Flight Planning at Scale: A Bipartite Matching based Approach

Tianlong Zhang† Chang Gao† Yuxiang Zeng† Shuyuan Li† Yi Xu‡
State Key Laboratory of Software Development Environment,
Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing,
School of Computer Science and Engineering and Institute of Artificial Intelligence, Beihang University, Beijing, China
{tianlong, gc021129, yxzeng, lishuyuan, xuy}@buaa.edu.cn

ABSTRACT

Flight planning, a pivotal challenge in the airline industry, strives to achieve economic and flexible scheduling of airplanes to serve designated flight itineraries. As the demand for air transportation soars, traditional planning methods can be inefficient in managing large-scale flights. Thus, we introduce Swift, a data-driven system tailored to enhance the scalability and effectiveness of flight planning. Swift primarily employs the bipartite graph model to derive optimal and economic flight plans for airlines. Our method not only minimizes the number of required planes but also ensures a balanced workload across these planes. Furthermore, Swift offers the capability of dynamic updates to flight plans in response to unexpected incidents at airports, such as bad weather conditions. Besides, Swift incorporates other functionalities like predicting future flight demand and monitoring real-time flight trajectories. Conference participants can interact with this system and explore our flight planning solution in real-world scenarios.

PVLDB Reference Format:
Tianlong Zhang, Chang Gao, Yuxiang Zeng, Shuyuan Li and Yi Xu. Flight Planning at Scale: A Bipartite Matching based Approach. PVLDB, 17(1): XXX-XXX, 2024.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Air transportation has emerged as an indispensable pillar of global connectivity in today's world. The International Air Transport Association (IATA) predicts that passenger volumes will reach 4 billion in 2024 [8]. As the demand of this service continues to grow, the complexity and scalability of arranging proper airplanes for flights are also expanding. Thus, it is crucial and urgent to investigate how to make an optimal plan for large-scale flights (*i.e.*, flight planning).

Flight planning refers to the process of deciding proper arrangements for multiple airplanes to collaboratively serve all flights in an airline company, taking into account various factors such as the spatiotemporal information for departures and arrivals. Beyond the pre-defined spatiotemporal constraints, the optimality of the flight plan typically depends on how effectively it aligns with the airline company's primary objectives. Specifically, two objectives are commonly considered in real-world airline companies (*e.g.*, *Sichuan Airlines* [13]): *minimizing the number of required airplanes* and *minimizing the workload balance of these airplanes*. The former can help save on the expenses of purchasing/leasing airplanes, and the latter can help reduce their maintenance fees. Thus, good solutions are desired to solve this multi-objective *flight planning* problem.

Motivation and Challenge. Existing studies on flight planning are primarily limited to small-scale data. For instance, integer programming based method and heuristic algorithms [3, 4] have been proposed to plan for a few hundred flights, falling short in real-world scenarios involving thousands of flights. By contrast, large-scale trip planning has attracted research attention in the database community. Efficient planning algorithms are devised and tailored for different applications, such as ride sharing [3, 6, 21], urban logistics [15, 22], and travel route recommendation [9, 14, 17, 24]. However, they usually consider a single optimization goal that is different from ours, *e.g.*, minimizing the total travel distance and maximizing the platform's total revenue. Thus, previous solutions are not suitable for this problem due to the following *technical challenges*: (1) multiple optimization objectives need to be simultaneously optimized and (2) large-scale flights need to be promptly processed.

Our Main Idea. To address these technical challenges, we are motivated to propose an effective and scalable solution to the Flight Planning (FP) problem. Specifically, we first reduce the problem into a bipartite graph model by splitting each flight into a departure vertex (*i.e.*, right-hand vertex) and an arrival vertex (*i.e.*, left-hand vertex), and establishing the edge connectivity between any two vertices from different sides that satisfy the spatiotemporal constraints. Then, we leverage existing maximum cardinality bipartite matching algorithm to find an initial plan to minimize the number of required airplanes. After that, we enhance the workload balance of these airplanes by swapping partial flights from two airplanes' current plans. Moreover, we also design optimizations to reduce the cost of the bipartite graph construction.

Contribution. The main contributions of our paper are summarized as follows:

- To the best of our knowledge, we are the first to formulate and study the Flight Planning (FP) problem over large-scale dataset with multi-objectives.
- We propose a bipartite matching based framework to solve the problem, and devise novel optimizations to enhance the efficiency and effectiveness.
- We formally prove our optimized algorithm not only minimizes the number of airplanes, but also guarantees a *Pure Nash Equilibrium* [11] in balancing the flight workload of these airplanes.

Proceedings of the VLDB Endowment, Vol. 17, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

- Extensive experiments are conducted on two real datasets to demonstrate the exceptional performance of our solution. Specifically, our method can lead to a reduction of up to 47% in the number of required airplanes, while also improving the workload balance by up to 89%. Moreover, the evaluation also verifies that our optimizations can reduce memory usage by up to 3 orders of magnitude and running time by up to 2 orders of magnitude.

RoadMap. In the rest of this paper, we first define the Flight Planning (FP) problem in Sec. 2. Next, we introduce our solution and present theoretical analysis in Sec. 3. Then, we conduct extensive experiments in Sec. 4. Finally, we review related studies in Sec. 5, and conclude in Sec. 6.

2 PROBLEM STATEMENT

This section introduces the basic concepts and formal definition of our **Flight Planning (FP) problem**. We first define the concept of a *flight* as follows.

Definition 1 (Flight)**.** A flight f is denoted by: $f = \langle DS_f, DT_f, AS_f, AT_f, Type_f \rangle$, where the departure airport is DS_f, the departure time from the airport DS_f is DT_f, the arrival airport is AS_f, and the arrival time at the airport AS_f is AT_f. Each flight f is assigned a specific type of airplane $Type_f$.

In real life, each flight usually has different departure and arrival airports. We use the set F to denote a collection of n flights, *i.e.*, $F = \{f_1, \dots f_n\}$.

Definition 2 (Airplane and Flight Plan)**.** An airplane, denoted by $c = \langle Type_c, P_c \rangle$, where $Type_c$ is the airplane type, and P_c is the set of flight assigned to it. We use $C = \{c_1, \dots, c_m\}$ to denote a fleet of airplanes that have the same type. For this type of airplane, the flight plan is denoted by $P = \{P_{c_1}, \dots, P_{c_m}\}$, where each $P_{c_i} = \{f_1, \dots, f_k\}$ is an ordered sequence of flights for airplane c_i. A **feasible flight plan** should satisfy the following constraints:

- **Flight Airport Constraint.** The flight airport constraint requires that the arrival airport of a previous flight is the same as the departure airport of the next flight, i.e.,

$$\forall i \in [1, k), \quad AS_{f_i} = DS_{f_{i+1}} \tag{1}$$

- **Flight Time Constraint.** The flight time constraint requires that the time interval between two consecutive flights in the flight plan should be no shorter than the minimum waiting time $T_{\min}$ set by the airlines, i.e.,

$$\forall i \in [1, k), \quad DT_{f_{i+1}} - AT_{f_i} \geq T_{\min} \tag{2}$$

- **Flight Type Constraint.** The flight type constraint indicates that each flight type $Type_f$ must match with the airplane type $Type_c$, i.e.,

$$\forall i \in [1, k], \quad Type_{f_i} = Type_c \tag{3}$$

The flight airport constraint and flight time constraint are spatial and temporal constraints of this problem. In the flight time constraint, the parameter $T_{\min}$ is determined by airline companies, since an airplane needs enough time for maintenance after one flight trip. As for the flight type constraint, it generally aligns with the real-world situation, where different flights prefer different types of airplanes depending on several factors, such as passenger capacity and flight speed. Based on these concepts, our problem definition is as follows.

Definition 3 (Flight Planning Problem)**.** Given a set of flights F, the Flight Planning (FP) problem aims to make proper plans for each flight $f \in F$ to optimize the following two objectives:

- **Minimize the number of required airplanes**. Due to the high cost of airplane leasing or procurement, airlines usually aim to use the minimum number of airplanes to cover all the flights [1].
- **Minimize the airplanes' workload balance score.** Moreover, it is crucial for airline companies to effectively balance the workloads of their airplanes to minimize maintenance costs [7].

$$\text{Workload Balance Score} = \frac{1}{|C|} \sum_{c \in C} (|P_c| - \frac{|F|}{|C|})^2 \tag{4}$$

Here, $|F|$ represents the total number of flights, and $|P_c|$ represents the number of flights served by the airplane c. Accordingly, Eq. (4) measures the balance of the number of flights per airplane over a period of time.

Moreover, the flight plan P should satisfy the following constraints.

- Each flight must be served by one airplane.
- Each flight plan must be feasible (see Definition 2).

In real life, the cost of purchasing and operating an airplane is a major expense for airlines. Therefore, the crux of flight planning is to minimize the number of airplanes to curb these costs. Moreover, achieving a balanced workload in flight planning is also crucial. Thus, we utilize the variance formula in Eq. (4) to measure workload balance. A higher variance indicates a less balanced workload plan, while a lower variance suggests a more balanced workload plan.

We also use the following example to illustrate our FP problem.

Example 1. Consider a set of flights $F = \{f_1, f_2, f_3, f_4, f_5\}$ from Table 1, which are depicted in Fig. 1. Each flight is represented by a tuple, for example, $f_1 = \langle Shenzhen, Chengdu, 7:25, 9:40, A319 \rangle$, indicating its departure from Shenzhen at 7 : 25 and arrival at Chengdu at 9 : 40. We have set the minimum waiting time $T_{\min}$ to be 30 minutes. Another flight $f_2 = \langle Chengdu, Beijing, 10:45, 13:30, A319 \rangle$, which departs from the airport where flight f_1 is scheduled to land. The waiting time between these two flights exceeds $T_{\min}$. Therefore, flights f_1 and f_2 can be assigned to the same airplane for sequential operations.

Table 1: The set of flights F

Flight ID	DS	AS	DT	AT	$Type_f$
f_1	Shenzhen	Chengdu	7:25	9:40	A319
f_2	Chengdu	Beijing	10:45	13:30	A319
f_3	Beijing	Chengdu	14:20	17:15	A319
f_4	Chengdu	Shanghai	18:00	21:00	A319
f_5	Chengdu	Changchun	19:40	23:10	A319

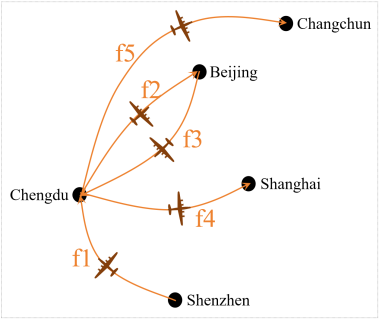

Figure 1: The set of flights F in Example 1

Table 2: Summary of major notations

Notation	Description
f	A flight
F	A set of flights
c, C	An airplane and a set of airplanes with the same type $Type_c$
P	The flight plan
P_c	The subset of flight plan in P for the airplane c
DS_f	The departure airport of the flight f
AS_f	The arrival airport of the flight schedule$f$
DT_f	The departure time of the flight f
AT_f	The arrival time of the flight f
$T_{\min}$	The minimum waiting time

3 OUR BIPARTITE MATCHING BASED SOLUTION

In this section, we first introduce our bipartite matching based framework in Sec. 3.1. Under this framework, we design an effective and efficient algorithm in Sec. 3.2. Finally, we present theoretical analysis of our solution in Sec. 3.3. Table 2 summarizes the commonly-used notations in the rest of this paper.

3.1 Bipartite Matching based General Framework

Main Idea: Reduction to Bipartite Matching. In Definition 1, each flight contains spatiotemporal information about the departure and arrival. Thus, we can use a right-hand vertex in a bipartite graph to represent the departure information of a flight, and its corresponding left-hand vertex to represent this flight's arrival information. Moreover, an edge will be connected if an airplane is capable of initially accommodating the flight represented by the left-hand vertex, followed by servicing the flight denoted by the right-hand vertex. As a result, the first objective, minimizing the number of required airplanes, is reduced to the minimum path cover problem on this bipartite graph, which can be solved by maximum (cardinality) bipartite matching [20].

Basic Framework. In Algo. 1, lines 2-6 construct a bipartite graph to model the relationships between the set of flights. In line 6, an edge will be established between two flights only if these two flights can be continuously served by one airplane under the flight time constraint and airport constraint. In line 7, a maximum cardinality bipartite matching M is obtained by using the Hungarian algorithm [5]. In line 8, we derive the flight plan P by considering the flights assigned to the same airplane, using the connecting edges within M.

Algorithm 1: Our bipartite matching based framework HA-FP

input : A set F of flights, Minimum waiting time $T_{\min}$
output: Flight plan P

1 $G, M, C, P \leftarrow \emptyset$;
2 **foreach** $f_i \in F$ **do**
3 **foreach** $f_j \in F$ **do**
4 $G[f_i][f_j] \leftarrow 0$;
5 **if** $AS_{f_i} = DS_{f_j}$ *and* $DT_{f_j} - AT_{f_i} \geq T_{min}$ **then**
6 $G[f_i][f_j] \leftarrow 1$;
7 $M \leftarrow$ maximum cardinality bipartite matching of G;
8 Transform matching result M into a flight plan P;
9 **return** Flight plan P;

Example 2. Back to Example 1. We use the HA-FP framework (Algo. 1) for flight planning. Firstly, we construct a bipartite graph G: For each flight node, we create an edge in the bipartite graph $G[f_i][f_j]$ if f_i and f_j satisfy the constraints. The resulting bipartite graph G is depicted on the left side of Fig. 2. Subsequently, we apply the Hungarian algorithm to find a maximum matching $M = \{(f_1, f_2'), (f_2, f_3'), (f_3, f_4')\}$ in the bipartite graph G. Next, we assign the same airplane to each matched pair of flights based on the maximum matching M. The results show that a minimum of 2 airplanes are needed, with the flight plans for these two airplanes being $P_{c_1} = \{f_1, f_2, f_3, f_4\}$ and $P_{c_2} = \{f_5\}$. This flight plan P is shown on the right side of Fig. 2. Finally, the workload balance score for the flight plan $P = \{\{f_1, f_2, f_3, f_4\}, \{f_5\}\}$ is calculated as: $((4 - \frac{5}{2})^2 + (1 - \frac{5}{2})^2)/2 = 2.25$.

Time and Space Complexity. In Algo. 1, constructing the bipartite graph takes $O(n^2)$ time, and finding the maximum cardinality bipartite matching takes $O(n^3)$ time. Thus, the time complexity of Algo. 1 is $O(n^3)$. Since the bipartite graph takes $O(n^2)$ main memory space, the space complexity of Algo. 1 is $O(n^2)$.

3.2 Optimizing Efficiency and Multi-Objective Effectiveness

Based on the above complexity analysis, a naive implementation of our general framework suffers from high complexity. Moreover, it does not perform well on the second objective (see Sec. 3.1). To tackle these limitations, we propose two optimization operations to reduce the complexity and improve the workload balance while retaining the optimality in the first objective.

Main Idea of Optimizing Bipartite Graph Construction. To construct the bipartite graph G, Algo. 1 first enumerates all pairwise vertices from scratch, then checks the constraints, and finally decides the edge connectivity. Our *basic idea* is to first filter out all right-hand vertices that (1) are unmatched with the flight type and arrival airport of the left-hand vertex or (2) violate the flight

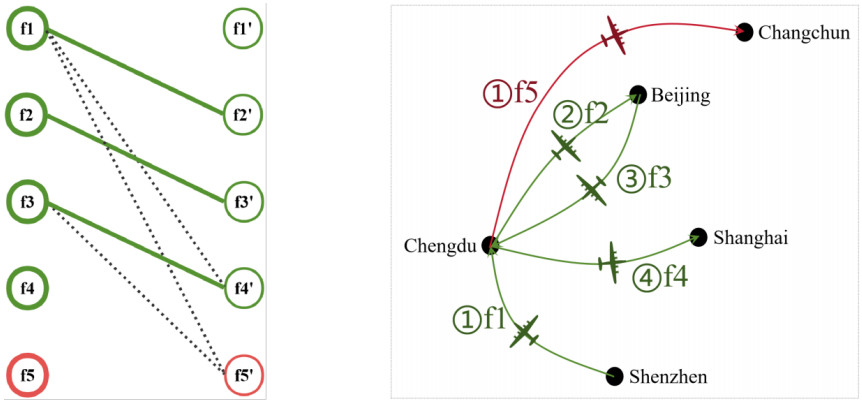

Figure 2: The flight planning results based on the HA-FP algorithm.

time constraint. Specifically, the first condition can be efficiently verified by a *hash* function with the key of flight type and departure airport, and the second condition can be by a *binary search* over the departure time. Moreover, we will also introduce how to reduce the space cost into linear complexity in Algo. 2.

Main Idea of Improving Optimization Objectives. Algo. 1 only considers the first objective. To further minimize the second objective while retaining the optimality of the first objective, a new operation called "*swap*" is proposed in Definition 4 to perform local search on the basis of the current plan.

Definition 4 (Swap). The swap operations exchanges some consecutive flights between $P_{c_i} = \{f_{i,1}, f_{i,2}, \ldots, f_{i,k_i}, \ldots, f_{i,m_i}\}$ and $P_{c_j} = \{f_{j,1}, f_{j,2}, \ldots, f_{j,k_j}, \ldots, f_{j,m_j}\}$, ensuring that the resulting plans P'_{c_i} and P'_{c_j} remain feasible, i.e.,

$$P'_{c_i} = \{f_{i,1}, f_{i,2}, \ldots, f_{j,k_j}, \ldots, f_{j,m_j}\},\ P'_{c_j} = \{f_{j,1}, f_{j,2}, \ldots, f_{i,k_i}, \ldots, f_{i,m_i}\} \tag{5}$$

We denote the difference in the number of flight sequences swapped between c_i and c_j as Δ. To ensure that each swap operation reduces the workload balance score, we require the swap operation to be conducted under the following conditions:

$$|P_{c_i}| - |P_{c_j}| > \Delta \text{ , where } \Delta = |(m_i - k_i) - (m_j - k_j)| \tag{6}$$

We will keep doing swap until the workloads between any two flights reach a balanced state. In Sec. 3.3, we will explain the rationale of the swap, proving that such a state is a *Pure Nash Equilibrium* [11] and proving the validity of the Eq. (6).

Optimized Algorithm Details. Algo. 2 describes the optimized algorithm GT-FP. In line 1, the algorithm models the set of flights F as a hash table H with an index for $Type$ and DS_{f_k}. In lines 2-3, for each flight f_i, the algorithm employs a binary search using the hash table to find the earliest flight meeting the flight time constraints, and subsequently appends it to the hash table. In line 4, the algorithm obtains a maximum matching using the compressed G to minimize the required number of airplanes. In lines 6-12, the algorithm uses swap operations to improve the second objective of minimizing workload balance score. In line 10, the difference in the number of swapped flights between c_i and c_j is calculated, and in line 11, the condition for swap in Equation 6 is verified. The iteration stops when no further improvement in workload balance score can be achieved through swap operations between any two airplanes. Finally, the flight plan P will be returned.

Algorithm 2: Our optimized solution GT-FP

input : A set F of flights, Minimum waiting time $T_{\min}$
output: Flight plan P

`// Represent the bipartite graph` G `by a hash table` H

1 Hash $H[(Type, DS_{f_k})] \leftarrow$ sort $\{(DT_{f_k}, f_k) \mid f_k \in F \wedge Type_{f_k} = Type\}$ in ascending order;
2 **foreach** $f_i \in F$ **do**
3 $\quad H' \leftarrow H[(Type_{f_i}, AS_{f_i})]$, $H'[j] \leftarrow$ BinarySearch $(AT_{f_i} + T_{\min})$ from H';
$\quad$ `//` $\forall k = H'[j].k, H'[j+1].k, \cdots, H'[|H'|].k$, `there is an edge between left-hand vertex of` f_i `and right-hand vertex of` f_k `in` G
4 $M \leftarrow$ maximum cardinality bipartite matching of compressed G via H;
5 Transform matching result M into a flight plan P;
`// Improve the second objective of minimizing workload balance score`
6 balanced $\leftarrow$ *false*;
7 **while** *balanced is false* **do**
8 $\quad$ balanced $\leftarrow$ *true*;
9 $\quad$ **foreach** $P_{c_i}, P_{c_j} \in P$ *with the same airplane type* **do**
10 $\quad\quad \Delta \leftarrow$ compute according to Eq. (6);
11 $\quad\quad$ **if** $|P_{c_i}| - |P_{c_j}| > \Delta$ **then**
12 $\quad\quad\quad$ Perform a swapping between P_{c_i} and P_{c_j}, balanced $\leftarrow$ *false*;
13 **return** *Flight plan* P;

Example 3. Back to Example 1. We use the GT-FP (Algo. 2) to reperform flight planning. First, we present the edges of the bipartite graph G by a linear list L. Then, we determine an arbitrary maximum matching M within L and transform it into a flight plan $P = \{\{f1, f2, f3, f4\}, \{f5\}\}$. The workload balance score of P is 2.25. Next, we improve the second objective of minimizing workload balance score. Specifically, we swap the subsequences $\{f_2, f_3, f_4\}$ from P_{c_1} with $\{f5\}$ from P_{c_2}, resulting in $\Delta = 2$. Because $|P_{c_1}| - |P_{c_2}| > \Delta$, it satisfies the condition for the swap operation. After executing the operation, the new flight plans are $P'_{c_1} = \{f_1, f_5\}$ and $P'_{c_2} = \{f_2, f_3, f_4\}$. The final results of flight planning using the GT-FP are shown in Fig. 3. This revised planning yields a workload balance score of 0.25, representing an 89% reduction in the workload balance score than initial planning.

Time and Space Complexity. Let c denote the maximum degree of the bipartite graph G. In Algo. 1, lines 1-3 take $O(n \log n)$ time due to the time cost of the binary search. Lines 4-5 take $O(cn^2)$ time. The iterations in lines 7-12 take $O(rn^2)$, where r denotes the number of rounds to reach a balanced state. The proof of *Nash equilibrium* [11] in the next subsection implies that the iteration round r is finite. Therefore, the overall time complexity is reduced from $O(n^3)$ to $O(cn^2)$. By utilizing hash and compressed bipartite graph in Algo. 2, the space complexity is reduced from $O(n^2)$ to $O(n)$ due to their linear space requirement.

3.3 Theoretical Analysis from A Game Theory Perspective

This subsection presents our analysis on the theoretical guarantees of Algo. 2.

Preliminary. We first introduce preliminary knowledge on game theory in our theoretical analysis. Specifically, the algorithm procedure in Algo. 2 can be modelled via game theory [11] as follows. Each airplane c_i behaves as a strategic player with the objective of minimizing its workload balance score (*a.k.a.*, utility function U_i) by selecting the optimal flight plan P_{c_i}, thereby leading to a reduced overall utility score. The set S_i represents all possible operations of swap operations that the player c_i can choose. A joint operation S of all players corresponds to the final flight plan P for the FP problem.

Formally speaking, a strategic game $\mathcal{G} = \langle C, \{S_i\}, \{U_i\}\rangle$ is called a *potential game* [11] if and only if there is potential function Φ that satisfies:

$$\Phi(s_i, s_{-i}) - \Phi(s'_i, s_{-i}) = U_i(s_i, s_{-i}) - U_i(s'_i, s_{-i}),\ \forall s_i, {s_i}' \in S_i \quad (7)$$

Here, C represents the set of airplanes participating in the game. s_i and ${s_i}'$ represent the swap operations between airplanes c_i and c_j. s_{-i} is the joint operation of the other airplanes except for c_i and c_j.

The essential goal of a potential game is to find the *Pure Nash Equilibrium (PNE)* [11]. For example, in our FP problem, a flight plan with PNE indicates that the overall workload balance score cannot be decreased by swapping between any two airplanes. This is also what our second objective asks for. Thus, the total utility function $U(\cdot)$ is denoted as the workload balance score defined in Eq. (4).

Main Result. The main theoretical result is summarized in Theorem 1.

Theorem 1. Algo. 2 can not only minimize the number of required airplanes, but also achieve a *Pure Nash Equilibrium* in minimizing the workload balance score.

To prove Theorem 1, we first present Lemma 1 to demonstrate the optimality in minimizing the number of required airplanes. Next, we present Lemma 2 to indicate that this is a potential game, and Lemma 3 to prove that the swap condition in line 11 of Algo. 2 strictly decreases the workload balance score. Based on these lemmas, our Algo. 2 can be viewed as a best-response solution [11] to the potential game. Since the best-response framework eventually achieves a *Pure Nash Equilibrium* [11], we complete the proof.

Lemma 1. Algo. 2 can minimize the number of required airplanes.

Proof. The FP problem aims to find the minimum number of required airplanes to serve all flights, which can be viewed as a minimum path cover problem. Based on Gallai's theorem [20], the minimum path cover here equals the number of vertices minus the maximum (cardinality) bipartite matching, *i.e.*, $n - |M|$, where n is the number of flights and M is the matching result. In line 4 of Algo. 2, we leverage the Hungarian algorithm [5] to find the optimal matching M. Since the cardinality of this matching is the maximum, the number of augment paths (*i.e.*, required airplanes) to cover all flights is the minimum. □

Next, we prove our FP problem can be modelled as a potential game.

Lemma 2. The strategic game of our FP problem is a potential game.

Proof. Let s_i and ${s_i}'$ denote any swap operation between airplanes c_i and c_j. In this context, the joint operation s_{-i} is applicable to all airplanes except c_i and c_j. The potential function is defined as the workload balance scores of c_i and c_j, we can derive that:

$$\begin{aligned}&\Phi(s_i, s_{-i}) - \Phi(s'_i, s_{-i})\\ &= \frac{(|P_{c_i}| - \Delta - \frac{|F|}{|C|})^2 + (|P_{c_j}| + \Delta - \frac{|F|}{|C|})^2}{|C|} - \frac{(|P_{c_i}| - \frac{|F|}{|C|})^2 + (|P_{c_j}| - \frac{|F|}{|C|})^2}{|C|}\end{aligned} \quad (8)$$

Moreover, based on the definition of the utility function, we have

$$\begin{aligned}U_i(s_i, s_{-i}) = &\frac{\left(|P_{c_i}| - \Delta - \frac{|F|}{|C|}\right)^2}{|C|}\\ &+ \frac{\left(|P_{c_j}| + \Delta - \frac{|F|}{|C|}\right)^2}{|C|}\\ &+ \frac{\sum_{c \in C \setminus \{c_i, c_j\}} \left(|P_c| - \frac{|F|}{|C|}\right)^2}{|C|}\end{aligned} \quad (9)$$

$$\begin{aligned}U_i(s'_i, s_{-i}) = &\frac{\left(|P_{c_i}| - \frac{|F|}{|C|}\right)^2}{|C|}\\ &+ \frac{\left(|P_{c_j}| - \frac{|F|}{|C|}\right)^2}{|C|}\\ &+ \frac{\sum_{c \in C \setminus \{c_i, c_j\}} \left(|P_c| - \frac{|F|}{|C|}\right)^2}{|C|}\end{aligned} \quad (10)$$

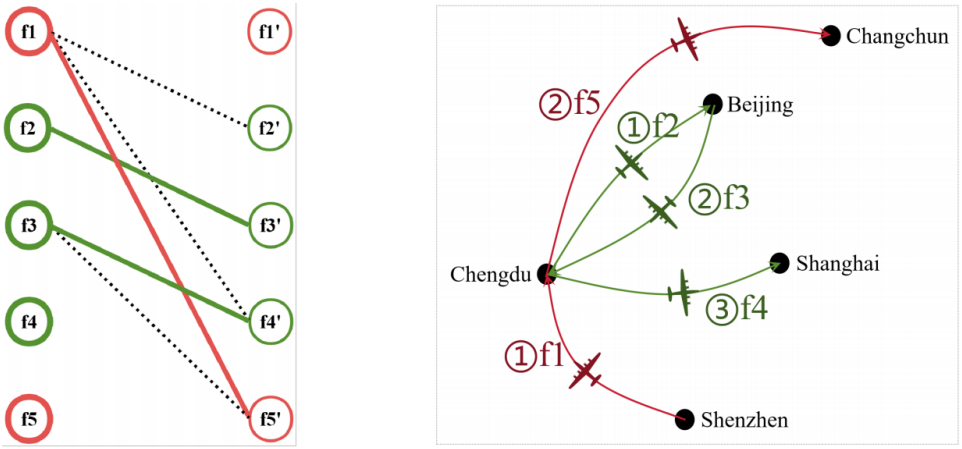

Figure 3: The flight planning results based on the GT-FP.

By subtracting Eq. (10) from Eq. (9), we can derive that

$$U_i(s_i, s_{-i}) - U_i(s_i', s_{-i}) = \Phi(s_i, s_{-i}) - \Phi(s_i', s_{-i}) \tag{11}$$

Eq. (11) satisfies the condition of a potential game, which proves this lemma. □

Finally, we prove our swap operation always decreases the total workload balance score, *i.e.*, $U_i(s_i, s_{-i}) - U_i(s_i', s_{-i}) < 0$ after a swap.

Lemma 3. The swap operation under the pre-condition $|P_{c_i}| - |P_{c_j}| > \Delta$ in line 11 of Algo. 2 can strictly decrease the utility function $U_i(\cdot)$.

Proof. Assume two airplanes, c_i and c_j, have executed a swap operation. Then, based on Eq. (8), the change of the utility function $U_i(\cdot)$ can be simplified as:

$$\frac{\Delta^2 - 2\Delta|P_{c_i}| + \Delta^2 + 2\Delta|P_{c_j}|}{|C|} = \frac{2\Delta(\Delta + |P_{c_j}| - |P_{c_i}|)}{|C|} \tag{12}$$

Because $|P_{c_i}| - |P_{c_j}| > \Delta$, we can infer $\Delta + |P_{c_j}| - |P_{c_i}| < 0$. Besides, since $|C| > 0$ denotes the number of airplanes and $\Delta > 0$, Eq. (12) is strictly lower than 0. Thus, the total workload balance score will be decreased after a swap. □

Based on the property of the potential game [11], Lemma 2 and 3 can deduce that Algo. 2 will ultimately reach a *Pure Nash Equilibrium* within finite swaps.

4 EXPERIMENTAL EVALUATION

This section introduces our evaluation setup and experimental results.

4.1 Experimental Setup

Datasets. Our evaluation is conducted on two real-world datasets:

- **National Flight Dataset.** We crawled 98,158 flight records from *WebXml* [19] as the large-scale dataset. This dataset covers 52 different airplane types from 29 Chinese airlines. We have open-sourced this dataset on an anonymous GitHub [10].
- ***Sichuan Airlines* Dataset.** This is a private dataset with 3636 pieces of domestic flight information collected by *Sichuan Airlines* [13] in 2020.

Table 3: Parameter settings

Parameter	Setting
Minimum waiting time $T_{\min}$	15, **30**, 45, 60 (minutes)
Data scale n ($\times 10^3$)	0.1, 1, 10, **98**
Days of records day	3, 4, 5, 6, **7**
Airplane type $Type_c$	A319, A320, A321, B737, B738, JET

Parameter Settings. The parameter settings of the *national flight dataset* are listed in Table 3, where the default settings are marked in bold. Specifically, the minimum waiting time $T_{\min}$ is varied from 15 minutes to 60 minutes. To test the impact of input size, we vary the number of flights from 100 to 98k (*i.e.*, all the records of our large-scale dataset). Besides, we also test different days of flight records. To show the results of different airplane types, we select 6 of the most common airplane types in this dataset, *i.e.*, A319, A320, A321, B737, B738, and JET. As for the *Sichuan Airlines* Dataset, we deploy our solution in the evaluation environment provided by *Sichuan Airlines* [13] to make a real-life performance comparison, and hence use their default settings.

Compared Algorithms. Four algorithms are compared in our evaluation: GREEDY, HA-FP, RSA-FP, and GT-FP. Among these algorithms, GREEDY is an industrial baseline algorithm provided by *Sichuan Airlines* [13], and its basic idea is to pair each flight node with its closest neighboring flight node based on time. HA-FP and RSA-FP are implementations of our general framework in Algo. 1, where HA-FP is a straightforward implementation and RSA-FP leverages a randomization in searching the maximum cardinality bipartite matching. GT-FP is our optimized method in Algo. 2.

Metrics. These algorithms are compared in terms of their objective values (*i.e.*, the number of required airplanes and workload balance score). Moreover, we also measure their efficiency in terms of the running time and memory usage.

Environment. All these algorithms are implemented in Python, and the experiments are conducted on a server with Intel Xeon(R) E5 2.30GHz CPUs.

4.2 Experimental Results

Effect on minimum waiting time $T_{\min}$. Fig. 4 illustrates the results when varying the minimum waiting time $T_{\min}$. First, we observe that our GT-FP requires the same number of airplanes as HA-FP and RSA-FP, with consistently better first objectives than GREEDY. For example, when $T_{\min} = 30$ minutes, our GT-FP reduces the number of required airplanes by 35% compared to GREEDY. Meanwhile, our GT-FP exhibits the best workload balance, effectively alleviating the workload imbalance while ensuring the minimum number of required airplanes. For instance, the workload balance score of GT-FP is up to 78% lower than the other compared methods. Additionally, as the minimum waiting time $T_{\min}$ increases, the number of required airplanes increases and the workload balance score decreases due to fewer connecting edges between flights in the bipartite graph. Since the number of airplanes increases, it becomes easier to achieve a good workload balance. In terms of *running time*, our GT-FP is 2 orders of magnitude faster than HA-FP and RSA-FP, demonstrating the effectiveness of our optimization techniques in Sec. 3.2. Although GT-FP is slower than the GREEDY baseline, the gap is below 21s, which is acceptable given the significant reduction in airplane acquisition or leasing costs. Regarding *memory usage*, our GT-FP is always the most efficient due to the compression of bipartite graphs, while the others consume nearly the same space.

Effect on data scalability n. Fig. 5 presents the results of varying the data scale. In terms of effectiveness, our GT-FP consistently achieves the minimum number of required airplanes and the best workload balancing performance. For instance, the minimum required number of airplanes is reduced by up to 32%, and the best workload balancing performance is increased by up to 93% than other algorithms. In terms of time efficiency, with the expansion of data scales, the running time of our GT-FP grows at a slower rate compared to HA-FP and RSA-FP. This trend indicates that our GT-FP outperforms HA-FP and RSA-FP for large-scale datasets. The baseline GREEDY is always the fastest, and our GT-FP is also relatively fast. In terms of space efficiency, our GT-FP can reduce memory usage by up to 3 orders of magnitude compared to other methods. This experiment demonstrates that our GT-FP is relatively efficient on large-scale dataset.

Effect on different days of records. Fig. 6 depicts the results of using different days of data records. The number of flights increases with the increase in the days of flight records. Similar to previous patterns, our GT-FP has the minimum number of required airplanes and the best workload balance score. In terms of *running time*, GREEDY is the fastest and our GT-FP is always the runner-up. HA-FP and RSA-FP require roughly the same amount of time to run, which is up to 77 × slower than our GT-FP, respectively. Moreover, the memory usage of GT-FP is always lower than the others. For instance, GREEDY consumes up to 3 orders of magnitude more space than the GT-FP.

Breakdowns of different airplane types. We select six airplane types that are most commonly seen in our national flight dataset, and Fig. 7 shows the result breakdowns of processing data for different types of airplanes. For any airplane type, our GT-FP always achieves the best objective values, *i.e.*, the minimum number of required airplanes and the lowest workload balance scores. Moreover, the running time for processing any airplane type by GT-FP is consistently lower than that by HA-FP or RSA-FP, and its memory cost is also the lowest. Overall, our GT-FP always outperforms the others in terms of effectiveness and space cost for any airplane type. This experiment verifies the robustness of GT-FP.

Table 4: Results on *Sichuan Airlines* dataset

	Objective #1	Objective #2	Running time	Memory usage
GREEDY	482	14.414	**0.053s**	1557KB
HA-FP	**255**	46.176	0.936s	1557KB
RSA-FP	**255**	41.452	0.666s	1557KB
GT-FP	**255**	**1.583**	0.196s	**13KB**

Case study on *Sichuan Airlines* evaluation environment. To test the real-life performance, we also deploy these compared solutions in the evaluation environment provided by *Sichuan Airlines* company [13]. Table 4 presents the results of this case study, where the best results are marked in bold. Our GT-FP demonstrates the best performance in both optimization objectives. By using GT-FP instead of GREEDY, the number of required airplanes is decreased by 47%, and the workload balance score is decreased by 89%. Furthermore, the memory cost of GT-FP is 2 orders of magnitude lower than that of the other algorithms. As for *running time*, GT-FP is slightly slower than GREEDY, but notably faster than HA-FP and RSA-FP. This case study has demonstrated the superior performances of our GT-FP than the other methods.

Summary. The aforementioned experimental results are summarized as follows:

(1) Our GT-FP consistently ensures the minimum number of airplanes required to serve all flights and demonstrates the best workload balancing performance. For example, in the *Sichuan Airlines* case study, our GT-FP required 47% fewer airplanes than the GREEDY used by *Sichuan Airlines*, while improving load balancing performance by 89%.
(2) Our GT-FP shows a satisfactory time efficiency. For instance, it is up to 2 orders of magnitude faster than HA-FP and RSA-FP. Although GREEDY is faster than GT-FP, the improvements on both optimization objectives make this limited sacrifice worthwhile.
(3) The memory usage of GT-FP is the lowest and can be 3 orders of magnitude lower than the others, indicating the effectiveness of our optimizations.

5 RELATED WORK

Our work is related to existing research on *trip planning over large-scale data* and *bipartite graph matching in real world applications*.

5.1 Trip Planning over Large-Scale Data

Trip planning over large-scale data is an important research direction in spatial databases. Specifically, several studies [9, 17, 24] focus on planning the trips consisting of multiple Points of Interest (PoIs) in travel route recommendation. Other recent work proposes

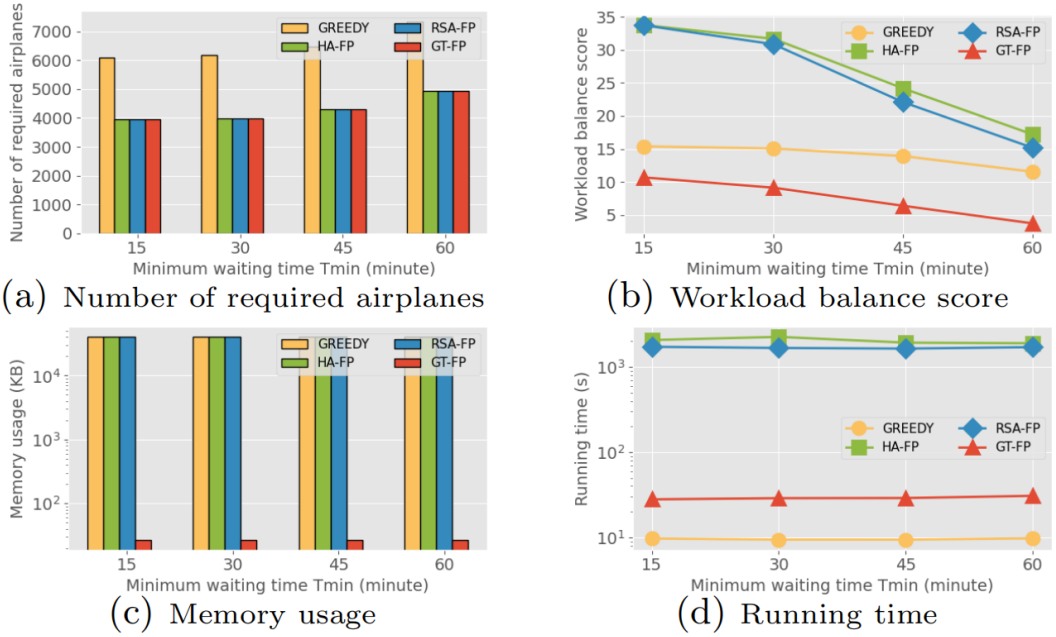

Figure 4: Results of varying minimum waiting time $T_{\min}$

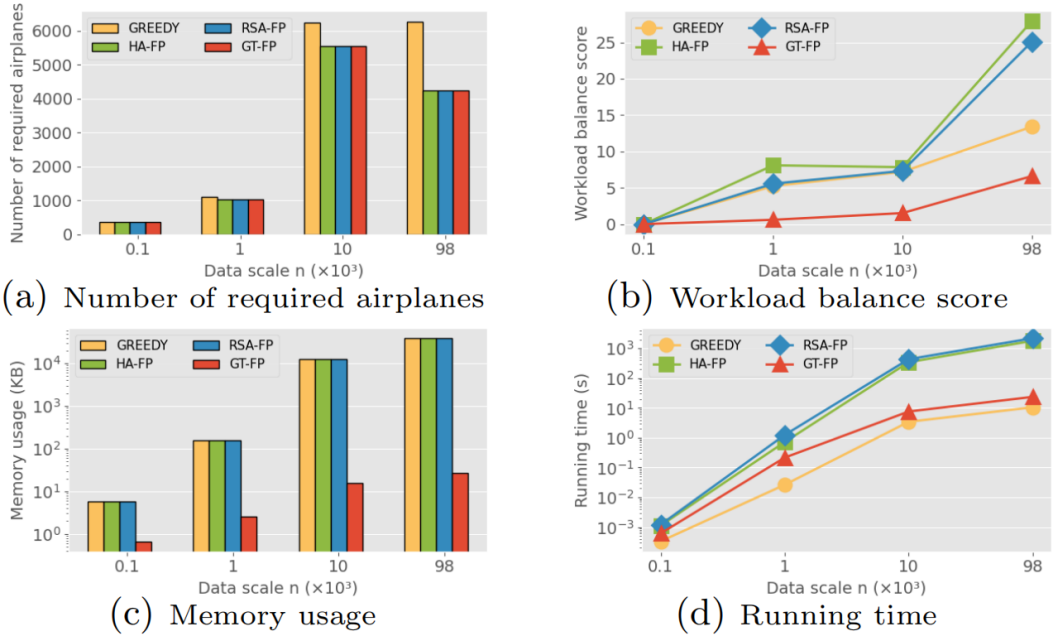

Figure 5: Results on data scalability test

efficient solutions to route planning of delivery orders in shared mobility services, such as ride sharing [6, 21] and urban logistics [15, 22]. These trip planning algorithms usually aim to minimize the total travel distance or maximize the platform's total revenue. By contrast, our FP problem aims to minimize the number of required airplanes and workload balance of these flights, which are quite different from the objectives of existing work. Thus, these methods cannot be applied to our problem.

5.2 Bipartite Graph Matching in Real World Applications

As explained in Sec. 3, our FP problem can be reduced to the bipartite graph matching problem. Many algorithms have been proposed to find the optimal matching for large-scale bipartite graphs, such as the classic Hungarian algorithm and Hopcroft-Karp algorithm [5]. Moreover, bipartite graph matching has been used in various applications, such as task assignment in spatial crowdsourcing [16, 23],

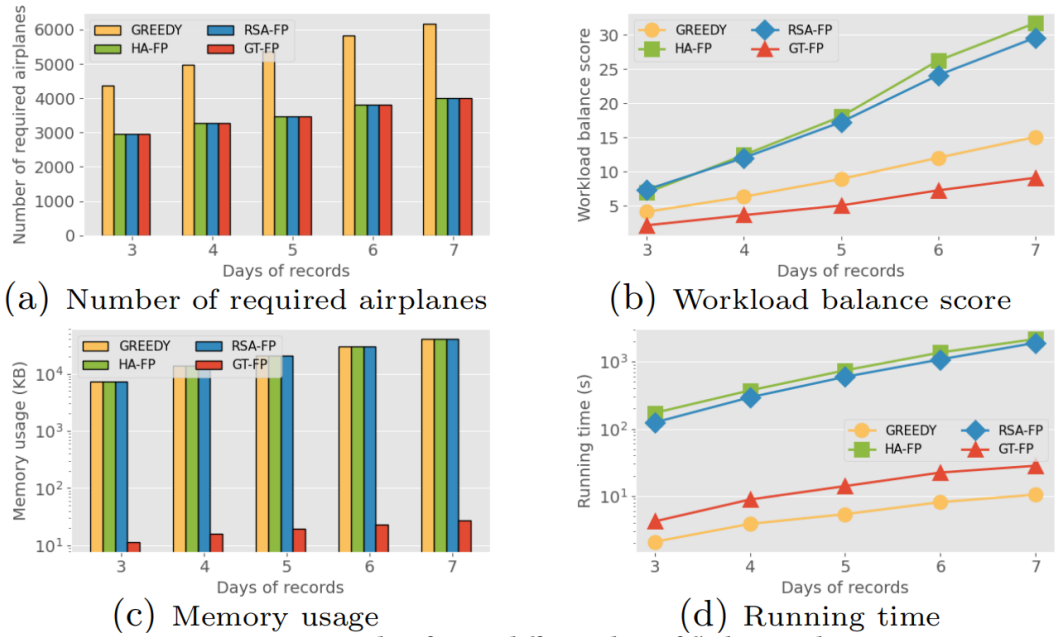

Figure 6: Results of using different days of flight records

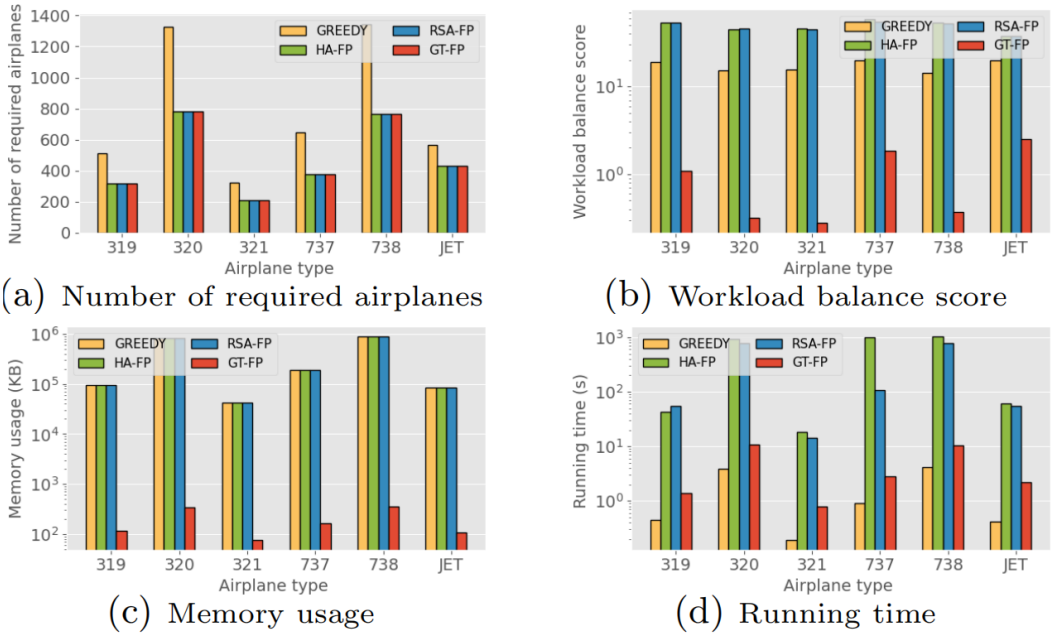

Figure 7: Breakdowns of results on different airplane types

ride-hailing order dispatching [2, 18], and entity resolution [12]. For more bipartite matching algorithms and applications, please refer to [5]. Among these studies, some of them are not designed for geo-spatial applications, and hence cannot solve our problem. The other studies (*e.g.*, research on task assignment and order dispatching) have similar spatiotemporal constraints in their problem settings to ours, but their optimization goals are essentially different. For instance, Ref. [23] is the most closely related work, and this work proposes effective and efficient solutions to task assignment in spatial crowdsourcing for maximizing the workers' payoff while minimizing the payoff differences among these workers. By contrast, our FP problem aims to minimize the number of required airplanes and their workloads' differences. Here, an airplane cannot be simply regarded as a "worker", since it gets no "payoff". Thus, it is still challenging to design effective solutions to our problem.

6 CONCLUSION

This paper studies the large-scale flight planning problem with multi-objectives of minimizing the required airplanes and optimizing workload balance. To solve this problem, we propose a bipartite matching based framework and design optimizations to compress the bipartite matching and enhance the objective. Moreover, we also prove that our solution has theoretical guarantees on the effectiveness of both objectives. Specifically, our solution not only requires the least number of airplanes, but also achieves a Pure Nash Equilibrium in balancing these airplanes' workload. Finally, we have conducted extensive experiments on two real-world datasets. The evaluation has demonstrated the promising performances of our solution in terms of both effectiveness and efficiency.

ACKNOWLEDGEMENTS

We are grateful to the reviewers for their constructive comments. This work is partially supported by National Science Foundation of China (NSFC) under Grant No. U21A20516, 6233000216, and 62076017, Beijing Natural Science Foundation No. Z230001, CCF-Huawei Populus Grove Fund, Didi Collaborative Research Program NO2231122-00047, and the Beihang University Basic Research Funding No. YWF-22-L-531. Yi Xu is the corresponding author of this paper.

REFERENCES

[1] Ahmed F. Abdelghany, Khaled F. Abdelghany, and Farshid Azadian. 2017. Airline flight schedule planning under competition. *Comput. Oper. Res.* 87 (2017), 20–39.
[2] Tenindra Abeywickrama, Victor Liang, and Kian-Lee Tan. 2021. Optimizing Bipartite Matching in Real-World Applications by Incremental Cost Computation. *PVLDB* 14, 7 (2021), 1150–1158.
[3] Mohamed Ben Ahmed, Maryia Hryhoryeva, Lars Magnus Hvattum, and Mohamed Haouari. 2022. A matheuristic for the robust integrated airline fleet assignment, aircraft Transportation Researc routing, and crew pairing problem. *Comput. Oper. Res.* 137 (2022), 105551.
[4] Nayla Ahmad Al-Thani, Mohamed Ben Ahmed, and Mohamed Haouari. 2016. A model and optimization-based heuristic for the operational aircraft maintenance routing problem. *Transp. Res. Part C Emerg. Technol.* 72 (2016), 29–44.
[5] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. 2012. *Assignment problems: revised reprint.* SIAM.
[6] Lu Chen, Qilu Zhong, Xiaokui Xiao, Yunjun Gao, Pengfei Jin, and Christian S. Jensen. 2018. Price-and-Time-Aware Dynamic Ridesharing. In *ICDE*. 1061–1072.
[7] Abdelrahman E. E. Eltoukhy, Felix T. S. Chan, and Sai Ho Chung. 2017. Airline schedule planning: a review and future directions. *Ind. Manag. Data Syst.* 117, 6 (2017), 1201–1243.
[8] IATA. [n.d.]. Air Passenger Numbers to Recover in 2024. https://www.iata.org/en/pressroom/2022-releases/2022-03-01-01.
[9] Jing Li, Yin David Yang, and Nikos Mamoulis. 2013. Optimal Route Queries with Arbitrary Order Constraints. *IEEE Trans. Knowl. Data Eng.* 25, 5 (2013), 1097–1110.
[10] National Flight Dataset. 2023. https://anonymous.4open.science/r/FPDataset.
[11] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. 2007. *Algorithmic Game Theory.* Cambridge University Press.
[12] George Papadakis, Vasilis Efthymiou, Emmanouil Thanos, Oktie Hassanzadeh, and Peter Christen. 2023. An analysis of one-to-one matching algorithms for entity resolution. *VLDBJ* 32, 6 (2023), 1369–1400.
[13] Sichuan Airlines Ltd. 2023. Sichuan Airlines. https://global.sichuanair.com.
[14] Anika Tabassum, Sukarna Barua, Tanzima Hashem, and Tasmin Chowdhury. 2017. Dynamic Group Trip Planning Queries in Spatial Databases. In *SSDBM*. 38:1–38:6.
[15] Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Lei Chen, and Ke Xu. 2022. Unified Route Planning for Shared Mobility: An Insertion-based Framework. *ACM TODS* 47, 1 (2022), 2:1–2:48.
[16] Yongxin Tong, Zimu Zhou, Yuxiang Zeng, Lei Chen, and Cyrus Shahabi. 2020. Spatial crowdsourcing: a survey. *VLDBJ* 29, 1 (2020), 217–250.
[17] Sheng Wang, Mingzhao Li, Yipeng Zhang, Zhifeng Bao, David Alexander Tedjopurnomo, and Xiaolin Qin. 2018. Trip Planning by an Integrated Search Paradigm. In *SIGMOD*. 1673–1676.
[18] Yansheng Wang, Yongxin Tong, Cheng Long, Pan Xu, Ke Xu, and Weifeng Lv. 2019. Adaptive Dynamic Bipartite Graph Matching: A Reinforcement Learning Approach. In *ICDE*. 1478–1489.
[19] WebXml. 2023. http://www.webxml.com.cn/webservices.
[20] Douglas B. West. 2001. *Introduction to Graph Theory, 2nd Edition.* Prentice Hall.
[21] Yi Xu, Yongxin Tong, Yexuan Shi, Qian Tao, Ke Xu, and Wei Li. 2022. An Efficient Insertion Operator in Dynamic Ridesharing Services. *IEEE TKDE* 34, 8 (2022), 3583–3596.
[22] Yuxiang Zeng, Yongxin Tong, and Lei Chen. 2019. Last-Mile Delivery Made Practical: An Efficient Route Planning Framework with Theoretical Guarantees. *PVLDB* 13, 3 (2019), 320–333.
[23] Yan Zhao, Kai Zheng, Jiannan Guo, Bin Yang, Torben Bach Pedersen, and Christian S. Jensen. 2021. Fairness-aware Task Assignment in Spatial Crowdsourcing: Game-Theoretic Approaches. In *ICDE*. 265–276.
[24] Chenghao Zhu, Jiajie Xu, Chengfei Liu, Pengpeng Zhao, An Liu, and Lei Zhao. 2015. Efficient Trip Planning for Maximizing User Satisfaction. In *DASFAA*. 260–276.