Capgemini engineering

HOW TO MANAGE A FLEET OF FLYING TAXIS IN A DYNAMIC ENVIRONMENT

FINAL DELIVERABLE

ND2I Project
Deliverable Document

SANA IKLI
December 31, 2021

"Knowledge is wasted when it isn't shared."

Abstract

abstract here...
abstract here...
abstract here...
abstract here...
abstract here...
abstract here...
abstract here...

1 Introduction

Flying taxis are expected to serve as an alternative to ground transportation to alleviate traffic congestion in metropolitan cities. Several transportation pioneers and airline manufacturers are preparing to launch their Urban Air Mobility (UAM) services in the near future. Indeed, Airbus Helicopters is currently working on new electric flying taxis as a part of the *CityAirbus Nextgen* project [3]. Their flying taxis are expected to operate in 2025. Apart from Airbus, Uber is also working on an air taxi service called *Uber Elevate*, which is estimated to lunch in 2023 [10].

Due to the dynamic nature of the problem, the companies proposing air taxi service will deal with several real-time decisions. Such decisions include (i) evaluating different candidate trips and scheduling the flying taxis so as to optimize a given objective (*e.g.*, reduce the operational costs, serve the maximum number of demands, etc.), (ii) dynamic estimation of the market demands, and (iii) battery charging management as well as other maintenance related-issues [9].

In this work, we are interested in the real-time management of a fleet of flying taxis. The management includes the dynamic scheduling of the air taxis and the battery charging handling. This work is one of the few that considers the (more realistic) real-time problem, where decisions have to be made dynamically to accommodate the new changes in the air taxi environment.

A common strategy used in the literature to tackle the dynamic scheduling is the *Rolling Horizon* (RH) approach. The latter usually subdivides the scheduling horizon into smaller sub-horizons, and then it solves the static problem on each sub-horizon. These strategies are used in the literature to tackle (the dynamic version of) two well-know optimization problem that are *similar* to our problem, namely: the Job-Shop Scheduling Problem (JSSP) and the Vehicle Routing Problem (VRP).

The analogy between a JSSP and the scheduling of air taxis can be described as follows. The machines represent the flying taxis, and the jobs represent the demands. The trip duration of a given demand can be interpreted as the processing time of a job in the JSSP framework. A fundamental aspect of the flying taxis framework is the battery charging; the latter can be translated to the JSSP model as the machine breakdown, *i.e.,* a specified period of time during which a machine is is out of order.

In addition, the scheduling of air taxis is clearly similar to the VRP. In the two frameworks, we have vehicles to dispatch and clients to serve. The battery charging of an air taxi is translated to the VRP framework as the vehicle refueling.

As a consequence of these analogies, we can derive two conclusions. On the one hand, the problem of scheduling flying taxis, the JSSP, and the VRP have the same complexity: they are NP-hard problems (JSSP and the VRP are already proven to be NP-hard [7, 11]). On the other hand, we can use the literature of the dynamic JSSP and the VRP to gain insights on how the real-time problems are handled. The latter is the objective of the next section.

2 Literature review: Dynamic scheduling techniques

Rolling horizon approaches are common strategies in the literature to solve the dynamic scheduling problems. They generally consist in subdividing the planing horizon into smaller sub-horizons. Then solve the (static) problem on each sub-horizon sequentially, as illustrated in Figure 3. These approaches represent valid solution methods to tackle the dynamic case, because their computation times are usually suitable for real-time applications.

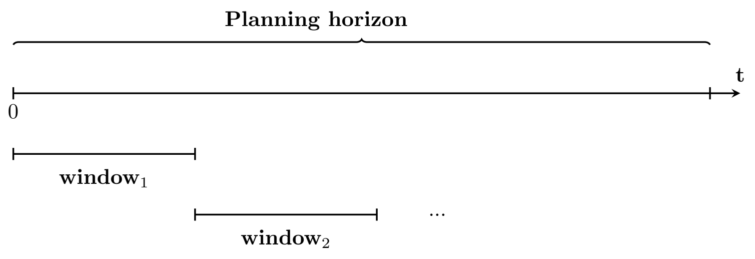

Figure 1: Illustration of the rolling-horizon approach.

In the next subsection we review the most relevant RH approaches for three optimization problems: the job-shop scheduling, the vehicle routing problem, and the unmanned ariel vehicle scheduling problem.

2.1 Job-Shop Scheduling

In the RH approaches for the Job-Shop Scheduling Problem (JSSP), there are jobs that will start inside a sub-horizon and finish outside that horizon, regardless of its length. This kind of jobs is called *cross-window* jobs. To deal with them, boundary conditions have to be defined for each sub-horizon. On the other hand, the computation complexity is expected to be reduced in the sub-horizons, since the problem size is smaller. The RH scheduling approaches can be categorized into two types of strategies:

- The *event-driven* scheduling in which the rolling horizon is a job window, *i.e.*, a number of jobs for scheduling and processing. This strategy first chooses a job window, which is defined by the (allowed) maximum number of jobs. Then, it divides the set of jobs in three sub-sets: (i) the sub-set of available jobs, (ii) the sub-set of current jobs, and (iii) the sub-set of finished jobs.

  A selection rule is then used to select jobs from the sub-set of available jobs. Finally, an optimization algorithm is required to solve the problem in the job windows, and the three above-mentioned sets are updated until all the jobs are processed. Notable examples from the literature that use this approach are [1, 4]. The former work used this strategy to solve the online workflow scheduling on cloud environment, and the latter used it for the classical dynamic JSSP.

- The *periodic scheduling* strategies in which the rolling domain is a *time window*, *i.e.*, a time slot on the scheduling horizon. This strategy is used in [12] to solve the dynamic JSSP. In this work, the authors subdivide the planning horizon, T, into two non-overlapping windows, T_1 and T_2. Then, they solve the (static) problem on T_2. Finally, they determine the set of cross-window jobs and schedules them with the jobs in the window T_1.

  The periodic scheduling is also used in [13], but with a different *rolling time-window*. In this context, the planning horizon, denoted $[0, K]$, is subdivided in R time periods of equal lengths. Then, the problem is solved in the whole planing horizon at the beginning. As time progresses, the window is shortened by one time period, and the problem is again solved in the new window, as illustrated in Fig. 2.

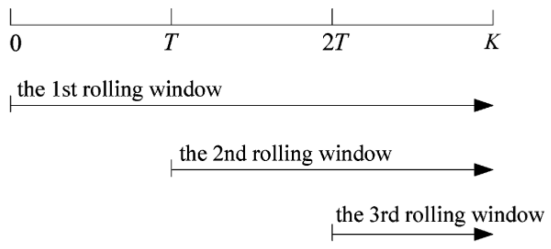

Figure 2: A rolling horizon with three time-windows (source: [13]).

2.2 Vehicle Routing Problem

In the dynamic VRP, the customer orders are usually considered to be the dynamic events. In the work of Hanshar and Ombuki-Berman [5], the RH periodic scheduling strategy is used, but with time-windows that differs from [12] and [13]. Indeed, in [5] , the authors subdivide the planing horizons into several slices of equal lengths. Then, they sequentially solve the (static) VRP on each slice. Furthermore, the authors define a cutoff time, which postpones the late orders to the next day. The genetic algorithm is then used to solve the static problem on each time slice.

A number of heuristics are proposed in [6] to solve the dynamic VRP:

- First-Come First-Served (FCFS) that serves the clients in the order which they are. A special case of this strategy is:
- Stochastic Queue Median (SQM) in which the vehicle travels directly from the median of the service region to the next demand location. After the service has been completed, the server returns to the median and waits for the next demand.
- Nearest Neighbor (NN) that completes the demands in one location, and then travels to the nearest neighboring demand.

2.3 Unnamed Ariel Vehicle Scheduling

The works in the literature addressing the real-time scheduling and routing of air taxis are rare compared to the machine scheduling or the VRP. Nonetheless, the two recent works [8] and [11] consider this dynamic scheduling for two types of air vehicles: the flying taxis for the former work, and the Unmanned Aerial Vehicle (UAV) for the latter.

In[11], the authors develop a real-time tool to manage a fleet of UAVs to serve customers. The management includes visiting the service station for battery recharging. The problem is formulated as a mixed-integer program, inspired from the VRP literature, and solved using *CPLEX* [2]. The dynamic scheduling is tackled using the *event-driven* and the *periodic scheduling* strategies explained in section 2.1.

In the very recent work of [8], the author considers the real-time dispatching of air taxis in a centralized taxi network. Two objectives are taken into account in the optimization model: minimizing the **number of idle taxis** and minimizing the **total travel time of air taxis at inactive state**. Since these two objectives are conflicting, the author uses a *goal programming* algorithm to solve the problem. The proposed strategy to handle the real-time demands is similar to the one used in [12]. To the best of our knowledge, this is the only work in the literature that considers the dynamic scheduling of flying taxi.

3 Dynamic flying taxi scheduling

The problem of dynamic flying-taxi scheduling consists in finding the optimal dispatching of a fleet of air taxis to serve a set of customers in real-time. In this context, some costumer requests as well as other problem parameters may not be known before the optimization starts. In the work of [14], a mixed-integer program was proposed to model this problem. Since this scheduling problem is NP-hard, exact methods may require long computation times to find an optimal solution. Thus, heuristic methods are more adapted to solve the real-time problem, where solutions must be computed in short times. For this reason, Panwadee [14] used a genetic algorithm to solve the (static) problem.

In this section, we present two classical heuristics used in the field of the VRP to dispatch ground vehicles, namely the FCFS and the NN. The dynamic adaptation of these heuristics consists in integrating them in a rolling horizon approach. We will adapt these two heuristics to use them to solve the dynamic scheduling of flying taxis. The RH approach we adopt is the *periodic scheduling* strategy. The rolling domain in this context is a *time-window*, *i.e.*, a time slot on the scheduling horizon (see Figure 3).

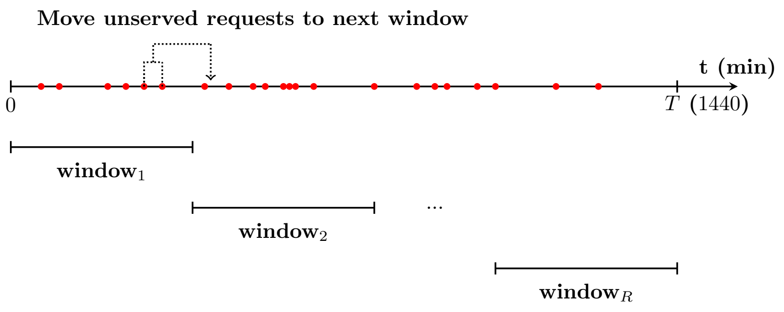

Figure 3: Rolling-horizon strategy with rolling time-windows.

In the context of our RH strategy, the (static) problem is solved in the first *rolling window*, using a static scheduling heuristic (FCFS, NN, of the genetic algorithm of [14]). At the end of this window, the RH strategy moves the unserved requests to the beginning of the next window, and schedules them together with the new available requests in this window, as explained in Algorithm 1.

The three scheduling heuristics that we use to solve the static problem inside each time window are explained in the next three sections.

3.1 First-Come, First-Served

The FCFS heuristic serves the requests according to the order given by their pick-up times. Indeed, the demands are sorted according to the non-decreasing order of their pick-up times. The advanced requests are

Pseudocode 1 Rolling-horizon algorithm

```
1. **initialize** problem parameters (available taxis, battery level, scheduling horizon, rolling-window length)
2. **initialize** served and unserved requests ($= \emptyset$)
3. **for** each rolling window **do**
        4. **compute** available requests
        5. **append** non-served requests to available requests
        6. **apply a static scheduling heuristic** to schedule the available requests
        7. **update** served and non-served requests
        8. **update** battery level
```

index	1	2	3	4	5	6
pick_t	803	53	1373	1319	1425	710
dur_t	26.1	19.17	30.01	18.89	19.77	23.89
ori_x	17182	5807	10958	12710	9890	10726
ori_y	5391	828	21818	2405	21282	13201
des_x	5753	10202	21534	20022	11893	21323
des_y	12415	7076	8931	3602	13390	17863

Table 1: Example of an instance with 6 requests.

served (without considering their locations) on their pick-up times or in an interval – defined by the user – centered around the pick-up time. The process continues until no request is available.

To illustrate how this heuristic works, let us consider an example. Suppose we have an instance with 2 available flying taxis and 6 requests, indexed by 1, 2, ..., 6. The pick-up times (`pick_t`) of each request is displayed in Table 1. The rows named "`dur_t`", "`ori_x`", "`ori_y`", "`des_x`", and "`des_y`"represent the duration, the x and y coordinates of the origin location of the request, and the x and y coordinates of the destination location, respectively.

The FCFS scheduler sorts the above-mentioned requests in the non-decreasing order of the pick-up times, then serves them in the resulting order. The battery level is checked before each customer pick-up, and updated after each customer drop-off. The solution provided by the FCFS scheduler for our example is showed in Figure 4. The green and orange bars illustrate the taxi service to serve the request and the battery recharging, respectively.

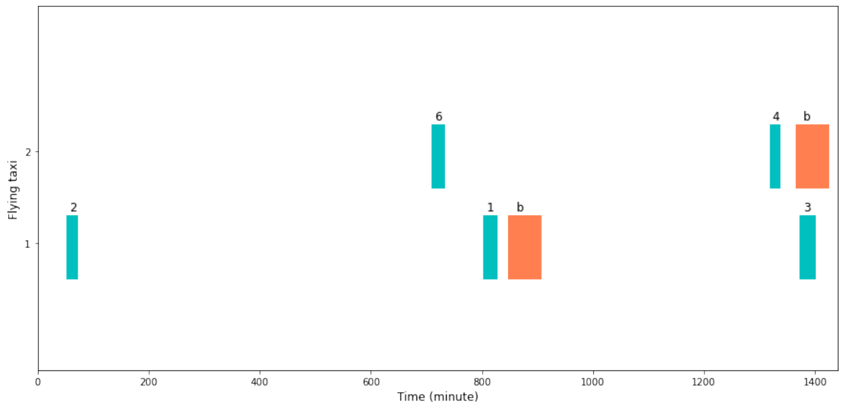

Figure 4: The FCFS heuristic solution.

3.2 Nearest Neighbor

The NN heuristic serves the closest requests to the current location of the taxi. Indeed, for each taxi, the NN scheduler serves first the closest demand to the center. After completing service at the location of the first demand, the taxi travels to the nearest neighboring demand and so forth. The key difference between a FCFS and a NN scheduler is that the former sorts the customers at the beginning of the scheduling and then serves them, while the latter needs to compute the distances from current request location to all other unserved requests, to serve the closest one. This process is repeated after each customer drop-off to find the next one to serve. Hence, this heuristic may lead to larger computation times than the FCFS.

If we consider the same example as in Section 3.1, the NN heuristic solution is given in Figure 5.

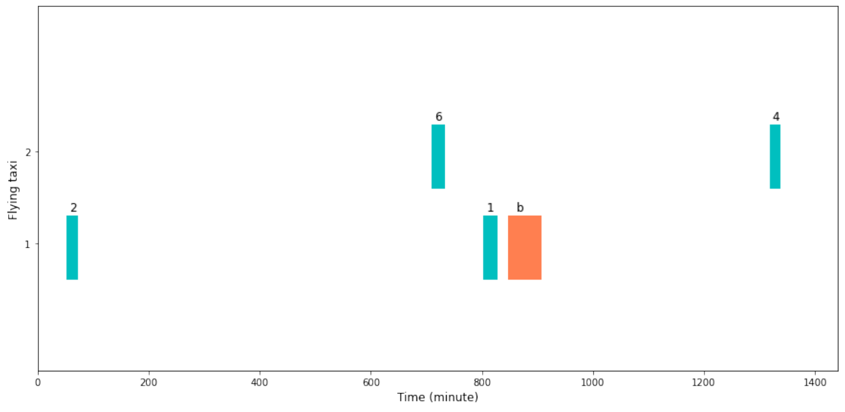

Figure 5: The NN heuristic solution.

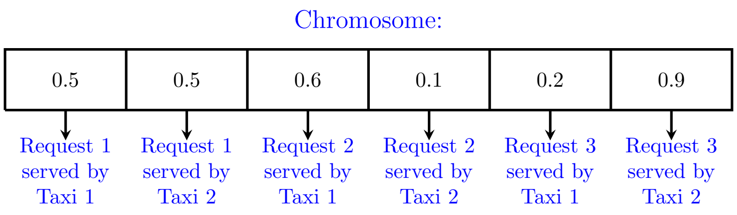
Chromosome:
0.5
0.5
0.6
0.1
0.2
0.9
Request 1 served by Taxi 1
Request 1 served by Taxi 2
Request 2 served by Taxi 1
Request 2 served by Taxi 2
Request 3 served by Taxi 1
Request 3 served by Taxi 2

Figure 6: Example of a chromosome with six genes (3 requests/2 taxis).

3.3 Genetic Algorithm

The third heuristic that we integrate in the RH approach is the GA of [14], that was proposed to schedule flying taxis for the static case. In this GA, the first population of chromosomes is randomly generated. Genetic operations such as mutations and crossovers are performed on this population, and best off-springs are selected to continue the process until the stopping criteria is satisfied. The latter corresponds to a number of iterations since the last improvement.

A chromosome is composed of genes whose values are randomly generated in the interval $[0, 1]$. Each gene represents a demand which is served by a flying taxi. Thus, the total number of genes in a chromosome is equal to the number of requests multiplied by the total number of taxis. For example, if we consider an instance with 3 requests and 2 flying taxis, the total number of genes is equal to 6 (see Figure 6)

4 Computational results and discussion

This section reports the computational results of implementing the above-mentioned heuristics, integrated in the RH approach. All experiments are run on a computer under Windows operating system, processor Intel(R) Core(TM) i5-10310U with 8 GB of RAM.

In section 4.1, we introduce new data-sets of instances that were generated based on [14]. Our new instances are however more congested and feature more realistic aspect of the problem. Then, in Section 4.2 we present computational results of implementing our two heuristics FCFS and NN, and the GA of [14], all integrated in a RH approach.

4.1 New generated instances

We randomly generate 10 test instances, based on the instance generator of [14]. In addiction to being more congested, our instances define a time window for each request during which it can be served, which is more realistic that imposing a strict pick-up time.

Table 2 summarizes some important characteristics of our generated instances. Throughout this table, the first, second and third columns present the name, the total number of requests, and the total number of air taxis in each instance (respectively). The fourth column "req/h" reports the average request per hour in each instance, which measures how dense the latter is. The remaining columns show the minimum, average, and the maximum duration of request trips in an instance.

Instance name	# req	#taxis	req/h	min duration	average duration	max duration
instance50_2	50	2	2.08	12.75	27.12	49.07
instance100_3	100	3	4.17	11.00	26.8	48.57
instance100_5	100	5	4.17	11.71	26.75	51.37
instance250_5	250	5	10.42	10.42	27.7	47.48
instance250_10	250	10	10.42	10.42	27.70	49.48
instance500_4	500	4	20.83	10.51	26.57	47.85
instance500_10	500	10	20.83	10.48	27.26	49.15
instance1000_9	1000	9	41.67	10.33	26.96	49.74
instance1000_15	1000	15	41.67	10.49	27.16	49.35
instance10000_20	10000	20	416.66	10.16	27.01	52.85

Table 2: Characteristics of the new constructed instances.

The data included in our instances contain the request identifier, the origin and destination locations of the requests and their pick-up times. Moreover, we define for each request a time window in which it can be served. The time window is defined by an earliest and a latest serving time, centered around the pick-up time.

4.2 Results and discussion

To compare the quality of the solutions provided by the above-mentioned heuristics, we define the following performance indicators:

- The **objective-value** that indicates the total service time (in minutes). In an optimization approach, this objective is maximized.
- The **non-profitable trips** duration that corresponds to the total travel time without passengers. This may occur when a taxi flies to the center to recharge its battery, or between two requests. In an optimization approach, this indicator should be minimized.
- The **CPU** time that indicates the computation times in seconds. The characteristics of the computer on which the experiments were run are provided at the beginning of Section 4.

Figure(reference here) shows...

5 Conclusion and future tracks of research

References

[1] Huangke Chen et al. “Real-time workflows oriented online scheduling in uncertain cloud environment”. In: *The Journal of Supercomputing* 73 (2017), pp. 4906–4922.

[2] IBM ILOG CPLEX. “V12. 1: User’s Manual for CPLEX”. In: *International Business Machines Corporation* 46 (2009), p. 157.

[3] Hugues-Olivier Dumez. *Toulouse. Un taxi volant pour circuler au-dessus des bouchons : le projet d’Airbus se concrétise.* `https://actu.fr/occitanie/toulouse_31555/toulouse-un-taxi-volant-pour-circuler-au-dessus-des-bouchons-le-projet-d-airbus-se-concretise_45082439.html`. 2021.

[4] Jian Fang and Yugeng Xi. “A rolling horizon job shop rescheduling strategy in the dynamic environment”. In: *The International Journal of Advanced Manufacturing Technology* 13 (1997), pp. 227–232.

[5] Franklin T. Hanshar and Beatrice Ombuki-Berman. “Dynamic vehicle routing using genetic algorithms”. In: *Applied Intelligence* 27.1 (2007), pp. 89–99.

[6] Allan Larsen, Oli B. G. Madsen, and Marius Solomon. “Partially dynamic vehicle routing—models and algorithms”. In: *Journal of the operational research society* 53 (2002), pp. 637–646.

[7] Jatoth Mohan, Krishnanand Lanka, and Neelakanteswara A. Rao. “A review of dynamic job shop scheduling techniques”. In: *Procedia Manufacturing* 30 (2019), pp. 34–39.

[8] Suchithra Rajendran. “Real-time dispatching of air taxis in metropolitan cities using a hybrid simulation goal programming algorithm”. In: *Expert Systems with Applications* 178 (2021), (13 pages).

[9] Suchithra Rajendran and Sharan Srinivas. “Air taxi service for urban mobility: A critical review of recent developments, future challenges, and opportunities”. In: *Transportation research part E: logistics and transportation review* 143 (2020), (20 pages).

[10] Suchithra Rajendran and Joshua Zack. “Insights on strategic air taxi network infrastructure locations using an iterative constrained clustering approach”. In: *Transportation Research Part E: Logistics and Transportation Review* 128 (2019), pp. 470–505.

[11] Byung Duk Song, Jonghoe Kim, and James R Morrison. “Rolling horizon path planning of an autonomous system of UAVs for persistent cooperative service: MILP formulation and efficient heuristics”. In: *Journal of Intelligent & Robotic Systems* 84 (2016), pp. 241–258.

[12] D. Sun and L. Lin. “A dynamic job shop scheduling framework: a backward approach”. In: *The International Journal of Production Research* 32 (1994), pp. 967–985.

[13] Lixin Tang, Shujun Jiang, and Jiyin Liu. “Rolling horizon approach for dynamic parallel machine scheduling problem with release times”. In: *Industrial & engineering chemistry research* 49 (2010), pp. 381–389.

[14] Panwadee Tangpattanakul and Ilhem Quenel. “Optimal Scheduling for Flying Taxi Operation”. In: *Proceedings of the 13th International Joint Conference on Computational Intelligence.* 2021, pp. 141–148.

A Abbreviations and Acronyms

EdC	Élément de Compréhension
FCFS	First-Come, First-Served
GA	Genetic Algorithm
JSSP	Job-Shop Scheduling Problem
NN	Nearest Neighbor
RH	Rolling Horizon
UAM	Urban Air Mobility
UAV	Unmanned Aerial Vehicle
VRP	Vehicle Routing Problem