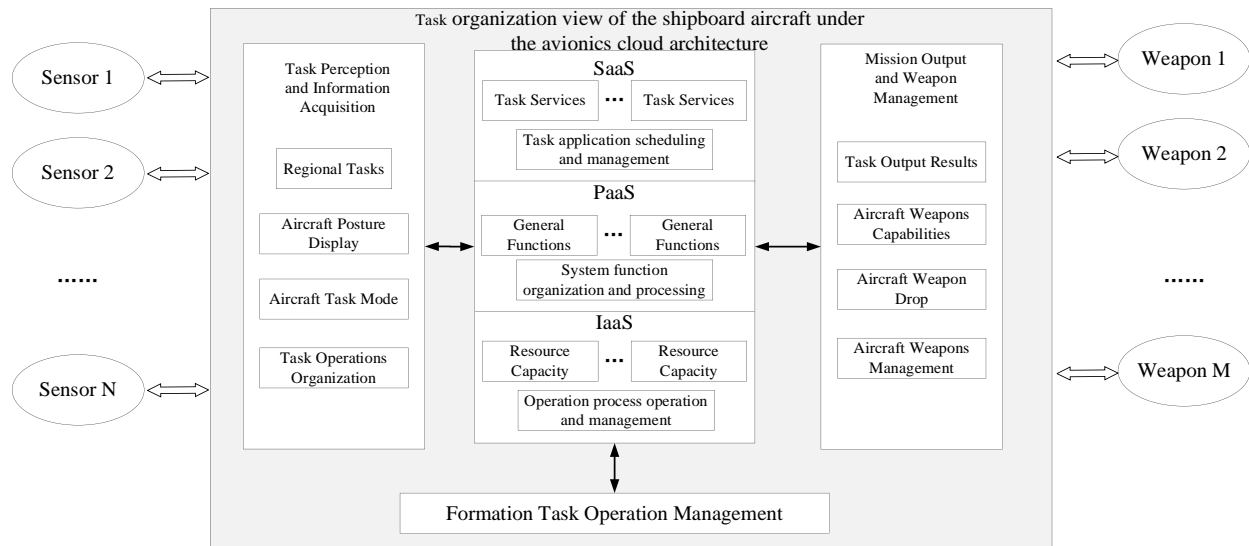


Figure 3. Task organization view of the shipboard aircraft under the Avionics Cloud architecture.

2.2. Avionics Cloud Logical Architecture

The Avionics Cloud hierarchical logical architecture can be divided into three layers: task layer, functional layer, and resource layer, as shown in Figure 4.

(1) Task layer

The task layer based on the Avionics Cloud architecture refers to the process that requires the collaboration of resources and functions in multiple flight units in order to meet the distributed system requirements' goals, which is essentially a process of complex system state change. The requirement goals of a systematic process often need to be satisfied by the collaboration of multiple unit tasks, so task synthesis is required. Task synthesis contains two meanings. On the one hand, the requirements are gradually broken down into unit resources and functions with corresponding capabilities. On the other hand, the appropriate sequence of units is selected in real-time to perform tasks based on current resources, functional capabilities and the operational status of the entire system.

Therefore, the elasticity of the task organization and synthesis based on the Avionics Cloud architecture reflects the task execution units and sequence selection is based on the cloud organization's way of dynamic selection of units. Thus, it is possible to execute each task sequence in an elastic and collaborative manner to ensure the completion of the task.

The process of executing tasks is as follows:

- (1) Based on the current functional cloud and resource cloud capabilities, the requirements are gradually decomposed into platforms with corresponding capabilities according to the requirements. In the task generation phase, the resource pool of the task cloud is the platform that can perform various tasks, and the platform that can be selected to meet the task requirements is selected from the current resource pool. The task is organized to ensure that the requirements can be met. In the process of organization, the process of how many platforms need to be selected from the resource pool and how to collaborate between platforms to generate plans is undertaken in the task cloud based on task dynamic demand elasticity.
- (2) Based on the task plan, a suitable sequence of units is selected to execute tasks in real-time based on the current platform operation status. In the task execution phase,

based on the changes in demand caused by changes in the task plan and current scenario, as well as battle damage, etc., the platform is automatically increased or decreased from the resource pool to ensure that the demand is met. This means that the task execution can be completed automatically and flexibly during the execution process.

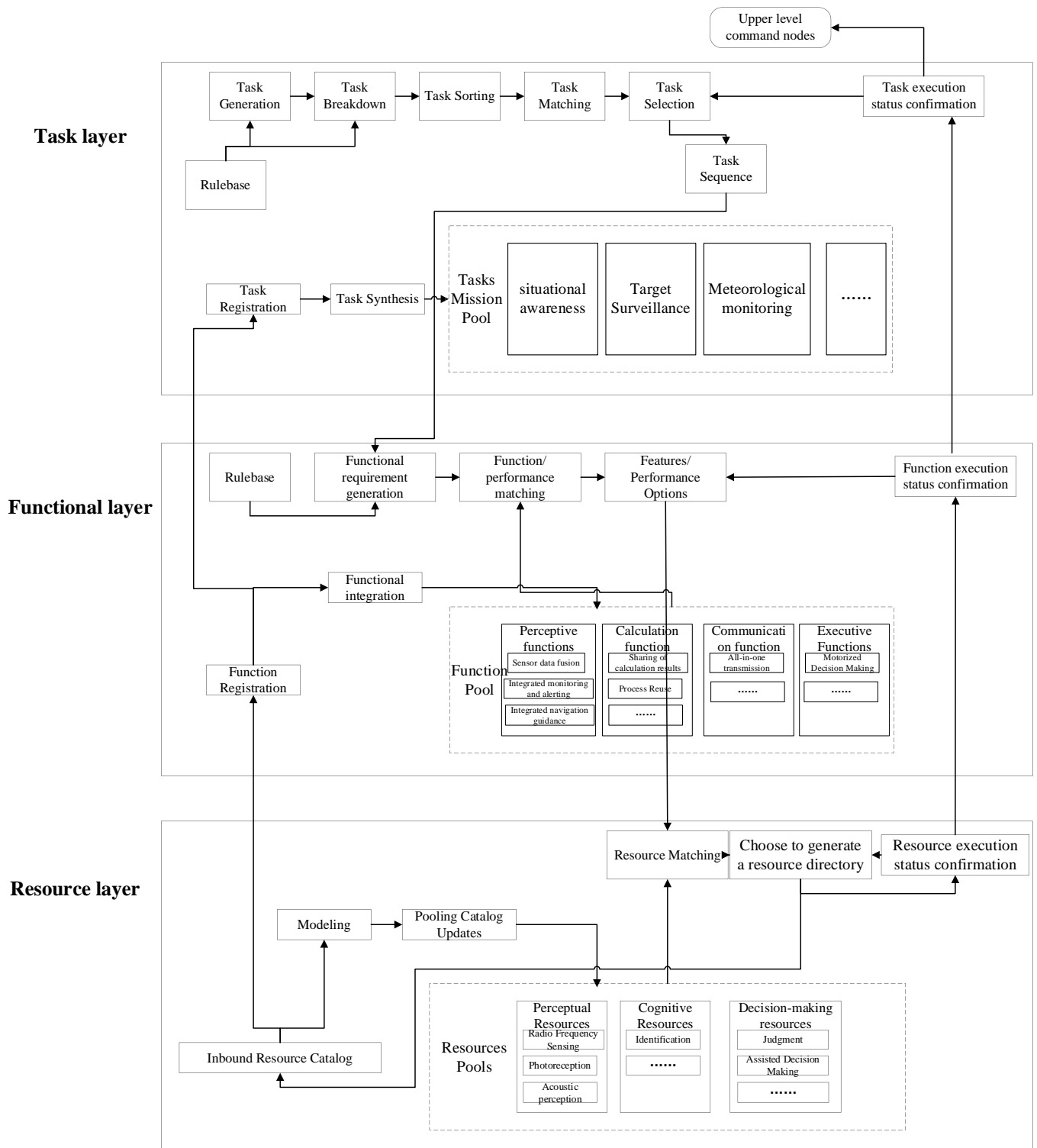


Figure 4. Avionics Cloud hierarchical logic architecture.

(2) Functional layer

The functional information fusion of Avionics Cloud is reflected in four aspects: sensing functional information fusion, computing functional information fusion, communication functional information fusion, and execution functional information fusion. Among them, the fusion of perceptual information is mainly reflected in the fusion of sensor data, integrated monitoring and alerting, and integrated navigation guidance. The computation function information fusion is reflected in the sharing of computation results and the reuse of the processing process. The communication function information fusion is reflected in the integration of transmission. The information fusion of the execution function is mainly reflected in the aspect of maneuvering and decision-making.

The process of function execution is as follows:

- (1) Task-oriented requirements and selection of available functions from the pool of functional resources. When, the function organization and fusion method meet the task capability requirements, this is called the function generation and organization phase. In this phase, facing the dynamic changes of the task, the functional cloud dynamically selects the required functions. In addition, based on the functional capability demand of real-time fusion to enhance the capability, the elasticity achieves dynamic function generation and organization.
- (2) Function execution is based on the current function's required physical resource operation state, and real-time selection of the appropriate physical platform to execute the function, called function execution. In the functional execution phase, based on the functional organization and changes in demand caused by changes in the current scenario and battle damage, functional modules are automatically added or reduced from the resource pool to ensure that the capability requirements are met. This means that functional execution can be performed automatically and flexibly during the execution process.

(3) Resource Layer

Resource pooling is oriented to functional capability requirements, and the appropriate resources are scheduled and selected from the currently registered resources to provide performance capabilities to ensure that capability requirements are met. Similar to functions and tasks, resource pool resiliency is also reflected in two aspects: resource generation and resource execution. Resource organization elasticity is based on the upper-level functional capability requirements, and automatically matches the appropriate resources from the current resource pool required to satisfy the functional capability. Similarly, to support different layers of tasks and their functional capability matching, resources in the resource pool will be characterized according to the granularity of resource capabilities required for task execution.

The resource pool can achieve resource generation elasticity by scheduling the required resources from the resource pool to support functional capability execution according to different tasks and their required capabilities. If during the resource generation process, it is found that the resources required to satisfy the functional capabilities cannot be dispatched from the resource pool, a warning is issued to the functional layer to resiliently organize other functional capability policies from the functional resource pool.

Resource organization is similar to functional generation. Resource organization is the process of dynamically scheduling the required capability resources from the resource pool in response to the task functional capability requirements. Resource execution is the process of executing resource performance in real-time based on the resource capability state and according to the resource scheduling method. However, in the actual execution process, the environment and the situation change rapidly, and our units and resource capabilities can fail with the task execution and other situations. Therefore, during the execution of real-time resource scheduling, the resources in the resource pool are dynamically adjusted and resources are flexibly scheduled to ensure that the functional capability requirements

are met. If the current capacity demand cannot be met through resource scheduling, a warning is sent to the functional layer.

The hierarchical logical architecture of the joint fleet under the Avionics Cloud architecture is shown in Figure 5. After the fleet command center specifies the basic action plan, the task sequence is obtained in the Avionics Cloud task layer with the assistance of intelligent scheduling algorithms for task synthesis. The specific tasks of the combat unit include task organization, combat coordination, target strike and weapon management. The task synthesis results are passed to the Avionics Cloud architecture functional layer to generate functional requirements. The matching functions are selected and passed to the Avionics Cloud architecture resource layer for resource matching to complete the process of resource, function and task execution status confirmation. When there is a multitask conflict caused by resource occupation and failure, the Avionics Cloud makes a plan adjustment to coordinate the multitask conflict.

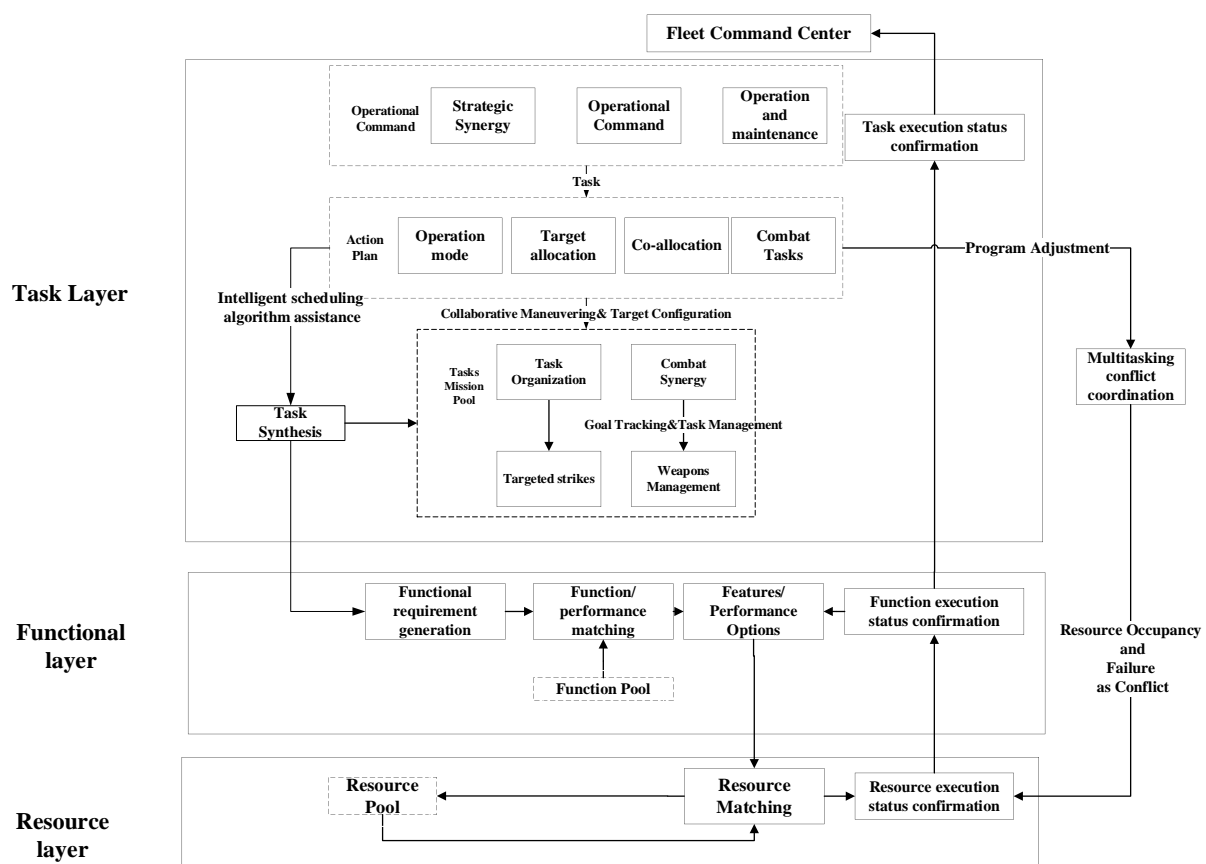


Figure 5. Joint fleet hierarchical logical architecture under Avionics Cloud architecture.

Analyzing the organizational and logical architecture of the Avionics Cloud, it is clear that using the Avionics Cloud to rationalize the order of task execution of multiple flight platforms and to achieve multi-task synthesis is the basis for realizing the Avionics Cloud. It can provide input for subsequent functional fusion and resource allocation. This paper proposes a new data mining algorithm—the DiMining algorithm, which can mine and analyze efficient task patterns by mining and analyzing high and low utility scenarios, to lay the foundation for task synthesis and task architecture design. A typical task scenario is chosen as an example for simulation verification. By analyzing the task assignment relationship between flight platforms in the task scenario dataset, the optimal task-matching pattern is obtained, which provides a basis for the task architecture design.

3. Problem Description

Definitions 1 and 2 give the definition of the dataset.

Definition 1. Given a task dataset $D = \{T_1 \dots T_n\}$. Each T contains desired serial number id , desired start-time, desired end-time, and start-time is less than end-time.

Definition 2. Let I denote the set of all possible tasks. Task $i \in I$ is maintained over a period of time $[s, e)$, denoted as (i, s, e) , where s is the start time and e is the end time. (i, s, e) is called the state interval. The state sequence in I consists of a series of state intervals $\langle (i_1, s_1, e_1), (i_2, s_2, e_2), \dots, (i_n, s_n, e_n) \rangle$, which $s_i < s_{i+1}, s_i < e_i$.

All the data in dataset D with the same serial number id are grouped together and sorted by incremental start time, then the dataset can be converted into a set of state sequences. Dataset D can be considered as a collection of state sequences.

Example 1. By definition, I may contain duplicate entries, and Tables 1 and 2 depict two examples of datasets D , which are represented as follows.

Table 1. Dataset D1.

Number of T	Data
T1	(A, 1, 5) (B, 1, 5) (C, 1, 5) (D, 1, 5) (B, 7, 13) (A, 15, 20)
T2	(A, 1, 8) (B, 1, 8) (C, 1, 8) (D, 1, 8) (A, 15, 20)
T3	(A, 1, 5) (B, 1, 5) (C, 7, 13) (B, 15, 20)
T4	(A, 1, 17) (D, 9, 11) (C, 11, 13) (D, 14, 18) (B, 18, 20)

Table 2. Dataset D2.

Number of T	Data
T1	(B, 1, 8) (A, 7, 17) (D, 9, 11) (C, 10, 12) (D, 14, 18) (A, 18, 20)
T2	(A, 1, 17) (D, 9, 11) (C, 10, 12) (D, 14, 18) (A, 18, 20)
T3	(A, 7, 17) (C, 9, 13) (D, 12, 14) (C, 14, 18) (A, 18, 20)
T4	(A, 1, 17) (C, 11, 13) (D, 12, 14) (C, 14, 18) (A, 18, 20)

Definition 3 describes the definition of possible relationships between time period data.

Definition 3 (temporal relationship). The execution relationship between tasks I_A and I_B can be divided into the following five types according to the execution time: follows, meets, overlaps, contains, and matches (without distinguishing between left and right), as shown in Figure 6. A and B in the figure represent the time periods of tasks I_A and I_B .

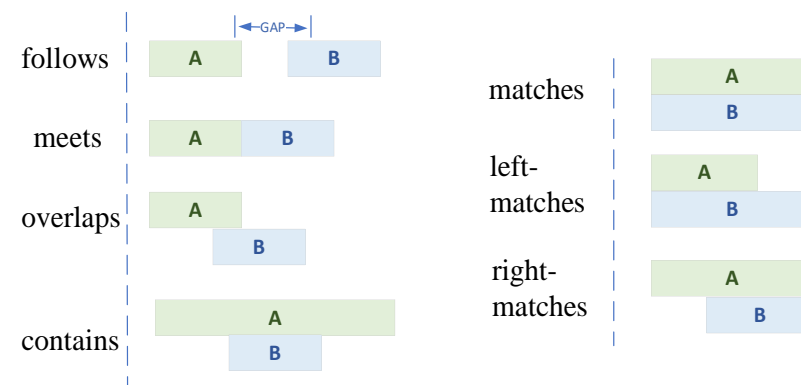


Figure 6. Five types of task relationships.

$r(I_A, I_B) = f$: A follows B if:
 $(B.s - A.e) > 0$ and $(B.s - A.e) < gap$
 $r(I_A, I_B) = m$: A meets B if:
 $|B.s - A.e| = 0$ and $(B.s - A.s) > 0$
 $r(I_A, I_B) = o$: A overlaps B if:
 $(B.s - A.s) > 0$ and $(A.e - B.s) > 0$ and $(B.e - A.e) > 0$
 $r(I_A, I_B) = c$: A contains B if:
 $(B.s - A.s) > 0$ and $(A.e - B.e) > 0$
 $r(I_A, I_B) = ma$: A matches B if:
 $|A.e - B.e| \leq 0$ or $|A.s - B.s| \leq 0$

Definitions 4 and 5 describe the definitions of frequent patterns that we want to mine.

Definition 4. The number of transactions that contain items is called the frequency of occurrence of items. The frequency of occurrence of item X , denoted as $\text{sup}(X)$, is the number of supported transactions containing X . The set of transactions supporting item X is represented as Γ_X . If $X \subseteq T_q$, then say T_q supports item X . Therefore, $\text{sup}(X) = |\Gamma_X|$. Let the user-specified minimum support threshold be α . If $\text{sup}(X) \geq \alpha \times |D|$, then X is called a frequent pattern (FP) in dataset D .

Definition 5. P is a task set and the Subspace Differential Frequent (SDF) support of P can be defined as:

$$SDF(P) = \frac{N_A(P)}{|A|} - \max_{\forall p_i, p_j \in P} \frac{N_B(p_i p_j)}{|B|}, \quad (1)$$

where $N_A(P)$ is the number of frequent samples of P in dataset A . $N_B(p_i p_j)$ is a binomial subset $p_i p_j$ of P the number of frequent samples in the data set B . $\max_{\forall p_i, p_j \in P} \frac{N_B(p_i p_j)}{|B|}$ is the proportion of the binomial subset of P in dataset B when the maximum number of frequent samples is in dataset B [21].

Example 2. Taking the databases shown in Tables 1 and 2 as examples, the differential frequent support of pattern A follows A is $SDF(A \text{ follows } A) = 1 - 0.5 = 0.5$.

4. Algorithm Description

In this section, we introduce the DiMining algorithm in detail. The algorithm mines frequent patterns that satisfy the definition of variance from a large amount of scene data to analyze the patterns associated with efficiency factors. The general framework of the algorithm is shown in Figure 7.

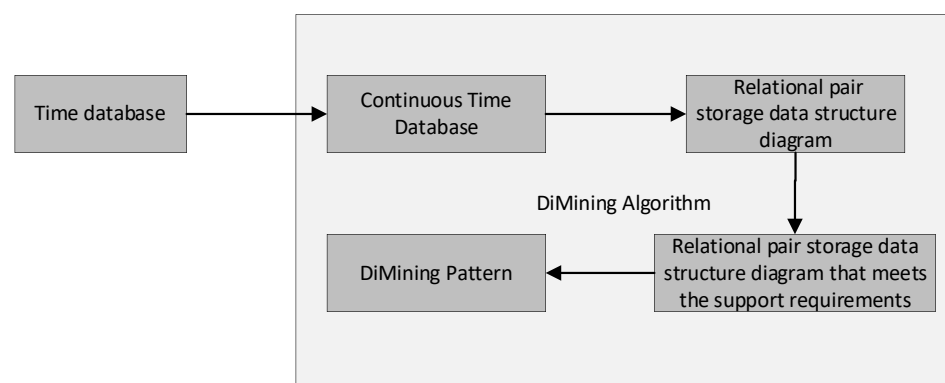


Figure 7. DiMining Algorithm framework diagram.

4.1. Data Structure

In differential frequent itemset mining, the computation of support is very time consuming. We design a data structure to store the transaction information and support degree of each item. The data structure consists of three parts: head node, struct node and

time node. The head node stores the head node and the relationships it may contain. The struct node stores the nodes that have a relationship with the head node and the number of T. The time node stores the time period data of the head node and the node in the relationship. Specific description is as follows.

For each head node i , $r(m) = \{stid_i\}$ collects the set of pointers of its relation pairs. $r(m)$ is as shown in the head note section in Figure 8, and follows, meets, overlaps, contains, and matches points to the corresponding link table node respectively.

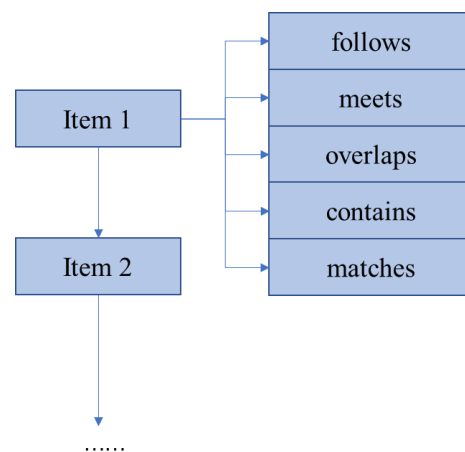


Figure 8. Head node of data structure diagram.

The link table nodes are shown in the struct note section in Figure 9. Transaction link 1 points to the row number where the current pattern appears in dataset 1. Support 1 indicates the support of the current pattern in dataset 1. Transaction link 2 points to the row number where the current pattern appears in dataset 2. Support 2 indicates the support of the current pattern in dataset 2.

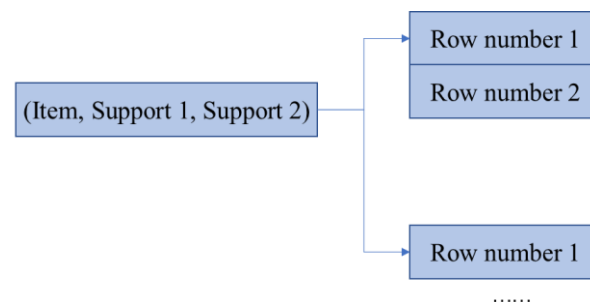


Figure 9. Struct node of data structure diagram.

The time series pattern has a start time and an end time for each item. Therefore, for a transaction node in a link table node needs to index a time node that indicates the start and end time of a pair of itemsets under the current relationship, as shown in the time note section in Figure 10. When in a row, there may be pairs of itemsets with the same relationship in time intervals. In order not to lose the mining information, the time information is maintained for all relationship pairs.

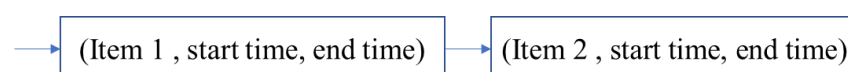


Figure 10. Time node of data structure diagram.

The data is characterized by temporal intervals, and the sets of items are temporal and occur multiple times. Therefore, it is necessary to store the temporal information of each time interval relationship pair, as shown in Figure 11.

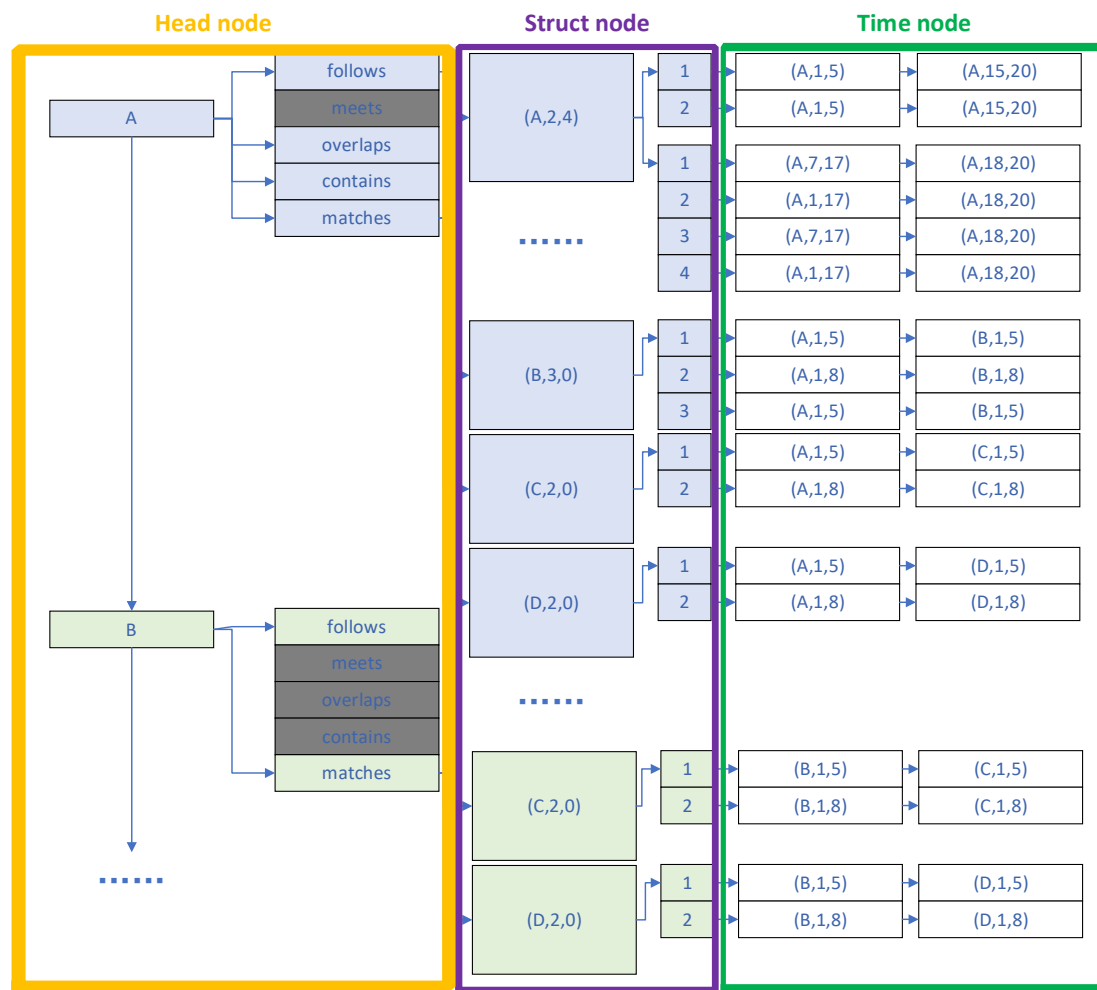


Figure 11. Partial relational pair storage data structure diagram.

Example 3. The generated partial relational pair storage data structure diagram from Tables 1 and 2, is shown in Figure 11.

According to Definition 5, it is necessary to prune the branches in the relational pair storage data structure that do not satisfy the differential frequent support threshold.

4.2. Pruning Strategy

After constructing relational pair storage data structure, this algorithm follows a linear extension for mining.

Example 4. Scanning head node A, $A \text{ matches } B$ is the first extended node, which satisfies the definition of subspace differential frequent patterns. The potential candidate patterns of $A \text{ matches } B$ are $B \text{ matches } C$. After the intersection of two relationship patterns transaction link 1 and transaction link 2, the values of transaction link 1 are 1 and 2. Transaction link 2 is the empty set, which satisfies the threshold of variance support. Therefore, $A \text{ matches } B \text{ matches } C$ is a subspace differential frequent pattern.

For the current extended pattern, according to Definition 5, we need to record the information of transaction link 1 and transaction link 2 after the intersection and the maximum value of the binomial subset in the current extended pattern to facilitate subsequent mining.

Example 5. After recording the information of transaction link 1 and transaction link 2 of $A \text{ matches } B \text{ matches } C$ and the maximum value of binomial subset, we find the candidate relationship pairs

of $AmatchesBmatchesCmatchesD$ in a linear expansion way from D is the branch with D as the head node to find the candidate pairs of $AmatchesBmatchesCmatchesD$. Since the set of transaction of the candidate pattern and the set of transaction of $AmatchesBmatchesCmatchesD$ are empty after intersection. Therefore, $AmatchesBmatchesCmatchesD$ is a subspace differential frequent time interval pattern and can be output with a differential support of $2/4 = 0.5$.

By observation, the complex relationship between each time interval is the main bottleneck in mining temporal patterns. We propose a pruning strategy to solve this critical problem.

Lemma 1. For the currently extended pattern P , N is its set of precursor candidates, M is its set of candidates, and M_i is any candidate pattern in M . For any precursor candidate pattern N_j in N , P can be pruned if the following strategies are simultaneously satisfied.

- (1) The set of transaction link 1 of PM_iN_j is a subset of the set of transaction link 1 of PN_j .
- (2) The maximum binomial subset support of PM_iN_j in dataset 2 is not less than the maximum binomial subset support of PN_j .
- (3) PM_iN_j is also a differential frequent pattern.
- (4) For the remaining candidate patterns M_k in M also all satisfy the above three strategies.

Proof. (1) The algorithm uses an extension depth-first mining maximum differential frequent pattern. If the transaction link 1 set of the current candidate node is a subset of the transaction link 1 set of a predecessor candidate node, then the transaction link 1 set of PM_iN_j must be equal to the transaction link 1 set of PN . That is, the set of transaction link 1 generated by the current candidate node can be generated by that predecessor node.

(2) If the maximum binomial subset support of PM_iN_j in dataset 2 is not less than the maximum binomial subset support of PN_j , it means that the current candidate node is added to the current extended node differential frequent pattern, and the maximum binomial subset support of the newly introduced subset of length 2 in dataset 2 is not the maximum, and it does not affect the maximum binomial subset of PM_iN_j in dataset 2.

(3) Candidate and antecedent nodes that satisfy the above two strategies do not necessarily satisfy the differential frequent support. According to the idea of pruning by antecedent test, if the current candidate node is pruned, it must be generated by some antecedent. Therefore, the current extended node must be a differential frequent pattern with some precursor node.

(4) We use the counterfactual method to prove this rule in two cases. First, assume that PM_i can be pruned when another candidate function M_j of P does not simultaneously satisfy the above three strategies. That is to say, there does not exist any precursor candidate N_j such that PM_jN_j is a differential frequent pattern. Then PM_j and PM_i may produce a differential frequent pattern PM_jMi . Since the differential frequent support satisfies the Apriori principle, a precursor candidate N_j must not be found. The precursor candidate node N_j can produce a PM_jN_jMi , then PM_jN_j is a new differential frequent pattern. Therefore, PM_i should not be pruned, which contradicts the hypothesis. Then, suppose another candidate node M_j of P satisfies the above three strategies and at the same time is a precursor candidate N_i distinct from N_j satisfied by M_i . That is to say, PM_i can be pruned as well when M_j and M_i satisfy the above three strategies with different precursor candidates. Because the differential frequent support satisfies the Apriori principle, at this time PM_j and PM_i cannot find the same precursor candidate node satisfying the differential frequent support. That is to say, PM_jN_j is a new differential frequent pattern, so PM_i should not be pruned, which contradicts the assumption. In summary, a candidate node can be pruned only if it satisfies all the above strategies at the same time. \square

Example 6. After extending the $AmatchesB$ branch, the current extended pattern P is $AmatchesC$, the precursor candidate set N includes $AmatchesB$, the candidate set M includes $CfollowsB$, etc. The transaction link 1 set of $AmatchesCfollowsB$ is a subset of the transaction link 1 set of $AmatchesBmatchesC$. The maximum binomial subset support of $AmatchesBmatchesCfollowsB$ in dataset

2 is not less than the maximum binomial subset support of $AmatchesBmatchesC$. The differential frequent support of $AmatchesBmatchesCfollowsB$ is 0.25, which is not a differential frequent pattern, so $AmatchesC$ cannot be pruned.

Example 7. The current extended pattern P is $AmatchesD$, the precursor candidate set N includes $AmatchesB$, and $AmatchesC$, the candidate set M includes $DfollowsA$, $DfollowsB$, and so on. The candidate pattern $DfollowsA$ is judged first. The transaction link 1 set of $AmatchesDfollowsA$ is a subset of the transaction link 1 set of $AmatchesBmatchesD$. The $AmatchesBmatchesDfollowsA$ in dataset 2. The maximum binomial subset support is not less than the maximum binomial subset of $AmatchesBmatchesD$. $AmatchesBmatchesDfollowsA$ is a differential frequent pattern; then judge the candidate pattern $DfollowsB$, $DfollowsB$ satisfies (1)(2) in the Lemma, but the differential frequent support of $AmatchesBmatchesDfollowsA$ is 0.25, not differential frequent pattern, at this time $AmatchesD$ cannot be pruned.

4.3. DiMining Algorithm

In this section we describe the general idea of the algorithm and its pseudocode. First, we introduced the main program steps of the DiMining algorithm in Algorithm 1. Then, we describe how to get the hidden relationship between time period data in Algorithm 2. Finally, we describe how to implement our proposed pruning methods in Algorithm 3.

Algorithm 1 outlines the main steps of the DiMining algorithm. First, the dataset is scanned by the `getRelation` algorithm to construct the relationship pairs of each item to store the data structure graph R (lines 1–2), and all branches that do not satisfy the differential frequent support threshold are removed from R (lines 3–6). The `growMSD` algorithm is used to determine whether the current R needs to be pruned (line 7), and if it does not need to be pruned, the differential frequent time pattern is recorded (lines 8–10), and it is cycled until all the differential frequent time patterns that meet the requirements are output (lines 11–12).

Algorithm 1 DiMining Algorithm

Data: D1: high execution efficiency dataset, D2: low execution efficiency dataset, r : current extended subspace differential frequent pattern, R : relational pairs store data structures, constraints: predefined constraints ε .

Result: Maximum differential frequent pattern set

```

1  $MSD \leftarrow \emptyset$ ;  $P \leftarrow \emptyset$ ;  $r \leftarrow \emptyset$ ;  $r' \leftarrow \emptyset$ ;
2  $R \leftarrow \text{getRelation}(D1, D2)$ ;
3 for each  $r \in R$  do
4   if the SDF of the current relation meets the constraints  $\varepsilon$  then
5     Store current relation;
6   end if
7    $\text{flag} \leftarrow \text{growDiM}(R, r, \varepsilon)$ ;
8   if  $\text{flag} == 0$  then
9      $MSD \leftarrow \text{get\_result}(r)$ ;
10  end if
11 end for
12 return DiMining Pattern

```

Algorithm 2 outlines the main steps of the `getRelation` algorithm, which is used to construct the relationship of each item to the stored data structure graph R . Dataset 1 is used as an example for illustration. First, the items $I1$ and $I2$ in database 1 are scanned and their relationships are determined (lines 1–12). After completing the scan of $I1$, $I1$ is stored into the relationship pair storage data structure graph R until the scan of all items is completed (lines 13–14). After completing the scan of all items, calculate the tl values of the items in the storage data structure graph R (lines 15–16).

Algorithm 2 getRelation (D1, D2)**Data:** D1: high execution efficiency dataset, D2: low execution efficiency dataset.**Result:** R: relational pairs store data structures

```

1 for each I1, I2 ∈ D1 (or D2) do
2   if there is follows relationship between I1 and I2 then
3     I1.follows <- I2;
4   else if there is meets relationship between I1 and I2 then
5     I1.meets <- I2;
6   else if there is overlaps relationship between I1 and I2 then
7     I1.overlaps <- I2;
8   else if there is contains relationship between I1 and I2 then
9     I1.contains <- I2;
10  else if there is matches relationship between I1 and I2 then
11    I1.matches <- I2;
12  end if
13  R ← get_head (I1.follows, I1.meets, I1.overlaps, I1.contains, I1.matches);
14 end for
15 get_tl(R);
16 return R

```

Algorithm 3 outlines the main steps of the growDiM algorithm, which is used to determine whether the current extended node r needs to be pruned. The node $tl1 > tl2$ in R is used as an example for illustration. First, R is scanned to get the precursor nodes N and candidate nodes M of the current extended node (lines 1–2). When the translink1 set of rm is not a subset of the translink1 set of rn , the output flag value is 0 (lines 3–6). Then get the $tl1$ value and $tl2$ value of rmn . When the $tl2$ value of rmn is smaller than the $tl2$ value of rn or rmn is not a differential frequent pattern, output flag value as 0 (lines 7–14). After scanning all precursor nodes and candidate nodes, if none of the above conditions are satisfied, the output flag value is 1 (lines 15–16).

Algorithm 3 growDiM(R, r, ε)**Data:** R: relational pairs store data structures, r : current extended subspace differential frequent pattern, constraints: predefined constraints ε .**Result:** flag: Determine whether the current expansion node needs to continue linear expansion.

```

1 N ← getPrecursornode(R, r);
2 M ← getCandidatenode(R, r);
3 if the set of transaction link 1 of PMiNj isn't a subset of the set of transaction link 1 of PNj then
4   flag=0;
5   return flag
6 end if
7 if the maximum binomial subset support of PMiNj in dataset 2 is less than the maximum
  binomial subset support of PNj then
8   flag=0;
9   return flag
10 end if
11 if PMiNj is not a differential frequent pattern then
12   flag=0;
13   return flag
14 end if
15 flag=1;
16 return flag

```

5. Experiment and Analysis**5.1. Efficiency Comparison**

The experiments compare the mining efficiency and results of this paper's algorithm with existing algorithms. The hardware environment for the experiments is: the processor

is Intel(R) Core(TM) i5-10300H CPU, 8G RAM, software environment: Microsoft Windows 11 operating system, algorithm programming and running environment is Microsoft Visual Studio 2015. The experimental data are obtained from clickstream dataset Kosarak and artificial datasets. According to the characteristics of the dataset, the parameter gap was set to 10 and minSDF was set to 3%.

(1) Pruning strategy analysis

The artificial dataset was used to compare pruning strategy efficiency. The DiMining with no pruning, the DiMining containing pruning strategy 1, the DiMining containing pruning strategy 2, and the DiMining containing pruning strategy 3 were compared in different size datasets to analyze the effect of the pruning strategy, as shown in Figure 12. From the figure, it can be seen that the overall running time shows the pattern of 'DiMining < DiMining containing pruning strategy 3 < DiMining containing pruning strategy 1 < DiMining containing pruning strategy 2 < DiMining without pruning'. The DiMining without pruning has the lowest mining efficiency due to repeated mining efforts, while the DiMining with pruning 3 has the second highest efficiency because pruning strategy 3 greatly simplifies the mining process of linear extension of differential frequent pattern mining. The mining efficiency of DiMining when the dataset increases to 400 (0.3236 s) is about twice that of DiMining without pruning (0.6742 s), and the advantage of DiMining becomes more obvious as the dataset increases, which indicates that the pruning strategy we designed greatly improves the mining efficiency of the algorithm.

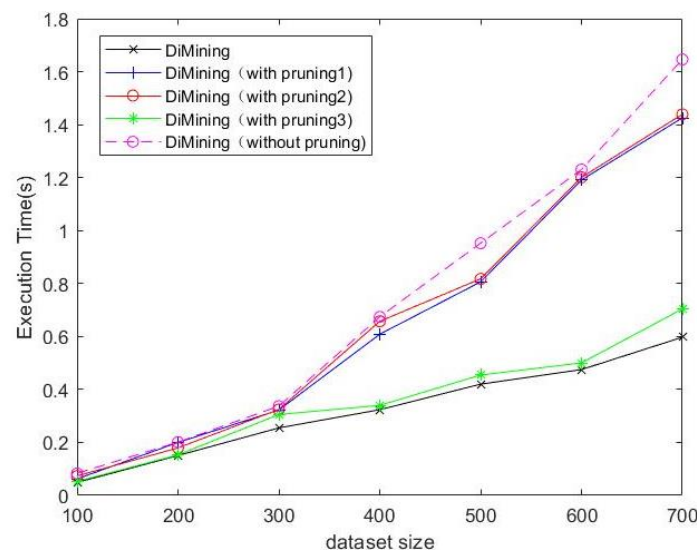


Figure 12. Comparison of pruning strategies.

(2) Efficiency comparison

The DiMining algorithm is compared with the SPM algorithm, the SPM-D algorithm and the SARA algorithm. The SPM algorithm finds frequent action sequences, and associates important action sequences together by association rules, and the SPM-D algorithm combines the SPM algorithm with multidimensional pattern mining. Unlike the SPM algorithm and the SPM-D algorithm which mine moment data, the DiMining algorithm mines the duration dataset and can mine continuous patterns of events. The SARA algorithm converts moment dataset into duration dataset and mines event patterns that show the relationship and time period between each event without any candidate generation. The SARA algorithm can only mine frequent temporal patterns, while the DiMining algorithm can mine differential frequent temporal patterns.

Figure 13 shows the execution times on the Kosarak dataset, with both the x-axes and y-axes represented on a logarithmic scale. We use three datasets of different sizes, arranged in ascending order of the number of sequences. The average item length of these four

datasets is 10, and the minimum support parameter is set to 3%. The execution time of all algorithms increases as the number of sequences increases. Specifically, DiMining takes less time compared to the other algorithms. The difference in efficiency between the algorithms is more pronounced when the size of the dataset is larger.

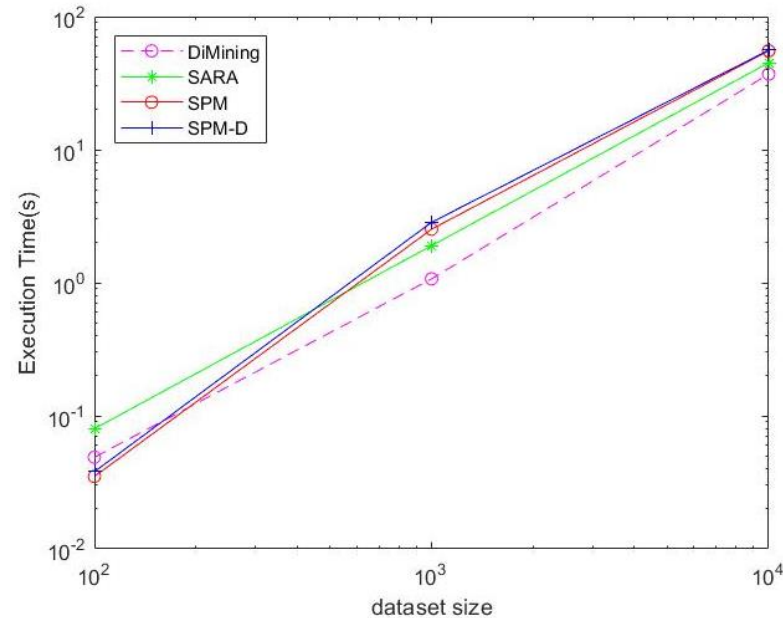


Figure 13. Comparison of efficiency of different algorithms.

5.2. Simulation Experiment

How to select the flight unit and mount for the task execution sequence before the task execution is a difficult decision-making point. The Avionics Cloud system can call the DiMining algorithm to analyze the past dataset to output an efficient task execution sequence pattern to assist in decision making after the target demand is known. As shown in Figure 14, the specific simulation scenario is as follows: UAV formation in a sea 10,000 square nautical miles within the sea stereo detection task, detection target for another UAV formation, and the experimental data were collected using a simulation scenario software named Origin, developed by Shareetech. In Figure 14, the blue side represents our reconnaissance aircraft and the red side represents the target to be detected. The efficiency is judged by the number of detected target clusters under the specified time, and the 700-field simulation is executed and the task execution results are collected by adjusting the presets such as the task sequence of UAVs in the simulation scenario. Four specific types of simulation scenarios are used: using electronic jamming to affect the efficiency of the opponent in detecting us; using UAVs with anti-radiation capabilities for detection; using UAVs to enter from the side for early detection; and focusing on using high-power radar for detection.

The typical tasks in the simulation scenario are selected for mining analysis in order to mine the better detection task assignment organization, corresponding to the table shown in Table 3, for example, the detection task of early warning aircraft radar 1 indicates the detection task performed by a certain radar sensor (number 1) of the early warning aircraft.

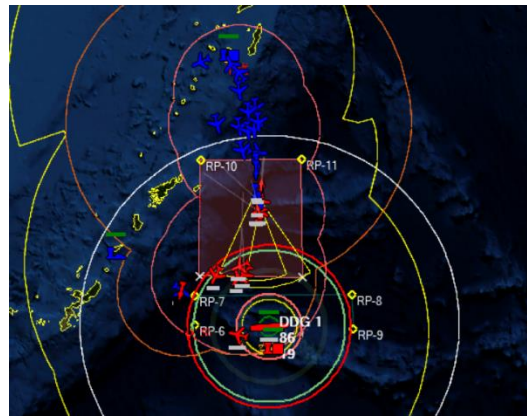


Figure 14. Task scenario model.

Table 3. Task correspondence table.

Task Number	Task Name
F1	UAV anti-radiation Task
F2	Early Warning Aircraft Radar 1 Detection Task
F3	Early Warning Aircraft Radar 2 Detection Task
F4	UAV Radar 1 Detection Task
F5	UAV Radar 2 Detection Task
F6	UAV Radar 3 Detection Task
F7	UAV Radar 4 Detection Task

The DiMining algorithm was used to mine the dataset of 350 high-efficiency running scenes and 350 low-efficiency running scenes, and the final results are shown in Table 4. Where $F1-F7$ denote the task numbers as shown in Table 3. f, m, o, c, a denote the relationship between events follows, meets, overlaps, contains, matches, respectively. Taking “ $F1 f F2 o F4$ ” as an example, the combination of tasks is indicated as follows: first, the UAV anti-radiation task is performed. After that, the AWACS radar 1 turns on to perform the detection task. During the detection task of the early warning aircraft radar 1, the UAV radar 1 is turned on for the detection task. A higher detection efficiency will be obtained under this task execution sequence. It is because detection time can be effectively shortened by ‘jamming the opponent’s sensors by performing anti-radiation tasks first, and switching on the early warning aircraft radar with a larger detection range first, and then switching on the UAV radar after the early warning aircraft has reduced its detection range’.

Table 4. Part of Mining results.

Number	Result
1	F1 f F2 o F4
2	F2 f F2 o F4
3	F4 f F5 o F5
4	F4 f F5
5	F4 m F7
6	F4 a F2 o F4
7	F5 c F5

The optimized simulation model is obtained by adjusting the task configuration in the simulation scenario according to the mining results. In this step, the task resource allocation algorithm (DiMining algorithm) is used in the simulated Avionics Cloud to optimize the execution task sequence of the existing scene. The optimized model is simulated 100 times to get the optimized data. The number of detected target clusters is normalized to compare

the data before and after the optimization, when the number of detections is larger, the higher the indicator value, the indicator value range is 0 to 100, set the detection time for 2 h, when no target clusters are detected the indicator value is 0, when all the target clusters are detected the indicator value is 100. The indicators' results obtained before and after simulation are sorted in descending order, which is convenient to see the difference before and after optimization, as shown in Figure 15. The optimized result is more reasonable in the task execution sequence, so the detection range of confrontation can be significantly increased. The purpose of this experiment is to simulate the process of the DiMining algorithm in Avionics Cloud assisted decision making, and verify the effectiveness of the algorithm by comparing the efficiency before and after the application of the algorithm. It can be seen that using the DiMining algorithm to assist in Avionics Cloud decision-making can significantly improve the efficiency of the cluster co-detection function. The DiMining algorithm is used to deep mine the generated high/low-efficiency task dataset to identify key task combination patterns related to improving detection efficiency, to adjust the scheduling rules of the task resource pool in detection and achieve intelligent resource scheduling.

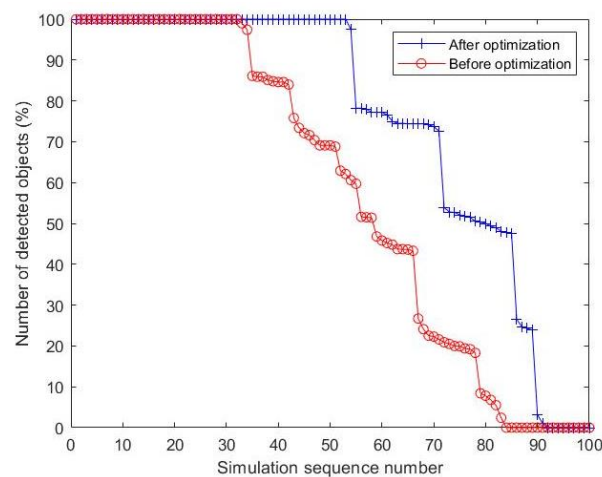


Figure 15. Performance comparison before and after optimization.

6. Conclusions

In this paper, we have studied a research method for UAV cluster task assignment in Avionics Cloud architecture based on the DiMining algorithm. We propose a new algorithm, the DiMining algorithm, which utilizes a memory efficient data structure and a new chain table pruning method for analyzing efficient task matching patterns between units in the task scenario dataset. The DiMining algorithm mines the differential dataset, uses a relational pair storage data structure to store the raw data, and uses multiple pruning strategies to mine the differential frequent time interval patterns to ensure that the two differential time interval patterns with high and low execution efficiency are mined at one time, avoiding the loss caused by secondary mining and improving the mining efficiency. The efficiency of the proposed algorithm and pruning method is verified by comparison to other algorithms.

Then, we used simulation software to simulate the UAV and unmanned ship collaborative detection scenarios, and analyzed and mined the data obtained from the simulation of 350 high-efficiency and 350 low-efficiency operation scenarios. The generated high-efficiency dataset and the low-efficiency dataset are mined by the DiMining algorithm to identify the key task execution patterns related to efficiency. According to the mining results, the task execution sequence is adjusted, and the scene is simulated again to realize the function of the resource allocation algorithm in the Avionics Cloud. By comparing the efficiency of two simulations, the effectiveness of the DiMining algorithm in Avionics Cloud decision-making is verified. In conclusion, we can say that the proposed DiMining

algorithm is a promising method for aided decision-making of task execution sequences under Avionics Clouds.

Author Contributions: Conceptualization, M.W. and G.W.; methodology, L.P.; software, X.D.; validation, X.D.; formal analysis, X.D.; investigation, X.D.; resources, X.D.; data curation, X.W.; writing—original draft preparation, X.D.; writing—review and editing, M.W.; visualization, X.D.; supervision, M.W.; project administration, X.W.; funding acquisition, M.W. All authors have read and agreed to the published version of the manuscript.

Funding: This study was funded by Natural Science Foundation of Shanghai (20ZR1427800), New 945 Young Teachers Launch Program of Shanghai Jiao Tong University (20X100040036).

Data Availability Statement: “Clickstream dataset Kosarak” at <https://www.philippe-fournier-viger.com/spmf/index.php>, accessed on 1 December 2022.

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

References

1. Mark, H.; George, V.; Gregory, B.P. A comparison of open architecture standards for the development of complex military systems: GRA, FACE, SCA NeXT (4.0). In Proceedings of the MILCOM 2012–2012 IEEE Military Communications Conference, Orlando, FL, USA, 29 October–1 November 2012.
2. Guertin, N. How the navy is using open systems architecture to revolutionize capability acquisition. In Proceedings of the 12th Annual Acquisition Research Symposium, Monterey, CA, USA, 13–14 May 2015.
3. Du, X.; Du, C.; Chen, J.; Liu, Y. An energy-aware resource allocation method for avionics systems based on improved ant colony optimization algorithm. *Comput. Electr. Eng.* **2023**, *105*, 108515. [\[CrossRef\]](#)
4. Elkholy, W.; El-Menshawy, M.; Bentahar, J.; Elqortobi, M.; Laarej, A.; Dssouli, R. Model checking intelligent avionics systems for test cases generation using multi-agent systems. *Expert Syst. Appl.* **2020**, *156*, 113458. [\[CrossRef\]](#)
5. Kabashkin, I.; Filippov, V. Reliability of Software Applications in Integrated Modular Avionics. *Transp. Res. Procedia* **2020**, *51*, 75–81. [\[CrossRef\]](#)
6. The Open Group. Technical Standard for Future Airborne Capability Environment (FACETM). *Edition 2.1*. Available online: <http://www.opengroup.org/bookstore> (accessed on 20 November 2019).
7. Turiak, M.; Sedláčková, A.N.; Novak, A. Portable electronic devices on board of airplanes and their safety impact. In *Telematics-Support for Transport, Proceedings of the 14th International Conference on Transport Systems Telematics, TST 2014, Katowice/Kraków/Ustroń, Poland, 22–25 October 2014*; Springer: Berlin/Heidelberg, Germany, 2014. [\[CrossRef\]](#)
8. Yang, H.; Sun, Y. A combination method for integrated modular avionics safety analysis. *Aircr. Eng. Aerosp. Technol.* **2022**, *95*, 345–357. [\[CrossRef\]](#)
9. Dong, X.; Wang, M.; Liu, Y.; Xiao, G.; Huang, D.; Wang, G. An Efficient Spatial High Utility Occupancy Frequent Item Mining Algorithm for Task System Integration Architecture Design using MBSE Method. *Aerosp. Syst.* **2022**, *5*, 377–392. [\[CrossRef\]](#)
10. Li, Z.; Li, Q.; Xiong, H. Avionics clouds: A generic scheme for future avionics systems. In Proceedings of the 2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC), Williamsburg, VA, USA, 14–18 October 2012; pp. 6E4-1–6E4-10. [\[CrossRef\]](#)
11. Nguyen, A.-Q.; Amrhar, A.; Landry, R. Direct RF Sampling Avionics Architecture for future multi-system integrated Avionics. In Proceedings of the 2018 16th IEEE International New Circuits and Systems Conference (NEWCAS), Montreal, QC, Canada, 24–27 June 2018; pp. 61–65. [\[CrossRef\]](#)
12. Jianchun, X.; Zhonghua, W.; Yahui, L. The Distributed Computing Framework Research for Avionics Cloud. In Proceedings of the 2018 International Conference on Networking and Network Applications (NaNA), Xi'an, China, 12–15 October 2018; pp. 390–393. [\[CrossRef\]](#)
13. Ma, X.; Jiang, C.; Huang, C. Cloud Task Optimization Scheduling based on Three-branch clustering. *Comput. Sci.* **2022**, *49*, 875–881.
14. Ye, F.; Shen, W. Task scheduling algorithm based on depth of reinforcement learning to improve. *J. Comput. Age* **2022**, *11*, 58+55–64.
15. Wang, S.; Sun, J.; Wang, P.; Yang, A. Cloud and synergy of resource scheduling optimization. *J. Telecom Sci.* **2022**, *12*, 1–9.
16. Fournier-Viger, P.; Nkambou, R.; Mephu Nguifo, E. A Knowledge Discovery Framework for Learning Task Models from User Interactions in Intelligent Tutoring Systems. In Proceedings of the 7th Mexican International Conference on Artificial Intelligence (MICAI 2008), Atizapan de Zaragoza, Mexico, 27–31 October 2008; Springer: Berlin/Heidelberg, Germany, 2008; pp. 765–778.
17. Huang, J.-W.; Jaysawal, B.P.; Chen, K.-Y.; Wu, Y.-B. Mining frequent and top-K High Utility Time Interval-based Events with Duration patterns. *Knowl. Inf. Syst.* **2019**, *61*, 1331–1359. [\[CrossRef\]](#)
18. Ritika; Gupta, S. K. HUFTI-SPM: High-utility and frequent time-interval sequential pattern mining from transactional databases. *Int. J. Data Sci. Anal.* **2022**, *13*, 239–250. [\[CrossRef\]](#)
19. Mordvanyuk, N.; Lopez, B.; Bifet, A. vertTIRP: Robust and efficient vertical frequent time interval-related pattern mining. *Expert Syst. Appl.* **2021**, *168*, 114276. [\[CrossRef\]](#)

20. Lee, Z.; Lindgren, T.; Papapetrou, P. Z-Miner: An Efficient Method for Mining Frequent Arrangements of Event Intervals. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20), New York, NY, USA, 6–10 July 2020; pp. 524–534.
21. Fang, G.; Kuang, R.; Pandey, G.; Steinbach, M.; Myers, C.L.; Kumar, V. Subspace Differential Co-expression Analysis: Problem Definition and A General Approach. In Proceedings of the 15th Pacific Symposium on Biocomputing (PSB), Kamuela, HI, USA, 4–8 January 2010; Volume 15, pp. 145–156.

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.