Swift: A Data-Driven Flight Planning System at Scale

Chang Gao
SKLCCSE Lab, Beihang University
gc021129@buaa.edu.cn

Tianlong Zhang
SKLCCSE Lab, Beihang University
tianlong@buaa.edu.cn

Yuxiang Zeng
SKLCCSE Lab, Beihang University
yxzeng@buaa.edu.cn

Yi Xu
SKLCCSE Lab, Beihang University
xuy@buaa.edu.cn

Shuyuan Li
SKLCCSE Lab, Beihang University
lishuyuan@buaa.edu.cn

Yuanyuan Zhang
North China Institute of Computing Technology
zyy-buaa@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:
Chang Gao, Tianlong Zhang, Yuxiang Zeng, Yi Xu, Shuyuan Li, and Yuanyuan Zhang. Swift: A Data-Driven Flight Planning System at Scale. PVLDB, 17(12): 4465 - 4468, 2024.
doi:10.14778/3685800.3685901

1 INTRODUCTION

Airplanes are probably the most convenient mode of long-distance transportation in daily life. The International Air Transport Association predicts a new high in air passenger volume, reaching 4 billion in 2024. As demand grows, the scalability and complexity of wisely planning appropriate airplanes to cover scheduled flights (referred to as "flight planning") are also increasing. Thus, it's a fundamental challenge for airlines to devise cost-effective, flexible flight plans for large-scale demands.

Traditional solutions, such as linear programming, branch-and-bound, and heuristic search, are primarily effective for medium-scale data [5]. However, as the data size of flight itineraries increases to large-scale, these solutions can become inefficient and, therefore, no longer suitable for meeting the real-world situation. Consequently, in recent years, both industry and academia focus on designing spatio temporal data driven approaches to effectively solve large-scale planning problems.

Recently, the databases community has proposed several solutions for large-scale spatio-temporal planning, including route planning in ride-sharing [3, 4], vehicle delivery planning in urban logistics [7], and trip planning for travel recommendation [6]. These studies prioritize minimizing travel distance or maximizing revenue. Nevertheless, they cannot effectively manage flight planning due to the following differences that originate from real-world scenarios:

(1) **Airlines have their own primary concern**. To efficiently serve all the anticipated itineraries, airlines must acquire sufficient airplanes, which is a principal operational expense. They also seek to evenly distribute usage to prolong airplanes service life. Therefore, flight planning objectives are twofold: (a) minimizing aircraft numbers and (b) balancing the workload across each plane.

(2) **Planned airplane routes may still undergo dynamic updates.** Bad weather is probably the most common factor that leads to dynamic updates in pre-planned airplane routes. Moreover, a delayed flight can cause a ripple effect throughout the entire schedule, which hurts passengers' experience. Thus, a flight planning system should be flexible to accommodate dynamic adjustments.

Thus, we are motivated to build a prototype system called Swift for large-scale flight planning. Swift is composed of three core modules: *itinerary demand prediction, flight graph matching, and dynamic plan update*. Specifically, we first collect various types of spatiotemporal data, such as historical ticket booking records and weather data, to predict the future demand for flight itineraries. Then, we aim to decide the minimum number of required planes to cover the predicted flights and assign a subset of flights with a balanced workload to each plane. To optimize both objectives, we have reduced this planning problem to a bipartite graph matching problem, solved it using maximum cardinality bipartite matching optimized for time efficiency and workload balance. Additionally, we have designed flexible strategies to handle dynamic updates to replan routes for planes that encounter delayed flights by minimizing the total latency. Finally, all these modules have been integrated into a user-friendly web client, and this client enables users or airlines to visualize, monitor, and manage flight plans in real time.

2 SYSTEM OVERVIEW

2.1 Basic Concepts in Flight Planning System

We introduce the basic concepts related to our Swift system.
Flight Table. Flight table is a collection of flight itineraries ("flights" as short). Each flight in the table is mainly composed of *spatio temporal data*, such as its departure airport, arrival airport, departure

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685901

time, and arrival time. It also includes other attributes, such as the type designator of the assigned flight.

Flight Plan Result. A flight plan result ("flight plan" as short) represents the outcome of a flight planning method. The result is expressed as a collection of tuples, where each tuple consists of two elements: the required airplane and its assigned flights. These flights, retrievable from the flight table, are arranged in a pre-defined order, indicating the designated schedule for the assigned airplane.

Flight Plan Update. In practice, a flight plan assigned to an airplane can be dynamically changed due to external factors, such as bad weather conditions. Therefore, the update of a flight plan is also an important consideration. Specifically, an update on the pre-scheduled flight plan mainly includes two solutions: canceling and delaying. For instance, suppose an airplane's plan is $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$. It means this airplane will serve the flights from A to B, from B to C, from C to D, and so on. If the city B experiences a rainstorm, the airline could delay the trip from A to B, which may inevitably result in delays for subsequent trips (*i.e.*, the ripple effect as aforementioned). Alternatively, the airline could opt to cancel two trips $A \rightarrow B$ and $B \rightarrow C$, and instead have the airplane fly directly to the city C from its current location, the city A.

Our Goal: Flight Planning at Scale. Based on the previous concepts, we now introduce our ultimate goal: *flight planning at scale*. Specifically, given a flight table with massive flights, we aim to compute an *optimal* flight plan that *minimizes the number of required airplanes and balances the workloads* among these airplanes. This flight plan needs to encompass all the flights and adhere to their designated time arrangements. Furthermore, we also strive to offer *a flexible and real-time solution to flight plan updates* by minimizing the overall delay to the unserved and impacted flights.

2.2 System Architecture

To achieve this goal, we have built four core components within our system: *itinerary demand prediction*, *flight graph matching*, *dynamic plan update*, and *flight plan visualization*, as shown in Fig. 1.

Itinerary Demand Prediction. To create an effective flight plan, the initial task for an airline is to accurately predict the itinerary demand between cities. This forecasting helps optimize resource management and boost operational efficiency. Therefore, this module is designed to forecast itinerary demand using historical data and external factors like weather and aircraft capacity, ultimately generating a potential flight table, serving as main inputs for the subsequent modules.

Flight Graph Matching. This module serves as the core of the flight planning process, receiving the flight table as its primary input and subsequently computing an initial flight plan. To attain this objective, we have designed a novel bipartite graph matching-based algorithm to make a flight plan for large-scale datasets. The algorithm details will be deferred in Sec. 3.2. This algorithm guarantees that the flight plan requires the minimum number of airplanes and ensures that their flight workloads are as balanced as possible.

Dynamic Plan Update. Leveraging the initial flight plan generated by the preceding module, this module dynamically adjusts the overall plan in response to factors like bad weather and air traffic control. When such unexpected occurrences happen, an airline must decide whether to delay or cancel specific flights in real-time. Thus, this module computes the possible changes to the original

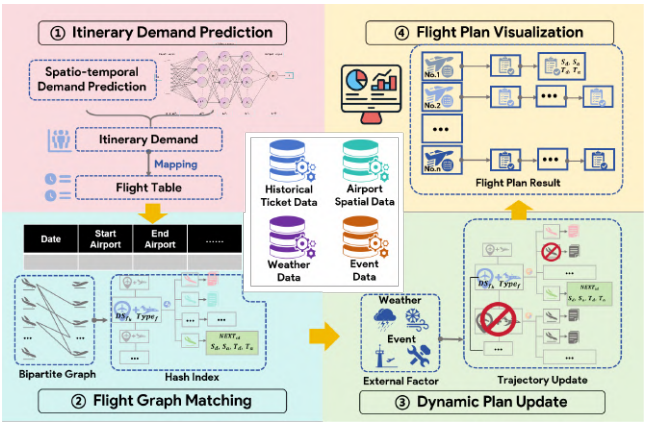

Figure 1: Architecture of our Swift system

flight plan, provides a comprehensive analysis of the total delay, and enables the airline to make a proper decision.

Flight Plan Visualization. This module functions as an interactive interface tailored for airlines' needs. Leveraging a front-end interface, users can access and visualize the predicted flights and flight plan results. Moreover, it also allows users to customize the local conditions (*e.g.*, weather) of the airports and provides real-time updates on the flight plan accordingly.

3 PROTOTYPE IMPLEMENTATION

3.1 Itinerary Demand Prediction Module

This module aims to predict the air transportation demands of passengers for a certain time period and subsequently generate a flight table based on the predicted flights.

To achieve this objective, we initially collect relevant data from the Internet. Specifically, we have developed a web crawler to gather 98,158 flight records from the WebXML website [1], serving as our historical flight dataset. This large-scale dataset offers rich information about all existing flights in China in 2023, including their departure/arrival airports, departure/arrival time, total travel duration, ticket prices, etc. Moreover, it involves a diverse range of 52 airplane types operated by 29 Chinese airlines. Similarly, we have also crawled seasonal weather data, flight delay data, and airport event data from the internet.

After collecting sufficient data, we proceed to tackle the air traffic demand problem, drawing inspiration from existing research on the spatio temporal graph neural network model, ST-MGCN [2]. Originally, ST-MGCN was proposed to forecast the demand for ride-hailing services between any two different regions. Similarly, we treat each airport as a spatial region in this model. We also create a transportation connectivity graph as the average flight time between cities and an airport similarity graph that captures their resemblance in terms of weather and event data representation embeddings. By using ST-MGCN, we can effectively predict the future flight demand for a certain time period (*e.g.*, the next season).

Predicted flights are derived from air transportation demands based on historical flight records and maintained in a flight table.

3.2 Flight Graph Matching Module

This module produces the initial flight plan by a bipartite graph matching-based method with two steps: *matching* and *optimizing*.

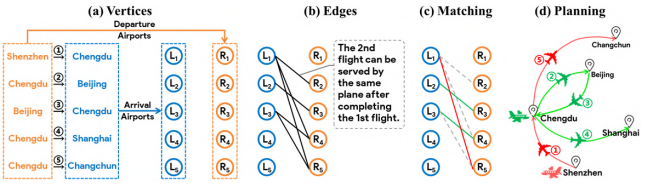

Figure 2: Illustration of bipartite graph matching

Matching. To minimize the number of required airplanes, we introduce a bipartite graph matching algorithm to generate the initial flight plan, as illustrated in Fig. 2. In this bipartite graph, a right-hand vertex R_i represents a departure airport, while its opposite left-hand vertex L_i represents the corresponding arrival airport for the same flight. Besides, if the arrival airport of the jth flight is also the departure airport of the ith flight, and there is sufficient airplane maintenance time between the arrival of the jth flight and the departure of the ith flight, an edge is established between vertices L_j and R_i. This edge indicates that a single plane can consecutively serve both flights. Thus, by finding the maximum cardinality bipartite matching, this method minimizes the number of required planes by enabling a single plane to serve as many flights as possible.

Optimizing. A naive implementation of maximum cardinality bipartite matching suffers from inefficiency under large-scale datasets, *e.g.*, the classic Hungarian algorithm takes cubic time complexity and quadratic space complexity. To *improve the scalability*, we compress the adjacency list into a hash index, where each hash bucket denotes an arrival airport. Then, for each arrival airport, we retrieve its adjacent right-hand vertices by using a time window query over the departure time of the corresponding airport. Moreover, to *balance the airplanes' workload*, we use an *optimization method* called swap. This operation tries to exchange some consecutive flights between the plans of two airplanes. If a swap can result in a more balanced workload, we will adopt the new flight plan. The swap operation is designed to converge at a Pure Nash Equilibrium state, ensuring the effectiveness of balancing the workloads.

Due to the page limitation, please refer to our full paper [9] for formal proofs on the optimality of this flight planning algorithm.

3.3 Dynamic Plan Update Module

This module is implemented to promptly respond to dynamic updates regarding flight situations, thus mitigating the ripple effects of delays. In general, *delays* are categorized into three levels: *mild*, *moderate*, and *severe*, based on the total late duration of the affected flights and their subsequent ones. These categories enable us to tailor the update strategies according to the severity of the delay:

(1) *Mild delays* usually require no immediate change;

(2) For *severe delays*, the system prompts the skipping of the affected flights and moving on to the subsequent ones;

(3) *Moderate delays* often require personalized decision-making by the airline. In such cases, we provide necessary delay information to assist the airline in making informed decisions.

Moreover, to reflect the latest flight situation, we also update the bipartite graph and its underlying hash index whenever there are changes in the flights. Finally, once a decision has been made regarding whether to retain or cancel the affected flights, this module seamlessly integrates the changes into the current flight plan.

3.4 Flight Plan Visualization Module

This module leverages the JavaScript visualization library, Apache ECharts, to build a robust web client and achieve these objectives:

(1) It can visualize diverse types of relevant data, such as spatiotemporal information of airports, historical flight itineraries, predicted flights, and the status of the served airplanes.

(2) It can simulate the intricate process of flight planning algorithms, including their initial plans and dynamic updates;

(3) It can analyze the performance of existing flight planning solutions, including two baselines from our previous work [8] based on Hungarian and Greedy respectively, and an optimized algorithm with implementation details in our full paper [9].

This module also employs several different ways to show the algorithm performance, such as heat maps, radar charts, and bar charts. Moreover, interactive functionality allows users to click on the map, change the status of airports (*e.g.*, delays due to bad weathers), and observe dynamic updates to the current flight plan.

4 DEMONSTRATION SCENARIO

4.1 Demonstrations of Flight Planning Methods

The first demonstration scenario will showcase to the audience the performance comparisons of different flight planning methods.

Visualizations of Prediction Flights. The interface in Fig. 3 provides users access to the predicted flight flows and generated flight table. Specifically, in Fig. 3 (a), users have the option to select different flight planning algorithms (*e.g.*, Hungarian, Greedy, and our method), and filter out predicted flights based on the selected departure airports or specific airplane types (*e.g.*, A319). Depending on the users' selections, Fig. 3 (b) reflects the predicted itinerary demand, and Fig. 3 (e) displays the distributions of the overall flight table through heatmaps.

Comparisons of Flight Planning Algorithms. Furthermore, based on the chosen flight planning algorithm, Fig. 3 (c) draws a radar chart that evaluates its performance across five dimensions: the *total number of (required) airplanes*, *workload balanced score*, *operating cost assessment* (*i.e.*, The cost of leasing the required airplane by the airline), *time efficiency*, and *space efficiency*. Accordingly, this enables a comprehensive comparison between existing methods [8] and our new method [9] (see Fig. 3 (f)). Our new method requires 33% less airplanes than the Greedy baseline [8] and achieves more balanced workload than the classic Hungarian. Moreover, this new method is up to two orders of magnitude faster than Hungarian. Fig. 3 (d) also depicts the flight workload assigned to each required airplane in a bar chart.

Overall, this scenario provides a comprehensive and interactive way for users to explore our prediction results and compare different flight planning methods when processing large-scale datasets.

4.2 Monitoring Dynamic Flight Plan Updates

In this demonstration scenario, audiences can monitor live flight trajectories, make real-time adjustments to flights, and observe the corresponding immediate updates accordingly.

Simulating Real-time Airplane Scheduling. The map displays real-time flight trajectories corresponding to the current time period, such as 2020/12/02 09:15 as illustrated in Fig. 4 (a). The simulation progresses in batches every 5 minutes, with flight trajectories

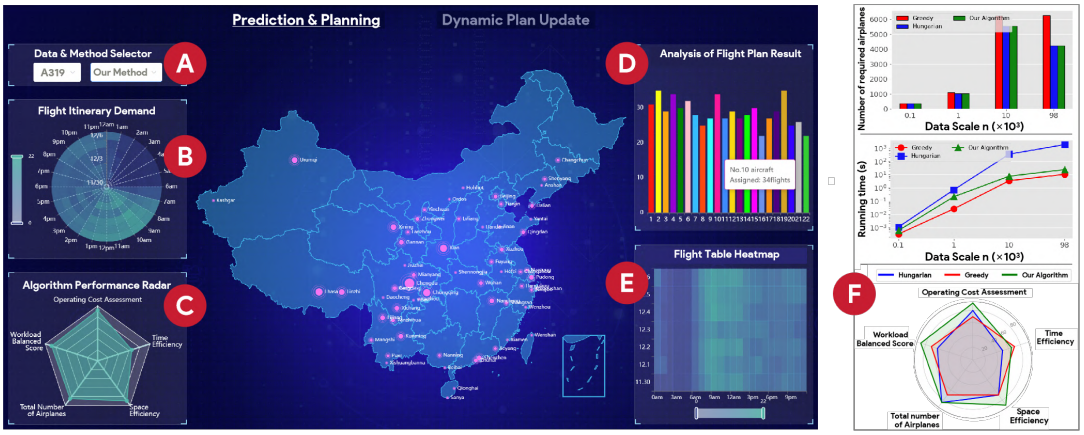

Figure 3: Visualization of predicted flights and comparisons of flight planning algorithms in our Swift system

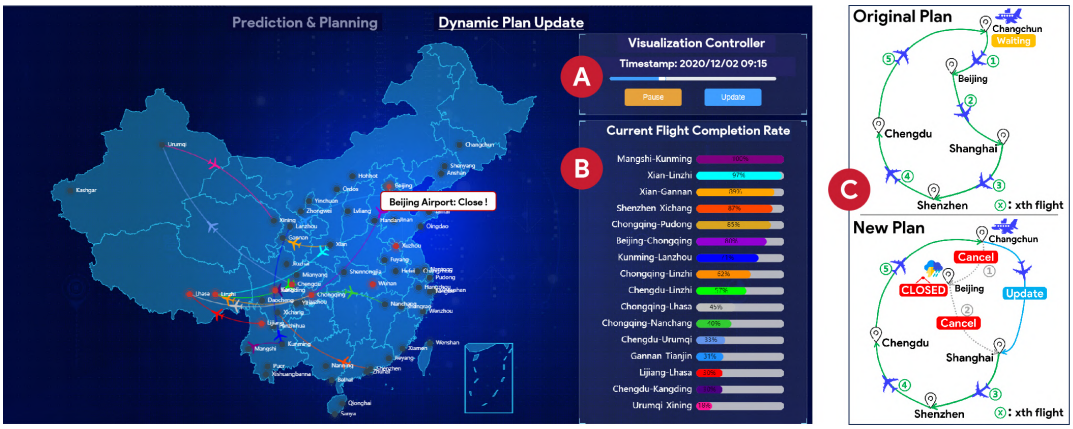

Figure 4: Simulating and monitoring dynamic plan updates in our Swift system

dynamically updating on the map. Additionally, Fig. 4 (b) demonstrates real-time flight status, with each progress bar representing the completion rate of a certain flight itinerary. For example, the flight itinerary from the city Mangshi to the city Kunming has been fully completed, resulting in a completion rate of 100%. Similarly, for the flight from Xi'an to Linzhi, the completion rate is 97%, indicating that the airplane is almost arriving at Linzhi airport. Users also have the flexibility to pause the simulator at any desired time in Fig. 4 (a), enabling them to observe the global status of airplanes.

Monitoring Dynamic Flight Updates. In this scenario, users can simulate unexpected flight incidents caused by bad weather and make accordingly adjustments to the current flight plan. As shown in Fig. 4 (c), if Beijing airport is temporarily closed due to a rainstorm, users can click on the marker of Beijing airport on the map, causing it to change color from black to red. Subsequently, flights scheduled to land in Beijing will be re-routed to the nearest available airport, while those scheduled to take off will be canceled. The completion rate displayed in Fig. 4 (b) will also adjust accordingly. Moreover, this incident will result in an increasing number of airports highlighted in red, indicating *severe delays* for departing flights. As depicted in Fig. 4 (c), an airline may decide to cancel some severely delayed flights involving the Beijing airport and promptly update the original flight plan for certain airplanes, in order to mitigate the ripple effect of flight delays.

ACKNOWLEDGMENT

This work was supported by National Key Research and Development Program of China under Grant No. 2023YFF0725103, National Science Foundation of China (Grant Nos. 62076017, U21A20516, 62336003) and Beijing Natural Science Foundation (Z230001), the Basic Research Funding in Beihang University No.YWF-22-L-531, and Didi Collaborative Research Program NO2231122-00047. Yuxiang Zeng and Yi Xu are the corresponding authors in this paper.

REFERENCES

[1] 2024. WebXML. http://www.webxml.com.cn/webservices.
[2] Xu Geng, Yaguang Li, Leye Wang, et al. 2019. Spatiotemporal Multi-Graph Convolution Network for Ride-Hailing Demand Forecasting. In *AAAI*. 3656–3663.
[3] James Pan, Guoliang Li, and Yong Wang. 2020. Evaluating Ridesharing Algorithms using the Jargo Real-Time Stochastic Simulator. *PVLDB* 13 (2020), 2905–2908.
[4] Yongxin Tong, Jieying She, Bolin Ding, Libin Wang, and Lei Chen. 2016. Online mobile Micro-Task Allocation in spatial crowdsourcing. In *ICDE*. 49–60.
[5] Paolo Toth and Daniele Vigo. 2002. *The vehicle routing problem.* SIAM.
[6] Sheng Wang, Mingzhao Li, Yipeng Zhang, et al. 2018. Trip Planning by An Integrated Search Paradigm. In *SIGMOD*. 1673–1676.
[7] 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.
[8] Yuxiang Zeng, Yongxin Tong, Yuguang Song, et al. 2020. The Simpler The Better: An Indexing Approach for Shared-Route Planning Queries. *PVLDB* 13 (2020), 3517–3530.
[9] Tianlong Zhang, Chang Gao, Yuxiang Zeng, et al. 2024. Flight Planning at Scale: A Bipartite Matching based Approach. https://github.com/Cecelia-cc/Flight-Planning