Flight Radius Algorithms

Keywords:
Flight Radius Problem, Route Network Development, Time-independent Model, Graph Database, Shortest Path Algorithms.

Abstract:
In this article, we present the flight radius problem (FRP) on the condensed flight network. The problem is derived from a decision tool for airline managers to analyze and simulate a new market. It concerns the network design and visualization when allocating a new flight. The flight radius problem consists in finding routes passing through a specific flight that represent business opportunities, for instance, regarding time and cost criteria. It is formulated as finding a maximal subgraph of nodes belonging to routes satisfying a regret constraint. This constraint is defined as a function of the optimal values of the time, distance, and cost criteria. We propose to compare four algorithms: two algorithms that decompose the FRP in shortest path problems (SPP) and solved them in parallel using either the Dijkstra or Bellman algorithm; the third algorithm extends the Dijkstra algorithm to avoid useless computations; the fourth algorithm extends the Bellman algorithm to compute all shortest paths for all criteria at once. These four algorithms perform parallel computation when it is possible. The experimental evaluation demonstrates that the best algorithm is the third one that meets the real-time constraints of the targeted industrial application.

1 INTRODUCTION

The growth of air passenger needs has forced airlines to improve their quality of service. The airlines should offer flights that match with preferences of passengers. For this reason, most of airlines are interested in quality of service models (QSI). QSI is a market share model used by most of airlines to estimate their part of the market. The model determines the probability a traveler selects a specific itinerary connecting an airport pair based on a list of criteria (Jacobs et al., 2012). Then, let us suppose an airline network, where nodes are the airports and arcs represent flights. Each arc is associated with a time, distance, and cost. The question is to decide if it is interesting to add a flight between a pair of airports (Origin-Destination) in that network. More precisely, adding a new arc would allow to passengers to make itineraries that are neither longer nor more expensive than going directly to their destination? ? For this, the flight radius problem (FRP) consists in finding routes that pass by a potential arc and satisfy a regret constraint. This constraint aims to model passengers preferences. Practically, there are many criteria to be considered but the three main considered in this study are: time, distance, and cost. This problem is related to the route network development problem which is considered as the initial problem addressed by the airlines. It aims to determine a set of routes to be operated in an airline's network (Idrissi et al., 2017).

The FRP is derived from the `PlanetOptim` application developed by the company which is a startup specialized in air transportation and has developed a decision tool for airline managers to analyze and simulate a new market using QSI models. Then, given a specific flight that is defined by an airport pair (origin & destination), the process in the application starts by finding important airports with respect to the specific flight, and then estimates market share for each route connecting the origin to the destination. Following this, the airline manager decides whether to add the new flight or not.

Our study goal is to improve the process of simulating a new market. Thus, the solution proposed by the FRP helps to reduce the size of the network by removing uninteresting routes regarding criteria of QSI models. Basically, it will allow to accelerate QSI computations by considering only interesting routes.

For instance, Figure (1) represents a screen-shot of the application. Let us consider this example for which a new route will be created between `NCE` and `BKK`. This new route allows passengers coming forward `NCE` (in connection with `NCE`) to go either to `PEK` or `ICN` via `BKK`. Passing by the new flight can be a little bit longer (respectively shorter) but cheaper (respectively more expensive) than going directly to the final destination . The choice depends on the passengers since they have different preferences over the criteria. For this reason, these preferences should be integrated in this problem.

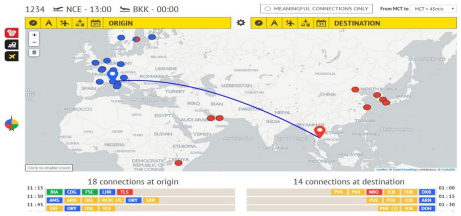

Figure 1: Screen-shot of the `PlanetOptim` application. Colors of airports refer to their localization towards the compass.

(Idrissi et al., 2018) formulated the FRP as finding a maximal sub-graph such for each node there exists a supported path by the regret constraint. Given a certain regret, we are searching for paths that are neither longer nor more expensive than the shortest path. The regret is defined for each time and cost criteria. This represents the real life, when a passenger accepts to pass through the new arc, the new path is accepted regarding to a certain regret. Then, the solution either minimizes the cost or minimizes the time. Such paths can be retrieved by using shortest paths algorithms. We begin simply by working on the condensed flight network (CFN). In such network, we omit scheduling information and keeps only transfer time information. This approach yields to smaller graph size (See section 4).

The authors proposed in (Idrissi et al., 2018) two methods to solve the problem. The first one is a query based on available procedures in the database where the condensed flight network is stored (See section 4.2). The second one is an optimized algorithm based on Dijkstra algorithm. Tests run to solve a part of the problem. It means in outgoing direction. The algorithm is more efficient than the query and meets real-world response time constraints.

In this paper, we compare four methods to solve the FRP. The first two are algorithms that decompose the FRP in shortest path problems (SPP) and solved them in parallel using as shortest path algorithm: Dijkstra and Bellman. The third one extends the algorithm presented in (Idrissi et al., 2018) to solve the FRP in both directions with more than two criteria. However, the last one uses a Bellman-Ford algorithm to solve the problem that computes at once all shortest paths from both origin and destination for all criteria.

The four algorithms perform parallel computation when it is possible. Our aim is to study the performance of a shortest path algorithm like Bellman-Ford on solving the problem compared to Dijkstra algorithm when adding more criteria. We are especially interested in comparison the runtimes of these algorithms which is important for the user experience of `PlanetOptim`. Another interesting metric is the number of nodes scanned.

The paper is organized as follows. Section 2 introduces some definitions of graph theory, then, introduction of flight timetables and some algorithms of shortest path. In Section 3, we review related work of the air scheduling development problems and transportation networks modelling approach. Section 4 describes the CFN modelling, the graph database, and also how we implement the CFN in the graph database. Section 5 gives the formulation of the FRP, and its properties. Section 6 describes the four algorithms. Section 7 is dedicated to experiments, questions we aim to answer in this study.

2 PRELIMINARIES

This section gives definitions of graph theory (Ahuja et al., 1993). Then, it describes the flight timetables. To sum up, the section states some shortest path algorithms.

2.1 Graph Theory

A *graph* G is a couple $G = (V, E)$ consisting of a finite set V of nodes or vertices and a set $E \subseteq V \times V$ of arcs which are ordered pairs (u, v) if the graph is directed. The node u is called the *tail* of the arc, and v is called the *head*. Each arc $(u, v) \in E$ has an associated non-negative weight $w(u, v)$. We define $|V| = n$, the order of the graph as the number of nodes meanwhile $|E| = m$ its size. Airline networks are weighted directed graphs where weights represent the cost or the duration of the

flight. A direct flight from one city to another does not necessarily imply that there is also a direct return flight. A graph $G' = (V', E')$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. We say that G' is the subgraph of G *induced* by V' if E' contains each arc of E with both endpoints in V'. A *path* is a sequence of nodes $\{v_1, v_2, ..., v_k\}$ such that for each $1 \leq i < k, (v_i, v_{i+1}) \in E$. If additionally $v_1 = v_k$, then the path is a *cycle*. The length of a path P is the sum of its arc weights along the path and is denoted by:

$$l(P) := \sum_{i=1}^{k-1} w(v_i, v_{i+1}).$$

$l^\star(s,t)$ for a given pair of nodes s and t, is the length of the shortest path starting at s and ending at t. A path in G is called *simple* if no node occurs more than once. A graph G is *connected* if there exists a path joining any two nodes. A transportation network should be a connected graph.

In this study, we restrict to flight networks that rely on timetables. The section 4 describes the modelling of the CFN.

2.2 Flight Timetables

A *flight timetable* is defined by a tuple $(\mathcal{C}, \mathcal{A}, \mathcal{F}, \mathcal{T})$ where $\mathcal{A}$ is a set of airports, $\mathcal{F}$ is a set of flights, $\mathcal{T}$ is the periodicity of the timetable, and $\mathcal{C}$ is a set of elementary connections. An *elementary connection* $c \in \mathcal{C}$ is a tuple $c = (f, o, d, t_s, t_e)$ which represents *flight* $f \in \mathcal{F}$ departing from the airport $o \in \mathcal{A}$ at $t_s < \mathcal{T}$ and arriving at the airport $d \in \mathcal{A}$ in time $t_e < \mathcal{T}$ (Pajor, 2009). Concretely, an elementary connection corresponds to an event in the flight timetable. A *passenger trip* $(c_1, c_2, \ldots, c_{n-1}, c_n)$ is a sequence of elementary connections, with the origin of an elementary connection is the same as the destination of its predecessor in the sequence, and the elapsed time between two successive connections is at least as great as the minimum connecting time:

$$\begin{cases} o(c_{i+1}) = d(c_i) \\ t_e(c_i) + MCT(d) \leq t_s(c_{i+1}) \ , \quad \forall\ 1 \leq i \leq n-1 \end{cases}$$

Where MCT is the minimum connecting time at the destination airport $d(c_i)$.

In this study, we omit schedule information by working on the condensed network (see section 4). In such network, we just consider the transfer time without checking if the route is viable, route that respects time constraints. However, the solution will give a lower bound on travelling time.

2.3 Shortest path Algorithms

Shortest path algorithms are based on labeling method for solving the shortest path problem.

Labeling method. The labeled method is defined as follows (Cherkassky et al., 1996). For each node v, the method maintains a distance label $d(v)$ which is an upper bound on the shortest path length to the node v, parents $p(v)$, and status $S(v)$. We have three status : unreached, labeled, and scanned. Initially for each node v, $d(v) = inf$, $p(v) = nil$, and $S(v) = unreached$. For a start node, the method sets $d(s) = 0$ and $S(s) = labeled$. Then, the algorithm starts by scanning labeled nodes until there does not exist such node. The `SCAN` operation of a labeled node consists in checking for all outgoing arcs $(v, w) \in E$, if $d(v) + l(v, w) < d(w)$. Then, if it is, $d(w)$ is updated. If there are no negative cycles, the arcs $(p(v), v)$ form a tree T rooted s. When the algorithm stops, T is a shortest path tree (SPT).

Function 1: SCAN(v)

```
foreach (v,w) ∈ E do
    if d(v) + l(v,w) < d(w) then
        d(w) ← d(v) + l(v,w);
        S(w) ← labeled;
        p(w) ← v;
    end
end
S(v) ← scanned;
```

There are two categories of shortest path algorithms: setting algorithms and correcting algorithms. The two types of algorithms differ in the strategy of selecting labeled nodes to be scanned (Cherkassky et al., 1996). We present Dijkstra's algorithm and its speed-up techniques: Bidirectional Dijkstra and A* search. We end by describing Bellman-Ford-Moore algorithm.

2.3.1 Setting Algorithms

Dijkstra's algorithm. Dijkstra's algorithm is the most known setting algorithm that works with positive weight arcs. In `Dijkstra's algorithm`, the principle is to select a node with the minimum weight at each iteration. It scans

each node at most once. That leads to a complexity of $\mathcal{O}(n^2)$ as time bound in the worst case (Ahuja et al., 1993) where n is the number of nodes. Two sets are maintained, a permanently set that represents selected nodes and temporary set that designates nodes not yet selected. The algorithm performs the node selections operation n times. Each operation requires that it scans each temporarily labeled node which leads to $\mathcal{O}(n^2)$. Besides, the algorithm performs the distance updates operation for all outgoing arcs of a node v. Overall, the algorithm requires $\mathcal{O}(m)$ since each operation requires, a constant time, $\mathcal{O}(1)$. At the end, `Dijkstra's algorithm` solves the shortest path problem in $\mathcal{O}(n^2)$.

There are many versions of `Dijkstra's algorithm` with the aim of improving this time bound by trying different data structures and several implementations of the algorithm (Ahuja et al., 1993). Using an adjacency list to represent the graph can reduce the bound to $\mathcal{O}((n+m)\times\log n)$. The idea is to traverse all vertices of the graph using `BFS (Breadth-first search)` in $\mathcal{O}(n+m)$ and use a heap for labeled nodes to extract the min in $\mathcal{O}(\log n)$ time. Moreover, using `Fibonacci heaps` to extract the minimum weight can reduce the running time to $\mathcal{O}(m+n\times\log n)$ (Fredman and Tarjan, 1987).

Bidirectional Dijkstra. In some applications of the shortest path problem, we want uniquely to determine the shortest path between two nodes. `Bidirectional Dijkstra's algorithm` solves the problem of finding the shortest path between two nodes faster since it eliminates some unnecessary computations by reducing the number of visited vertices in practice. In `Bidirectional Dijkstra's algorithm`, we apply `Dijkstra's algorithm` between origin node (forward search) and destination node (backward search) at the same time and stop when the shortest path is found. That is, some nodes w has been processed, *i.e.*, deleted from the queue of both searches (Ahuja et al., 1993).

A* search. The A* search is an acceleration algorithm of `Dijkstra's algorithm` in the sense that it preferably settles nodes that are closer to the destination when finding the shortest path between two nodes. The algorithm is based on heuristic function $h(v)$ that estimates the weight of the cheapest path from v to the destination. Then, at each step, it selects a labeled node v with the smallest weight, defined as $l(v) = d(v) + h(v)$. In transportation networks, the classical way to use A* estimates $h(v)$ based on the Euclidean distance between these two nodes (Delling et al., 2009b). It is easy to see that A* search is equivalent to `Dijkstra's algorithm` when $h(v)$ is zero.

2.3.2 Correcting Algorithms

The `Bellman-Ford-Moore` algorithm is known as a correcting shortest path algorithm. It achieves the best currently known bound of time with negative weight arcs $\mathcal{O}(nm)$ where m is the number of edges. The algorithm maintains the set of labeled nodes in a FIFO queue and allows detecting negative cycle in a weighted directed graph. In Bellman-Ford, arcs are considered one by one. The next node to be scanned is removed from the head of the queue; a node that becomes labeled is added to the tail of the queue. The algorithm performs at most $n-1$ passes through arcs. Since each pass requires $\mathcal{O}(1)$ computations for each arc, this implies $\mathcal{O}(nm)$ time bound for the algorithm. `Bellman-Ford-Moore` is qualified as a robust algorithm since there is no priority queue. Some heuristics have been introduced to improve the practice performance of the algorithm. For instance, (Cherkassky et al., 1996) introduce a `parent checking heuristic` that scans a node only if its parent is not in decrease.

(Cherkassky et al., 1996) provide an extensive computation study of shortest path algorithms with theoretical explanations and experimental results in function of different instances of various problems.

3 RELATED WORK

In this section we give a review literature about air scheduling development optimization problems with focus on the route network development problem that presents the scope of our study. Then, we introduce the main approaches to model transportation networks.

3.1 Air Scheduling Development Problems

The air scheduling development problem has been broken, in practice, into several subproblems (Barnhart and Cohn, 2004). This is due to its very large-scale nature. Thereby, the route network development, schedule design, fleet assignment, aircraft routing, and crew scheduling are

the five facets of the air scheduling development optimization problems. The airline has to resolve these problems before beginning an operation (Rebetanety, 2006):

Route network development : deciding which set of origin-destination pairs to serve.

Schedule design : defining the frequency of each flight. Scheduling determines where and when the airline will fly.

Fleet assignment : specifying the type and the size of aircraft serving each flight in a given schedule.

Aircraft routing : determining feasible aircraft routes under maintenance and time constraints.

Crew scheduling : assigning crews to flights.

Route network development. The route network development is a key element of an airline process. Given a fleet plan that determines the constraints of each aircraft, the route network development consists in determining the set of routes to be flown: airline manager should select which route to add or delete (Hall, 2012). Adding a new flight to the current network or creating a new route on the network is a complicated process that has some impacts notably on the flight schedules. Schedules have to be chosen according to all flights connected with the new flight but either to those of competitors.

Route network development approaches. The most common approach is route profitability models that have the objective to allow airlines to select routes that maximize total airline profits, given a set of candidate routes and expected demands, subject to fleet constraints. Such models are designed to offer the airline with the economic impacts of adding a new route to its network. The accuracy of route profitability models depends on the accuracy of the inputs and the nature of assumptions, typically, demand, revenue, operating cost estimates and assumptions concerning expected market shares. The evaluation of the market share, that the airline can gain of the total demand, uses a QSI models or a logic-based model (see (Jacobs et al., 2012) for more information). Nowadays, the QSI models are more used by airlines due to the complexity of maintaining parameter estimates in logic-based model.

A second approach has been given by (Sha et al., 2015) who proposed a discrete-choice model using random-utility theory. The utility function is a linear function of decision variables with interaction effects. The preferences for each variable are estimated using historical datasets. The decision of route selection is then modeled using a binary choice model derived from the utility function. The advantage of this approach is that it takes into account the uncertainty constraints. However, the approach requires historical data to build the model.

A third approach is the network design models, that are addressed to design the network of an airline. On the route network development, such models are used to plan routes in the network. Basically, two sets of decision variables designed are: one assigning supply to the network and an other assigning demand to the network. This approach models the problem as integer programming problems with an objective function of maximizing the profit (Barnhart and Cohn, 2004).

The FRP is related to the route network development problem. It comes before applying the route network development models cited above. Given a set of candidate routes passing through the specific flight, the problem consists in selecting the subnetwork that contains interesting routes regarding the regret constraint. Therefore, instead of applying route profitability models to the whole network. It would be applied only on important routes respecting the preferences modeled by the regret constraint. Normally, it will help to speed-up QSI computations on the application.

(Idrissi et al., 2017) formulated the FRP as finding a maximal sub-graph that for each node there exists a supported path by the regret constraint (see section 5). Given a certain regret, we are searching paths that are neither longer nor more expensive than the shortest path. The regret is defined for each time and cost criteria. In literature, this kind of problem consists to find Pareto paths which can be found in polynomial time using lexicographic order (Gandibleux et al., 2006). However, we are looking for paths satisfying the time or distance or cost criterion which is a sufficient condition to solve the FRP.

3.2 Modelling Transportation Networks

There are mainly two approaches in the literature to model transportation networks: the time-independent and the time-dependent. The time-independent is mostly used in road networks where each arc is assigned a constant value which

may be travel time, distance or any other metrics that we would like to minimize. Such model is more simpler since it excludes time-dependencies. For this reason, a lot of research focused toward accelerating shortest path algorithms on time-independent models (Pajor, 2009). The time-dependent model is an efficient and compact approach to model time-dependencies. In the case of flight networks, a node is introduced for each airport and an edge is inserted if there is a direct connection between two airports. At each edge, we store a function rather than a constant value. This function returns the arrival time at the destination airport according to the departure time from the source airport (Delling et al., 2008). This model yields to small graph with realistic travel time. However, transfer time between airports are not included.

To overcome this issue, the time-expanded model was proposed to included transfer times. In the graph, each node is an event of the timetable and an edge connects two consecutive events. Thereby, this approach yields to a huge graph since that it includes all time-dependent information.

The time-independent model lets to a condensed graph where an edge corresponds to all aggregated connections between a pair of nodes. In this kind of model, an airport is represented by a single node rather than multiple nodes per airport. In our study, we focus on the condensed model which yields a very small graph and represents the structure of the flight network adequately. Then, we obtain a lower bound regarding the travel time between two airports.

4 CONDENSED FLIGHT NETWORK

4.1 Modelling

First, we define the condensed graph. It is generated from the flight timetable where nodes represent airports meanwhile the presence of an arc indicates that there exists at least one elementary connection between two airports. In such graph, we omit time scheduling and keep only the transfer time. Then, each arc is constructed by aggregating all elementary connections between each pair of airports. Let $\mathcal{C}_{od} = \{c \in \mathcal{C} \mid o = o(c) \wedge d = d(c)\}$ be the set of elementary connections between two airports o & d. The following labels are associated with the arc (o, d) (Idrissi et al., 2017):

- $F_{od} = |\mathcal{C}_{od}|$ is the number of elementary connections between o and d;
- $C_{od} = \sum_{c \in \mathcal{C}_{od}} cap(c)$ is the total capacity filled;
- $P_{od} = \sum_{c \in \mathcal{C}_{od}} pax(c)$ is the total number of passengers;
- $R_{od} = \sum_{c \in \mathcal{C}_{od}} r(c)$ is the total revenue;
- $\bar{R}_{od} = \min_{c \in \mathcal{C}_{od}} \frac{r(c)}{pax(c)}$ is the minimum revenue per passenger;
- $T_{od} = \min_{c \in \mathcal{C}_{od}} t(c)$ is the minimum flight duration;
- D_{od} is the distance between the two airports.

The cost, time, and distance criteria are represented respectively by $\bar{R}_{od}$, T_{od}, and D_{od}. To take into account the transfer time, we propose to represent it by an arc in the graph. This technique is often used to model the information about transfer since it is important in computing shortest paths (Delling et al., 2009a). The model consists in introducing for each airport node $A \in \mathcal{A}$ a terminal node. Then, we insert two more nodes per airport: a departure node to model flight departures and an arrival node to represent flight arrivals. In this way, we can model the fact that all flights either begin or end at the airport following the flight timetable. Then, we introduce three different types of arcs. `board_at` is inserted from an airport to departure node, and finally a `connect_to` to model the transfer time between an arrival node and departure node of the same airport with a transfer time (see Figure 2). The graph contains four airports: `NCE, BKK, PEK, ICN` and four flight arcs referenced by `year_month`. Nodes in thin style represent departures (origin nodes), dashed nodes for arrival nodes (destination nodes). Double arcs is transferring time meanwhile bold arcs model flight time. Besides, dotted arcs for arrivals and dashed arcs for departures.

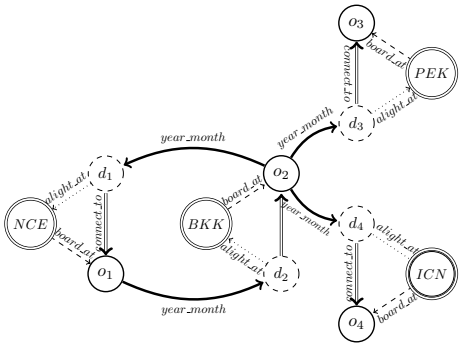

Figure 2: Model of condensed flight network. Source (Idrissi et al., 2018)

4.2 Graph Database Structure

The condensed flight network (CFN) is generated from a `NoSQL` database and stored in `Neo4j` graph database using a time-independent approach (Neo Technology, 2017). It is one of the most popular graph databases where queries can easily be expressed through *Cypher* query language. `Neo4j` is used for many applications, typically recommendation systems and complex networks like transportation network. `Neo4j` graph database follows the property graph model to store and manage its data. In `Neo4j`, data are represented in nodes, relationships, and properties or attributes. Both nodes and relationships contain properties (Robinson et al., 2015). A relationship connects a pair of nodes, it has a direction, a type, a start node, and an end node. `Neo4j` graph database is proved suited for the shortest path calculation for transport purposes (Miler et al., 2014).

Shortest path algorithms in graph database. `Neo4j` is an efficient graph database for performing graph traversal like shortest path computations (Holzschuher and Peinl, 2014). (Miler et al., 2014) compare the performance of `Dijkstra's algorithm` in transportation network stored in a relational database and stored in a graph database. Results show that `Neo4j` outperforms other databases. However, the graph database consumes a lot of memory when dealing with large transportation networks. In our study, we are working on the condensed flight network which is not a large network. Thus, the memory consumption of `Neo4j` is not an issue.

Besides, `Neo4j` proposed *APOC* (Awesome Procedures on Cypher) as stored procedures that regroup a list of procedures. Graph algorithms are part of these procedures namely some shortest path algorithms (Neo Technology, 2017). `Bidirectional Dijkstra's algorithm` is one of these graph search algorithms that solves the single-source shortest path problem for a weighted graph (see section 2.3). In addition, the graph database `Neo4j` offers the possibility to implement algorithms as user defined procedures to call in *Cypher* query. That is can be easy to use it by the final user. Therefore, graph database is the right choice to store the condensed flight network and then calculate the shortest path.

4.3 Implementation

Existing database. Currently, we are working on a real-world database that used `MongoDB`. Data are collected and queried monthly then it makes sense to create a relationship per period which is a month of the year. The relationship represents flight information (see Figure 2). We dispose of historical data about the last fifteen years. The current database stored data in a disconnected way and did not use a graph structure.

Figure 2 in section 4.1 illustrates the condensed graph in `Neo4j`. It represents the motivation example of the figure 1 modeled in `Neo4j`.

When testing the shortest path algorithm, it assumes that all nodes are reachable from the source. We use a fictive source and arcs to get a connected graph since some real-world data are missing. Hence, we introduce a connector node to connect each origin node to destination nodes. That leads to good results when the graph is connected. Let's n the order of the original graph and m its size, the generated graph has $n' = 3 * n + 1$ nodes and $m' = m + 5 * n$ arcs.

The condensed graph was generated for two years and has 4,577 nodes and 1,125,418 relationships of all types. Thus, the transformed condensed graph has 13,732 nodes and 1,148,303 relationships.

5 PROBLEM FORMULATION

This section outlines the formulation of the flight radius problem defined and also some of its properties.

5.1 Formulation

The flight radius problem consists in retrieving only relevant routes passing through a specific

flight, and satisfying the regret constraint. The considered flight is represented by an arc (o,d) in the graph (CFN) where $o,d \in V$. Then, we are interested in retrieving paths passing by the arc (o,d) that could be relevant regarding the regret defined for the time, distance, and cost criteria. More precisely, travelling from $o_1 \in V$ to $d_1 \in V$ by passing through the arc (o,d) is interesting if and only if the path $\{o_1,..,o,d,..,d_1\}$ between o_1 and d_1 satisfied the regret constraint. The satisfaction of the constraint depends on the shortest path between o_1 and d_1.

Our goal is to find alternative paths that are slightly longer or cheaper than the shortest path but passing by the arc (o,d). Let R be a Boolean regret constraint defined on paths of the graph G. Therefore, the problem consists in finding a maximal subgraph, in terms of nodes, such that each node supports a path satisfied by the regret constraint R. It means that there exists a path connecting nodes of this sub-graph passing through the arc (o,d) satisfying the time or distance or cost criteria. Such paths are called valid paths.

The problem is formulated as follows:

Input: *a graph $G=(V,E)$, the arc (o,d), and the regret constraint R*
Output: *a maximal subgraph $G'=(V',E')$ of G that each node belongs to a path passing through the arc (o,d) and satisfying the regret constraint.*

In this paper, we use the regret constraint R to identify which paths are supported. Let's define what the regret constraint R is. Let $w(i,j)$ be the weight of the arc (i,j) and let $l^\star(i,j)$ be the length of the shortest path from i to j. Let $l(i,j)$ be the length of a path passing through the arc (o,d), and let consider the following regret constraint defined for each criterion:

$$R_{od}(i,j) = l(i,j) \leq l^\star(i,j) + K \qquad (1)$$

Where $K \geq 0$. Each node must support at least a valid path. Then, we are looking for retrieving paths that satisfied at least one criterion.

5.2 Properties

The flight radius problem consists in finding valid paths. These paths depend on finding shortest path. Most traditional path finding are based on shortest path finding:

$$l(i,j) \geq l^\star(i,o) + w(o,d) + l^\star(d,j) \qquad (2)$$

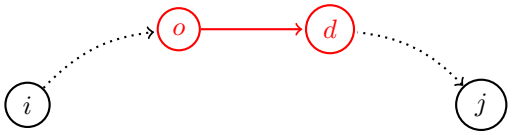

Figure 3: Decomposition of path. The red arc represents the studied arc (o,d) while the dotted arc is the shortest path.

Property. The subpath of a valid path is also a valid path.

The shortest path satisfies the triangle inequality property. Then, following the shortest path from i to o, passing by the arc (o,d), and then following the shortest path from d to j is always a valid path if it exists (see Figure 3).

Proof.

$$\begin{aligned} l^\star(i,o) + w(o,d) + l^\star(d,j) &\leq l^\star(i,j) + K \\ \cancel{l^\star(i,o)} + w(o,d) + l^\star(d,j) &\leq \cancel{l^\star(i,o)} + l^\star(o,j) + K \\ w(o,d) + l^\star(d,j) &\leq l^\star(o,j) + K \\ \overrightarrow{R_{od}}(j) = l(o,j) &\leq l^\star(o,j) + K \end{aligned} \qquad (3)$$

Following to the inequality 3, the subpath from o to j of a valid path is also valid. In addition, the subpath from i to d is valid.

$$\begin{aligned} l^\star(i,o) + w(o,d) + l^\star(d,j) &\leq l^\star(i,j) + K \\ l^\star(i,o) + w(o,d) + \cancel{l^\star(d,j)} &\leq l^\star(i,d) + \cancel{l^\star(d,j)} + K \\ l^\star(i,o) + w(o,d) &\leq l^\star(i,d) + K \\ \overleftarrow{R_{od}}(i) = l(i,d) &\leq l^\star(i,d) + K \end{aligned} \qquad (4)$$

Finally, search can be restricted to shortest valid paths starting from o or ending at d. Note that when $K=0$, the set of valid paths represents all the shortest paths passing by the arc (o,d).

Lemma 1. *Let p be a valid path, all the nodes belong to G'. For any shortest path p from d to i in G. If the arc (o,d) belongs to p, then it is a valid path and consequently i is going in the subgraph G'.*

The subpath of the shortest path is also a shortest path (Ahuja et al., 1993). Consequently, nodes j represent supported nodes: nodes that paths satisfy the regret constraint.

Solving the subproblem of finding the subpath from o to each node j is similar to find the subpath from each i to d in the reverse graph.

6 ALGORITHMS

In this section, we propose four algorithms: two algorithms that decompose the FRP in shortest path problems (SPP) and solved them in parallel using either the Dijkstra or Bellman algorithm; the third algorithm extends the Dijkstra algorithm to avoid useless computations; the fourth algorithm extends the Bellman algorithm to compute all shortest paths for all criteria at once. These algorithms perform parallel computations when it is possible.

All algorithms have the same input and output.

input : a CFN $G = (V, E)$, an arc (o, d), the criteria C
output: the set of supported nodes S

For sake of simplicity, the algorithm simply returns the set of supported nodes and does worry about storing the parents of the supported nodes. In practice, the algorithm also returns the union of the supported trees for each direction and for each criterion. Based on the regret functions, let's define:

$$R(i, dir) = \begin{cases} R_{od}(o, j), & \text{if } (dir = out) \\ R_{od}(j, d), & \text{if } (dir = in) \end{cases}$$

6.1 Shortest Path Decomposition

The shortest path decomposition solve two shortest path problems, one from the origin o and the other from the destination d, for each direction and for each criterion in parallel.Then the supported nodes are computed by checking the regret constraint.

Algorithm 1: SP Decomposition

```
foreach dir ∈ {in, out} in parallel
    foreach c ∈ C in parallel
        ShortestPaths(o, c, dir);
        ShortestPaths(d, c, dir);
        foreach i ∈ V do
            if R(i, dir) then
                S ← S ∪ {i} ;
            end
        end
    end
end
return S
```

The shortest path subroutine is either the Dijkstra or the Bellman algorithm. The computation of the shortest paths from the origin and from the destination is sequential so that finding the supported nodes is made easier.

6.2 Flight Radius Algorithms

Here, we design variants of the Dijkstra and Bellman algorithm tailored for solving the flight radius problem.

6.2.1 Dijkstra FR Algorithm

We present an algorithm that compute the shortest path from o, and then computes lazily the shortest path from d by skipping unsupported nodes. Here, the criteria are processed sequentially. At each iteration, the algorithm scans the node with the minimum weight and then relax its neighbors. So, we check if the node satisfies the regret function otherwise we skip the node. Therefore, we skip non supported nodes that cannot be extended into valid paths. The algorithm ends if the queue becomes empty or if all non supported nodes have been closed.

Algorithm 2: Dijkstra FR Algorithm

```
foreach dir ∈ {in, out} in parallel
    foreach c ∈ C do
        Dijkstra(o, c, dir);
        enqueue(Q, d);
        while Q ≠ ∅ and isNotClosed() do
            i ← argmin_{j∈Q}(d[j]);
            dequeue(Q, i);
            if R(i, dir) then
                S ← S ∪ {i} ;
                SCAN (i,c,dir);
            end
        end
    end
end
return S
```

6.2.2 Bellman FR Algorithm

We propose a variant of the Bellman algorithm that computes at once all shortest paths from the origin and from destination for all criteria. Here, the `SCAN` function updates all paths from the origin and from the destination for all criteria.

Algorithm 3: Bellman FR Algorithm

```
foreach dir ∈ D in parallel
    enqueue (Q,o);
    enqueue (Q,d);
    while (Q ≠ ∅) do
        i ← dequeue(Q);
        SCAN (i,C,dir));
    end
    foreach i ∈ V do
        if R(i,dir) then
            S ← S ∪ {i} ;
        end
    end
end
return S
```

7 EXPERIMENTS

In this section, we evaluate the algorithms to solve the flight radius problem on real-world datasets. We are especially interested in comparing the runtimes of the algorithms which impact the user experience of `PlanetOptim`. Another interesting metric is the number of nodes scanned by the algorithm which determines the accessed property values (criteria) of a relation that incurs extra IO the first time those properties are accessed because the properties reside in a separate store file from the relationships (after that, however, they're cached) (Robinson et al., 2015). Our experimental protocol aims to answer the following questions:

1. Which algorithm is the fastest and, will, therefore provides the best user experience?
2. Is it worthwhile to design algorithms for flight radius problem compared to the decomposition into shortest path problems?
3. Does reduction of the number of scanned nodes provoke a reduction of the runtime ?
4. How does the performance evolve with the number of criteria ?

First, we present how the benchmarks instances have been generated. Second, the runtimes of the algorithms are compared, and then, the relation between runtimes and number of scanned nodes is studied. Last, we analyze the ratio of the flight radius algorithms over shortest path decompositions depending on the number of criteria.

All the experiments were led on a computer running on Ubuntu 16.04.5 with 32 GB of RAM and one Intel Core i7-3930K 3.20GHz processors (6 cores). The implementation is based on `Neo4j` and *APOC* version 3.2.0. All algorithm are implemented in Java 8.

7.1 Instances Generation

The `Neo4j` database contains historical data for the years 2016 and 2017 (24 year-months). Each year-month corresponds to a different graph. The condensed flight network contains 13,732 nodes and 1,148,303 arcs.

A benchmark instance must specify the year-month, the OD-pair (o, d), the number of criteria, and their regrets.

The number of criteria is one (time duration), two (time duration, distance), or three (time duration, distance, cost). For each criterion, two values are considered for its regret K: 0 and the median (over all relations and year-months). The value 0 means that only shortest paths passing through the arc (o, d) are valid, whereas many other paths are valid with the median. For instance, the median duration of a flight is approximately two hours. So a path between i and j passing through the arc (o, d) is valid if it does not exceed the duration of the shortest path between i and j by four hours (the median duration plus the minimum connection time).

For each year-month, each number of criteria, and each combination of criteria values, a few pairs of origin and destination (o, d) are drawn randomly. At the end, more than ten thousand instances have been tested.

7.2 Algorithms Comparison

Table 1 gives the average, standard deviation, and maximum runtime in milliseconds of the shortest path decompositions and of the flight radius algorithms using Bellman or Dijkstra. The Dijkstra variants are approximately two times faster and have a lower standard deviation than their Bellman counterparts. The algorithm based on Dijkstra is slightly faster than the decomposition whereas it is not the case for Bellman.

Figure 4 analyzes the relation between the runtime and the number of scanned nodes for each algorithm. Each point represents one instance and its x coordinate is the runtime in milliseconds, whereas its y coordinate is the number of scanned nodes. The color of a point indicates which algorithm solved the instance. The flight radius algorithm based on Bellman scans less nodes than the shortest path decomposition

	SP Decomposition		FR Algorithm	
	Dijkstra	Bellman	Dijkstra	Bellman
avg	266	500	231	604
std	60	198	90	282
max	418	1328	644	1612

Table 1: Distribution of runtimes given in milliseconds.

	Dijkstra		Bellman	
	Runtime	#Scans	Runtime	#Scans
1	0.69	0.57	1.04	0.81
2	0.79	0.41	1.15	0.57
3	1.09	0.43	1.37	0.49

Table 2: Improvements of the FR algorithms over the SP decompositions.

based on Bellman (red points are below the blue ones), but without reducing the runtimes (blue points are on the the left of the red ones). It means that the additional time spent to read all properties is not compensated by the decrease in the number of scanned nodes. For Dijkstra, the number of scanned nodes for the decomposition only depends on the number of criteria whereas it is not the case for the flight radius algorithm. In this case, a lower number of scanned nodes implies a reduction of the runtime because the number of read properties is also reduced (green points are on the left and below purple points). As expected, Dijkstra based algorithms scan less nodes than those based on Bellman.

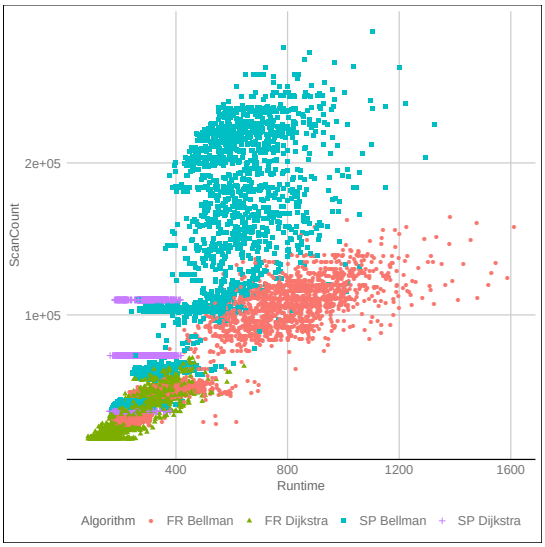

Figure 4: Analysis of the runtimes and scan counts.

Last, Table 2 gives the geometric mean of the improvements provided by the FR algorithms over the SP decompositions in terms of runtimes and number of scanned nodes. The improvement is the ratio of the runtime of the FR algorithm over the SP decomposition for Bellman or Dijkstra. The improvement is lower than 1 if the FR algorithm is better than the SP decomposition and greater than 1 otherwise. Both FR algorithms based on Dijkstra or Bellman reduces the number of scanned nodes and the reduction increases with the number of criteria. The runtimes are not reduced for Bellman, but are also reduced for Dijkstra when the number of criteria is one or two. When there are three criteria, the reduction of scanned nodes does not compensate for the additional parallelization of the decomposition.

To conclude, the FR algorithms based on Dijkstra is the most efficient, and satisfy the real-time constraint of `PlanetOptim`. Beside, the runtimes of the algorithms increase with the number of criteria in spite of the parallelization. When there are three criteria, the reduction of the number of scanned nodes does not help to reduce the runtimes of the FR algorithms based on Dijkstra, a perspective is to parallelize along the criteria.

8 CONCLUSION

This work presents the algorithm of solving the flight radius problem. We propose to compare four algorithms: two algorithms that decompose the FRP in shortest path problems (SPP) and solved them in parallel using either the Dijkstra or Bellman algorithm; the third algorithm extends the Dijkstra algorithm; the fourth algorithm extends to the Bellman algorithm to compute all shortest paths from both origin and destination for all criteria at once. The experimental evaluation demonstrates that the FR algorithms based on Dijkstra is the most efficient, and satisfy the real-time constraint of `PlanetOptim`. Besides, the FR algorithms based on Dijkstra can be enhanced by following the same parallel processing of the SP decomposition. The next step of the study is to include QSI computations. We hope that the flight radius solution accelerates the computations by working on the small graph rather than the entire graph with uninteresting routes.

REFERENCES

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Prentice hall.

Barnhart, C. and Cohn, A. (2004). Airline schedule planning: Accomplishments and opportunities. *Manufacturing & service operations management*, 6(1):3–22.

Cherkassky, B. V., Goldberg, A. V., and Radzik, T. (1996). Shortest paths algorithms: Theory and experimental evaluation. *Mathematical programming*, 73(2):129–174.

Delling, D., Giannakopoulou, K., Wagner, D., and Zaroliagis, C. (2008). Timetable information updating in case of delays: Modeling issues. *Arrival Technical Report*.

Delling, D., Pajor, T., Wagner, D., and Zaroliagis, C. (2009a). Efficient Route Planning in Flight Networks. *ATMOS*, 12.

Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2009b). Engineering route planning algorithms. In *Algorithmics of large and complex networks*, pages 117–139. Springer.

Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615.

Gandibleux, X., Beugnies, F., and Randriamasy, S. (2006). Martins' algorithm revisited for multi-objective shortest path problems with a maxmin cost function. *4OR: A Quarterly Journal of Operations Research*, 4(1):47–59.

Hall, R. (2012). *Handbook of transportation science*, volume 23. Springer Science & Business Media.

Holzschuher, F. and Peinl, R. (2014). Performance optimization for querying social network data. In *EDBT/ICDT Workshops*, pages 232–239.

Idrissi, A. K., Malapert, A., and Jolin, R. (2017). The route network development problem based on qsi models. In *International Conference on Operations Research and Enterprise Systems (ICORES 2017)*, pages 3–11.

Idrissi, A. K., Malapert, A., and Jolin, R. (2018). Solving the flight radius problem. In *Proceedings of the 7th International Conference on Operations Research and Enterprise Systems - Volume 1: ICORES,*, pages 304–311. INSTICC, SciTePress.

Jacobs, T. L., Garrow, L. A., Lohatepanont, M., Koppelman, F. S., Coldren, G. M., and Purnomo, H. (2012). Airline planning and schedule development. In *Quantitative Problem Solving Methods in the Airline Industry*, pages 35–99. Springer.

Miler, M., Medak, D., and Odobašić, D. (2014). The shortest path algorithm performance comparison in graph and relational database on a transportation network. *Promet-Traffic&Transportation*, 26(1):75–82.

Neo Technology (2017). Neo4j. `https://www.neo4j.com`.

Pajor, T. (2009). *Multi-modal route planning*. PhD thesis, Universitat Karlsruhe (TH) - Institut fÃ$\frac{1}{4}$r Theoretische Informatik (ITI).

Rebetanety, A. (2006). *Airline schedule planning integrated flight schedule design and product line design. University Karlsruhe (TH)*. PhD thesis, PhD thesis, 2006. Available at http://www. iks. kit. edu/fileadmin/User/calmet/stdip/diprabentanety. pdf. Accessed 2013 January 30.

Robinson, I., Webber, J., and Eifrem, E. (2015). *Graph databases: new opportunities for connected data.* " O'Reilly Media, Inc.".

Sha, Z., Moolchandani, K., Maheshwari, A., Thekinen, J., Panchal, J. H., and DeLaurentis, D. A. (2015). Modeling airline decisions on route planning using discrete choice models. In *15th AIAA Aviation Technology, Integration, and Operations Conference*, page 2438.

APPENDIX

We would like to thank Carine Fedele $< Carine.Fedele@unice.fr >$ for its insightful comments on the paper, as these comments led us to an improvement of the work.