A Rolling Horizon Approach for the Dynamic Scheduling of Flying Taxis

Sana Ikli
Hybrid Intelligence, Capgemini Engineering, 4 Avenue Didier Daurat, 31700 Blagnac, France

Keywords: Flying Taxis, Dynamic Scheduling, Genetic Algorithm, Heuristic Solutions, Rolling Horizon Approaches.

Abstract: Flying taxis are a promising alternative to ground transportation to alleviate the congestion problem in metropolitan cities. The launching of the first air taxis is expected in the next few years. The companies operating air taxi services will deal with several real-time problems. Such problems include the scheduling of flying taxi operations, and the battery charging management as well as other maintenance issues. In this work, we are interested in the *dynamic* scheduling of flying taxis, so as to serve a set of clients. This problem is on the one hand under-explored in the literature, as we will show in the next sections, and on the other hand, it is more realistic than the *static* case. We present in this work a *rolling-horizon* approach, integrated with three heuristics, to solve the dynamic scheduling of flying taxis. We also construct new realistic and difficult instances to test and validate our algorithms. Our instances and implementations are publicly available for the scientific community online. Finally, we perform a computational study on our generated instances to show the benefits and the limits of each heuristic.

1 INTRODUCTION

Air taxis are expected to serve as an alternative to ground transportation to alleviate traffic congestion in metropolitan cities (Rajendran and Zack, 2019). Several transportation pioneers and airline manufacturers are preparing to launch their Urban Air Mobility (UAM)[1] services in the near future. Indeed, Airbus Helicopters is currently working on new electric flying taxis as a part of the *CityAirbus Nextgen* project (Dumez, 2021). 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 (Rajendran and Zack, 2019).

Due to the dynamic nature of the problem, the companies proposing air taxi service will deal with several real-time decision problems. 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, maximize the profit from the trips, 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 (Rajendran and Srinivas, 2020).

The scheduling of air taxi operations combines two problems from the literature: *parcel delivery drones*, and *ground taxi demand scheduling*. The former is used to derive the general characteristics of the flying taxis; the latter helps to design the scheduling models, objectives, and constraints. The problem consists in dispatching a fleet of air taxis so as to serve the costumers, while respecting several operational constraints. The constraints can be divided in two categories: (i) flying taxi-related constraints, like the vertical take-off and landing restrictions, and the battery recharging constraints, and (ii) customer's constraints, such as the time-windows that are a specified period of time during which the customer can be served. Other flying taxi characteristics such as the operating time, the battery recharging time, and battery consumption rate are also present in the parcel delivery drones.

The ground taxi demand scheduling consists in dispatching a fleet of ground taxis to serve a set of clients. Serving a client includes the transportation from an origin to a destination point, taking into consideration several operational constraints such as the *time-windows* and the traffic jam. A time-window is a specified time slot during which a customer can be served. The objective of this problem is usually to maximize profit. According to (Tangpattanakul and Quenel, 2021), the profit from a ground-taxi trip is usually calculated using: (i) a *base rate* (initial charge for the first kilometers), (ii) a *distance rate* (count of the next kilometers), and (iii) a *minute rate* (cost of waiting time in case of congestion). The profit from a flying taxi however should involve only the base and

[1] Acronyms meaning are also outlined in Table 3.

Ikli, S.
A Rolling Horizon Approach for the Dynamic Scheduling of Flying Taxis.
DOI: 10.5220/0011376500003332
In *Proceedings of the 14th International Joint Conference on Computational Intelligence (IJCCI 2022)*, pages 65-73
ISBN: 978-989-758-611-8; ISSN: 2184-2825

the distance rates, since there will be no traffic jam in low level airspace (Kellermann et al., 2020). In the dynamic case, the scheduler must be flexible to accommodate unpredictable events that may occur, such as: the traffic jam, new requests, changes in the customer location, etc.

The scheduling of flying taxis can be classified in two main categories, according to the availability of the data:

- The *static* case, when all input data are known in advance before the day of the operations.
- The *dynamic* or the *real-time* case, when all or some input parameters are unknown for the considered scheduling horizon. This case is underexplored in the literature, and to the best of our knowledge, only the work of (Rajendran, 2021) considers this problem.

In this work, we are interested in the dynamic management of a fleet of flying taxis. The management includes the dynamic scheduling of the flying taxis as well as the battery charging handling. This work is one of the few (Rajendran, 2021) that considers the (more realistic) real-time problem, where decisions have to be made dynamically to accommodate the new changes in the air taxi system. The contributions of our paper are summarized as follows:

- Efficient scheduling heuristics to solve small problem instances. These heuristics are the first-come, first-served, the nearest neighbor, and the genetic algorithm.
- A Rolling Horizon (RH) framework coupled with the above-mentioned heuristics to solve larger instances and to tackle the dynamic case.
- New realistic and challenging problem instances that are publicly available from https://github.com/sanaikli/Dynamic-Flying-Taxi-Scheduling.
- A comparative study of the proposed RH approaches.

2 LITERATURE REVIEW

The scheduling of flying taxis is similar to two well-known combinatorial optimization problems, namely the Job-Shop Scheduling Problem (JSSP) and the Vehicle Routing Problem (VRP). The analogy between these problems is highlighted in Section 2.4. Since the literature on the dynamic scheduling of flying taxis is very scarce, we propose in this section an overview of the research articles addressing the general problems of the dynamic machine scheduling and vehicle routing. The goal of this literature review is to give insights to the readers on how the dynamic scheduling and routing are tackled for different well-known problems, that are similar to the problem of our interest.

2.1 Dynamic Machine Scheduling

The static scheduling assumes that all problem parameters are known in advance and they do not change. In the actual production process, several disturbances may occur, such as a delayed processing time, the arrival of new jobs, etc., which make the previous schedule obsolete. To accommodate the new changes, one must constantly adjust the scheduled plan, in a process known as *dynamic scheduling*.

A common strategy used in the literature to tackle the dynamic scheduling is the *Rolling Horizon* (RH) approach. The latter, also referred to as receding horizon, usually subdivides the scheduling horizon into smaller sub-horizons, and then solves the static problem on each sub-horizon sequentially. In the RH approaches, there are jobs that will start inside a sub-horizon and finish outside that sub-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 (Chen et al., 2017; Fang and Xi, 1997). 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 (Sun and Lin, 1994) 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. Some heuristic rules are used for this subdivision. 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 (Tang et al., 2010), 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 planning 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. 1.

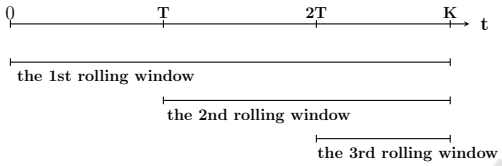


Figure 1: A rolling horizon with three rolling windows.

2.2 Dynamic Vehicle Routing Problem

In the dynamic vehicle routing problem, the customer orders are usually considered to be the dynamic events. The RH approach is also used to solve this problem. For instance, in the work of (Hanshar and Ombuki-Berman, 2007), the periodic scheduling strategy is used, but with time-windows that differs from (Sun and Lin, 1994) and (Tang et al., 2010). Indeed, in (Hanshar and Ombuki-Berman, 2007) , 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 heuristic solutions are proposed in (Larsen et al., 2002) to solve the dynamic VRP. Examples of such heuristics are:

- First-Come First-Served (FCFS) that serves the clients in the order which they are.
- Nearest Neighbor (NN) that completes the demands in one location, and then travels to the nearest neighboring demand.

2.3 Real-time Scheduling and Routing of Air Vehicles

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 (Rajendran, 2021) and (Song et al., 2016) 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 (Song et al., 2016), 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, and solved using *CPLEX* solver (Holmstrom et al., 2009). The real-time management is performed using the two RH approaches: the event-driven and the periodic scheduling strategies presented in Section 2.1.

In the very recent work of (Rajendran, 2021), the author considers the real-time dispatching of air taxis in a centralized taxi network. Two objectives are taken into account: 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 (Sun and Lin, 1994). However, (Rajendran, 2021) does not consider explicit rolling windows, but rather defines some control points on the scheduling horizon, and re-solve the problem at each control point. To the best of our knowledge, this is the only work in the literature that considers the dynamic flying taxi scheduling.

2.4 Analogy and Complexity

The problem of scheduling flying taxi operations is similar to two well-known combinatorial optimization problems, namely the Job-Shop Scheduling Problem (JSSP) and the Vehicle Routing Problem (VRP).

The analogy between a JSSP and the scheduling and routing 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.

In addition, the scheduling and routing of air taxis is clearly similar to the VRP. In the two frameworks, we have vehicles to dispatch and clients to serve. The classical time-windows constraints (Belhaiza et al., 2019) in the VRP framework can directly be translated to the scheduling of flying taxis. Finally, the battery charging of an air taxi can be viewed 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 (Mohan et al., 2019; Derigs and Vogel, 2014)). On the other hand, exact methods may require long computation times as the problem size increases. Hence, heuristic methods are more suitable to solve this scheduling problem, especially the dynamic case, where decisions have to be made in real-time.

3 DYNAMIC FLYING TAXI SCHEDULING: ROLLING HORIZON APPROACH

In the static flying taxi scheduling, it is assumed that complete information about customers requests are known in advance. In the dynamic case however, new requests may arrive on the planning horizon and the solution must be revised. Hence, it is appropriate to adopt a rolling-horizon approach, to accommodate the new changes and reschedule the demands accordingly.

3.1 Rolling Horizon Framework

The RH approach we adopt in this work is the *periodic scheduling* strategy, introduced in Section 2.1. The rolling domain in this context is a *time-window*, *i.e.*, a time slot on the scheduling horizon. Figure 2 illustrates how this approach works.

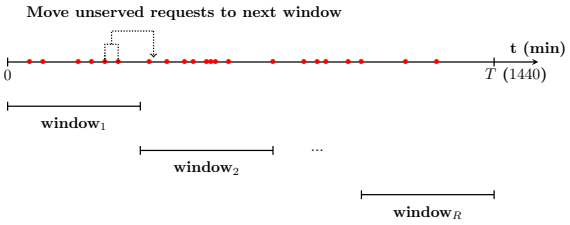


Figure 2: Rolling-horizon strategy illustration.

In the context of our RH strategy, the static problem is solved in the first rolling window, using heuristic methods for creating a flying taxi schedule. In this work, we propose three heuristic methods: one is the GA of (Tangpattanakul and Quenel, 2021) adapted to our problem; the remaining two heuristics are borrowed from the VRP literature and adapted to the flying taxis framework. At the end of the first rolling window, the RH strategy moves the unserved requests to the beginning of the next window, and schedules them together with the new available requests. The battery level for all the available flying taxis is set to 100% at the beginning of the RH approach. Then, its is updated at the end of each rolling window according to the battery consumption rate.

3.2 Heuristic Solutions for Creating Flying Taxi Schedules

In this subsection we describe two heuristic methods and a genetic algorithm that we use to solve the static problem inside each rolling window. The first two heuristics, named “the First-Come, First-Served” (FCFS) and the “Nearest-Neighbor” (NN), are borrowed from the VRP literature.

3.2.1 First-Come, First-Served

The FCFS is a classical heuristic method used in practice to solve several optimization problems, including the VRP. This 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 served (without considering their location), 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. The battery level is checked before each customer pick-up, and updated after each customer drop-off, according to the battery consumption rate. If the battery level of the available taxi is not enough to serve the next client, then the flying taxi is sent to the center for battery recharging.

3.2.2 Nearest Neighbor

The NN is also borrowed from the field of vehicle routing problems. This 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 recompute 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 longer computation times than the FCFS. As for the FCFS, the battery level of each taxi is checked before the customer pick-up, and updated after its drop-off.

3.2.3 Genetic Algorithm

Genetic algorithms are evolutionary algorithms inspired by the process of natural selection. They have shown their efficiency in solving complex combinatorial optimization problems, such as the aircraft landing problem (Hu and Di Paolo, 2011), and the vehicle routing problems (Hanshar and Ombuki-Berman, 2007).

The genetic algorithm we consider in this section is based on (Tangpattanakul and Quenel, 2021). In this work, 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. For example, if we consider an instance with 3 requests and 2 flying taxis, the total number of genes is equal to 6 (Figure 3). A solution of the scheduling problem is constructed from the chromosomes, and it contains the following information:

- The sequences of the selected demands which will be serviced by the flying taxis, and the battery recharging tasks.
- The set of starting time of each task/demands.
- The set of finishing time of each task/demands.
- The operational time of the selected demands.

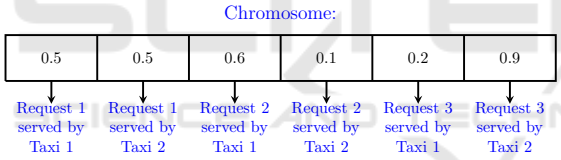


Figure 3: Example of a chromosome with six genes (3 requests/2 taxis).

In our adaptation of this algorithm, the demands can be served during an acceptable time-window, which is more realistic than serving them on strict pick-up times, as it is assumed in (Tangpattanakul and Quenel, 2021). The first population of chromosomes is randomly generated. At each iteration, a new population is generated from three sets of chromosomes: the *elite* set for the selection process, the *crossover* set for the crossover operations and the *mutation* set for the mutation operations. The elite set copies the best chromosomes from the previous iteration. The crossover set contains the offspring chromosomes whose gene values are generated from two different parent chromosomes. The first and the second parent are respectively selected from the elite and the non-elite sets. As for the initial population, the mutation set is also randomly generated, to help escaping from local optimum. The process of mutation and crossover continues until the stopping criteria is satisfied. The latter corresponds to a number of iterations since the last improvement. The fitness function corresponds to the total service time, that the GA seeks to maximize. The parameters of the GA are shown in Table 1.

Table 1: Values of the GA parameters.

Parameter	**Value**
Population size	$2\times$ (chromosome size)
Elite set size	10% of the population size
Mutation set size	20% of population size
Crossover set size	70% of population size
Crossover probability	0.7
Stopping criterion	30

4 COMPUTATIONAL RESULTS

This section reports the computational results of implementing the above-mentioned heuristics, integrated in the rolling horizon 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 we generate for the numerical study. Then, in Section 4.2 we present computational results of implementing our three heuristics FCFS, NN, and the GA, all integrated in the RH approach. The test instances and implementations are publicly available from the following Link: https://github.com/sanaikli/Dynamic-Flying-Taxi-Scheduling.

4.1 New Generated Instances

We randomly generate 10 test instances, based on the instance generator of (Tangpattanakul and Quenel, 2021). In addition to being more congested, our instances define a time window for each demand during which it can be served, which is more realistic than imposing a strict pick-up time.

Table 2 summarizes some important characteristics of our 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 each instance.

Table 2: Characteristics of the new constructed instances.

Instance name	#req	#taxis	req/h	min duration	average duration	max duration
instance10_2	10	2	0.42	17.52	28.16	44.12
instance30_2	30	2	1.25	12.22	26.31	40.61
instance50_3	50	3	2.08	12.16	24.41	43.81
instance80_4	80	4	3.33	11.81	28.58	45.98
instance100_3	100	3	4.17	11.00	26.80	48.57
instance100_4	100	4	4.17	11.48	26.33	48.35
instance200_5	200	5	8.33	11.25	26.97	50.04
instance500_5	500	5	20.83	10.31	27.12	48.33
instance500_10	500	10	20.83	10.74	27.14	51.51
instance1000_15	1000	15	41.67	10.49	27.20	49.35

4.2 Results and Discussion

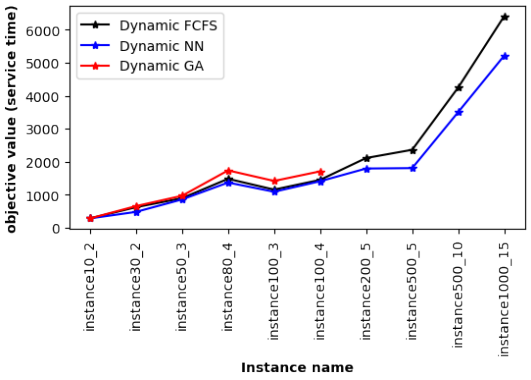
To compare the quality of the solutions provided by our heuristics, we define the following performance indicators:

- The **objective-value** that indicates the total service time (in minutes) that we seek to maximize.
- 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.
- The **CPU** time that indicates the computation times in seconds.

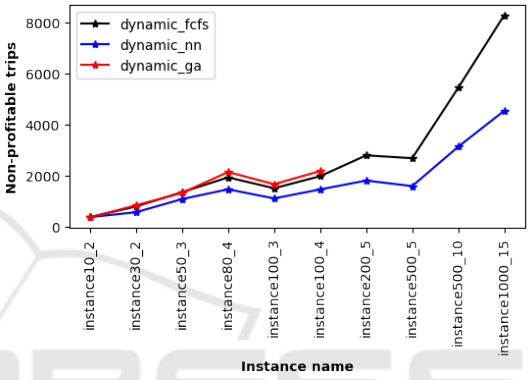
Figure 4 shows the results of the FCFS, the NN, and the GA heuristics on the basis of our three performance measures. The tests are performed on our generated instances, involving 10 to 1000 requests and 2 to 15 flying taxis. The name of each instance is showed in the x-axis of each figure. For these tests, the parameter of the rolling-window length, R, is chosen to be 60 minutes.

It can be seen in Figure 4 that the three heuristics obtain similar performances in terms of the objective-value, for the first six instances. However, in terms of the non-profitable trips, the NN obtains (as expected) better results. This may be explained by the fact that in the NN heuristic, Taxis fly to the nearest neighboring demand location, which minimizes the non-profitable trip between two demands location. In Figure 4c, we can observe that the FCFS heuristic requires very short computation times, even for the very large instances involving more than 100 requests. On the other hand, the computation times for the GA explodes for instances involving more than 100 requests, and we couldn't get any solution with this heuristic for instances involving 200 or more requests. With the NN heuristic, we obtain solutions for all the instances in our data set, but the computation times remain long for the following instances: "instance500_5", "instance500_10", and "instance1000_15".

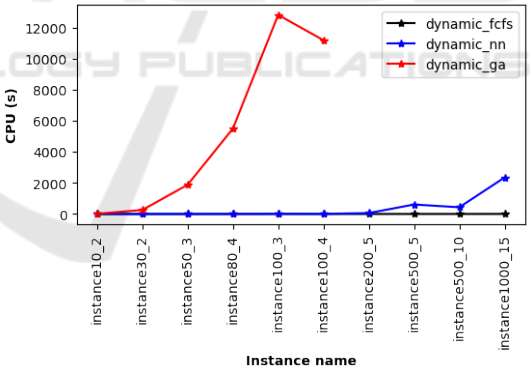
To improve the performance of the NN heuristic, in terms of computation times, we relied on the work



(a) Objective value.



(b) Non-profitable trips.



(c) CPU time (seconds).

Figure 4: Comparison of the FCFS with the NN heuristic on the basis of three performance indicators.

of (Mocnik, 2020). The latter proposes to decompose the space where demands are located into several zones, as illustrated in Figure 5. Then, searching in the current taxi zone for the next neighboring. If no neighbor is found in the zone, our algorithm extends the search to the global zone. The author proves that, by choosing a suitable zone size, the complexity of the algorithm can be reduced.

We implemented this zone decomposition in our

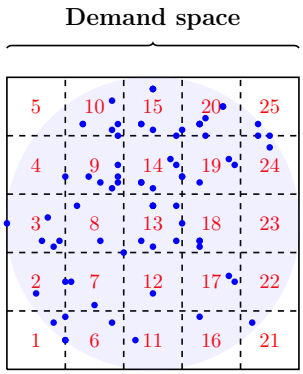


Figure 5: Illustration of the decomposition of the space of requests into 25 zones: the blue dots represent the demands, and the red numbers represent the zones.

NN algorithm using different number of zones: 4, 6, 9, 15, and 20. Figure 6 shows the new computation times of our algorithm with this decomposition. In Figure 6, the curve in red color represents the computation times when considering only one zone (space of demand). The remaining curves (shades of blue) show the results for different number of zones, ranging from 4 to 20. It can be seen in this figure that the computation times are significantly reduced for the improved NN heuristic (blue curves), compared to the naive NN (red curve), for the three above-mentioned large instances. In particular, for the instance named "`instance1000_15`", the computation times were divided by three in the improved NN with 20 zones.

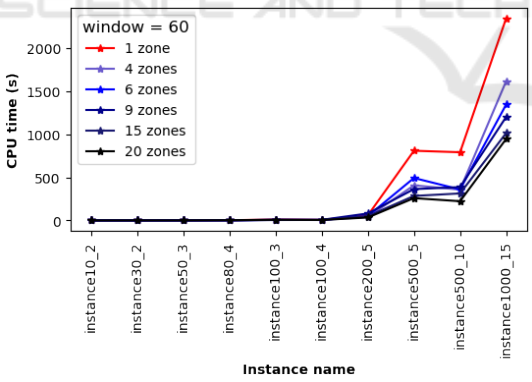
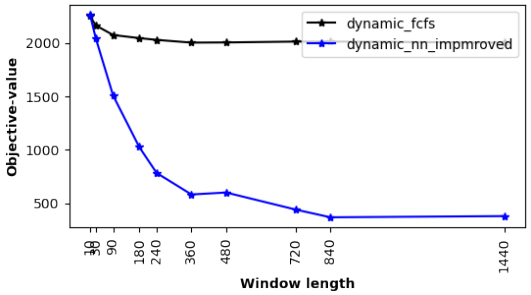


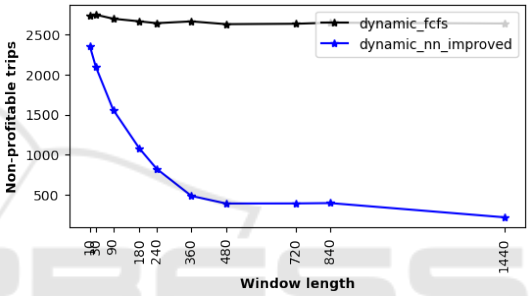
Figure 6: Computation times of the improved NN heuristic.

Effect of the Window Length

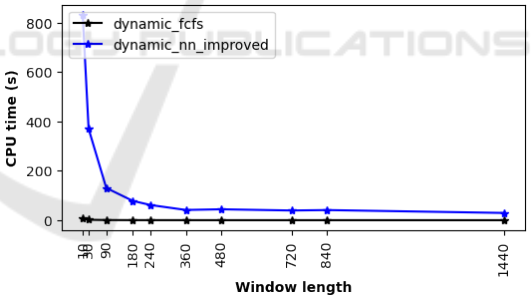
In the previous tests, the rolling-window length was chosen to be 60 minutes. In this study, we test different values for this parameter in order to choose an appropriate value for it. These tests are performed on the FCFS and the improved NN heuristics. For the rolling window length, ten values are chosen in our study: 10, 30, 90, 180, 240, 360, 480, 720, 840, and 1440. The latter value correspond to the scheduling horizon of one day. We perform the tests on our 10 generated instances from Table 2, and the results are averaged over these tests. In Figure 7, the relationship between the length of the rolling window and our three performance indicators is plotted, for each scheduling heuristic.



(a) Objective-value.



(b) Non-profitable trips.



(c) CPU time (seconds).

Figure 7: Effect of the window length on the performance indicators.

First, we observe that the value of our performance indicators (objective value, non-profitable trips, and CPU time) decreases with increasing values of the windows length. In particular, the CPU of the improved NN heuristic drastically decreases in the interval [10, 180] (Figure 7c). Results in Figure 7 suggest that a window length in the interval [90, 180] can be a good compromise between the quality of the solutions – in terms of the objective-value and the duration of the non-profitable trips – and the CPU time.

5 CONCLUSION AND PERSPECTIVES

In this paper, we consider the problem of the *dynamic* scheduling of flying taxis. In this context, all or some problem parameters are unknown in the considered scheduling horizon. We propose a *rolling-horizon* approach, coupled with some heuristics from the literature to solve the dynamic case. The heuristics are: the *First-Come, First-Served (FCFS)*, the *Nearest Neighbor (NN)*, and the *Genetic Algorithm (GA)*.

We conduct several computational experiments to compare the FCFS and the NN heuristics with the genetic algorithm. Results suggest that the FCFS is a good alternative to the GA, because it obtains competitive results and has very short computation times. Moreover, the computation times of the NN heuristic are improved by integrating the decomposition proposed in (Mocnik, 2020). The GA is efficient to solve small and medium instances, involving less than 80 demands. However, for the large instances, this algorithm requires very long computation times, making it unsuitable for a real-time application.

For future studies, we will allow taxis to serve multiple requests in one trip, because serving one client at a time may not be very profitable for the taxi company. We will also construct additional performance measures that could express better the profit obtained from the flying taxi trips.

REFERENCES

Belhaiza, S., M'Hallah, R., Ben Brahim, G., and Laporte, G. (2019). Three multi-start data-driven evolutionary heuristics for the vehicle routing problem with multiple time windows. *Journal of Heuristics*, 25:485–515.

Chen, H., Zhu, J., Zhang, Z., Ma, M., and Shen, X. (2017). Real-time workflows oriented online scheduling in uncertain cloud environment. *The Journal of Supercomputing*, 73:4906–4922.

Derigs, U. and Vogel, U. (2014). Experience with a framework for developing heuristics for solving rich vehicle routing problems. *Journal of Heuristics*, 20:75–106.

Dumez, H.-O. (2021). 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. Online; accessed Mai 13, 2022.

Fang, J. and Xi, Y. (1997). A rolling horizon job shop rescheduling strategy in the dynamic environment. *The International Journal of Advanced Manufacturing Technology*, 13:227–232.

Hanshar, F. and Ombuki-Berman, B. (2007). Dynamic vehicle routing using genetic algorithms. *Applied Intelligence*, 27:89–99.

Holmstrom, K., Goran, A. O., and Edvall, M. M. (2009). V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46:106 pages.

Hu, X.-B. and Di Paolo, E. A. (2011). A ripple-spreading genetic algorithm for the aircraft sequencing problem. *Evolutionary Computation*, 19:77–106.

Kellermann, R., Biehle, T., and Fischer, L. (2020). Drones for parcel and passenger transportation: A literature review. *Transportation Research Interdisciplinary Perspectives*, 4:13 pages.

Larsen, A., Madsen, O. B. G., and Solomon, M. (2002). Partially dynamic vehicle routing—models and algorithms. *Journal of the operational research society*, 53:637–646.

Mocnik, F.-B. (2020). An improved algorithm for dynamic nearest-neighbour models. *Journal of Spatial Science*, pages 1–28.

Mohan, J., Lanka, K., and Rao, N. A. (2019). A review of dynamic job shop scheduling techniques. *Procedia Manufacturing*, 30:34–39.

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

Rajendran, S. and Srinivas, S. (2020). Air taxi service for urban mobility: A critical review of recent developments, future challenges, and opportunities. *Transportation research part E: logistics and transportation review*, 143:(20 pages).

Rajendran, S. and Zack, J. (2019). Insights on strategic air taxi network infrastructure locations using an iterative constrained clustering approach. *Transportation Research Part E: Logistics and Transportation Review*, 128:470–505.

Song, B. D., Kim, J., and Morrison, J. R. (2016). Rolling horizon path planning of an autonomous system of UAVs for persistent cooperative service: MILP formulation and efficient heuristics. *Journal of Intelligent & Robotic Systems*, 84:241–258.

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

Tang, L., Jiang, S., and Liu, J. (2010). Rolling horizon approach for dynamic parallel machine scheduling problem with release times. *Industrial & engineering chemistry research*, 49:381–389.

Tangpattanakul, P. and Quenel, I. (2021). Optimal scheduling for flying taxi operation. In *Proceedings of the 13th International Joint Conference on Computational Intelligence - Volume 1: ECTA*, pages 141–148. SciTePress.

APPENDIX: ACRONYMS

Table 3: Table of acronyms.

Acronym	Meaning
JSSP	Job-Shop Scheduling Problem
RH	Rolling Horizon
UAM	Urban Air Mobility
UAV	Unmanned Aerial Vehicle
VRP	Vehicle Routing Problem