Engineering Applications of Artificial Intelligence 126 (2023) 106856

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

A hybrid multi-start metaheuristic scheduler for astronomical observations

Nariman Nakhjiri [a,d,e,*], Maria Salamó [a,b], Miquel Sànchez-Marrè [c], Juan Carlos Morales [d,e]

[a] Facultat de Matemàtiques i Informàtica, Universitat de Barcelona (UB), Spain
[b] Institute of Complex Systems (UBICS), Universitat de Barcelona (UB), Spain
[c] Intelligent Data Science and Artificial Intelligence Research Centre (IDEAI-UPC), Department of Computer Science, Knowledge Engineering and Machine Learning Group (KEMLG-UPC), Universitat Politècnica de Catalunya (UPC), Spain
[d] Institut de Ciències de l'Espai (ICE, CSIC), Campus UAB, Bellaterra, Spain
[e] Institut d'Estudis Espacials de Catalunya (IEEC), Barcelona, Spain

ARTICLE INFO

Keywords:
Metaheuristic
Scheduling repair
Multi-start algorithm
Telescope scheduling
Artificial intelligence

ABSTRACT

In this paper, we investigate Astronomical Observations Scheduling which is a type of Multi-Objective Combinatorial Optimization Problem, and detail its specific challenges and requirements and propose the Hybrid Accumulative Planner (HAP), a hybrid multi-start metaheuristic scheduler able to adapt to the different variations and demands of the problem. To illustrate the capabilities of the proposal in a real-world scenario, HAP is tested on the Atmospheric Remote-sensing Infrared Exoplanet Large-survey (*Ariel*) mission of the European Space Agency (ESA), and compared with other studies on this subject including an Evolutionary Algorithm (EA) approach. The results show that the proposal outperforms the other methods in the evaluation and achieves better scientific goals than its peers. The consistency of HAP in obtaining better results on the available datasets for Ariel, with various sizes and constraints, demonstrates its competence in scalability and adaptability to different conditions of the problem.

1. Introduction

The scheduling of astronomical observations is a complex problem that can be described from two perspectives: computer science and astronomy. From the computer science perspective, it is a form of combinatorial optimization (Pardalos et al., 2013) with multiple types of tasks, objectives and constraints, bound to different computational cost limits. From the astronomy perspective, scientists are looking for a tool to schedule their requests, which are of varying nature, over different time periods, while giving them sufficient flexibility and adaptability to easily simulate different scenarios, and add or remove conditions from the problem and its dependencies. This paper is aimed at researchers in both the computer science and astrophysics fields, and will hopefully facilitate further studies and collaborations on this interdisciplinary problem.

In complex multi-objective instances of the problem, in order to obtain the most satisfying results, the algorithm should be able to give control to the user over various aspects of the scheduling process down to the individual task level on demand, without needing a structural change. As in any scientific endeavor, researchers in the field of astronomy always look for new ideas and possibilities to explore, and, in order to test them efficiently, a scheduling algorithm is required that can easily take onboard new conditions. For ongoing problems, it is also important to adapt changes without drastically altering plans in the long term or their expected scientific returns.

Scheduling astronomical observations in telescopes is also referred to as the telescope scheduling problem, and, in general, can be defined with the following terminology:

- The scheduler receives a list of *proposals* with different *priorities*. Each *proposal* consists of a set of *scientific tasks* (*sci. task*) which are usually observations of different astronomical targets, and have a scientific value only if all of them are scheduled.
- Each *sci. task* comes with certain constraints and dependencies. According to these conditions, and considering their target coordination and telescope location, the time periods in which the tasks can be done are calculated. Individual tasks do not have priorities and are evaluated on their proposal.
- These time periods are referred to as the *windows* for each task. Depending on the constraints, while the windows for most of the tasks can be calculated before scheduling, for others windows are determined during the process.

The main objective for *sci. tasks* is to maximize the scientific return of the schedule. That means scheduling as many proposals as possible within the designated survey time prioritizing higher valued ones. The

* Corresponding author at: Facultat de Matemàtiques i Informàtica, Universitat de Barcelona (UB), Spain.
E-mail address: nnakhjna21@alumnes.ub.edu (N. Nakhjiri).

https://doi.org/10.1016/j.engappai.2023.106856
Received 22 February 2022; Received in revised form 30 March 2023; Accepted 21 July 2023
Available online 30 August 2023

secondary objective is to maximize the percentage of the time that the telescope is dedicated to do the *sci. tasks*. This requires considering the duration of available tasks.

In addition to scientific tasks, other types need to be scheduled. *Engineering tasks* (*eng. tasks*) are usually related to the maintenance of the telescope and the other involved systems, or to ensuring their precision and quality of operation. These tasks have to be repeated within a certain interval range throughout the schedule. Unlike *sci. tasks*, the main objective for *eng. tasks* is to minimize their number while not violating the interval range. *Eng. tasks* are essential parts of a schedule. However, in planning priority should be given to the *sci. tasks*. In addition to the above-mentioned main objectives related to *sci.* and *eng. tasks*, there are more project specific ones, such as minimizing telescope rotation and device switching between tasks. The high variety in tasks constraint, types and specific objectives leads to the majority of the developments being project specific.

In this paper, we propose a Hybrid Accumulative Planner (HAP), which is a multi-start metaheuristic algorithm to address the main objectives of the telescope scheduling problem, and to fulfill the following design goals:

- Define an expandable, generic task capable of taking on different types of *sci. tasks* and *eng. tasks*. This was necessary to ensure that optimization is coherent along the whole schedule rather than planning different types of tasks in separate routines.
- Flexible and customizable data structure to ease the adaptation of the process of the proposal to new conditions and demands. Abstract constraint definition and the way of handling one task at a time free the user to define a specific setup up to the smallest scale.
- Search the most interesting sections of the solution space for results. This way, even with limited time, a satisfying result is output, and if more time is available, the search expands to other parts of the solution space.
- Achieve a balance between the quality of the constructed solutions and the time required to build and assess them. In algorithms like Multi-Objective Evolutionary Algorithm (MOEA) (Deb, 2015) and Genetic algorithm (GA) (Katoch et al., 2021), the balance is tipped heavily toward faster construction and evaluation, which, in return, produce a large number of less refined solutions. On the other hand, heuristics commonly relies on building one or a few competent solutions but at a higher computational cost. This proposal aims to strike a balance between the quality and computational cost of the building solutions.

The customizability of a HAP is reinforced by the control points in the task and the proposal levels that the user can manipulate to create an optimal schedule. The HAP is a multi-start algorithm (Martí et al., 2018), which runs on a core metaheuristic (Glover and Sörensen, 2015). By utilizing a metaheuristic algorithm as the core approach it was possible to define a generic scheduling task and process that can take numerous forms depending on the requirements and definitions. The metaheuristic of the HAP follows the methodology of a Tabu-search (Laguna, 2018) which has a low computational cost, good exploitation, and offers competent skills to search in the solution space. The multi-start layer of the HAP is used to boost the exploration of the algorithm and to provide a selection of viable solutions based on multiple objectives.

In order to have a scheduling algorithm that is flexible and open enough to allow the user to more easily define specific constraints and introduce new conditions at any time, the HAP schedules are based on a repair strategy (Lange and Werner, 2019). This allows HAP to focus on the scheduling of individual tasks and their demands, if necessary. Whereas fully global optimizations without auxiliary functions do not. The repair strategy of HAP with a multi-start layer has a scalable optimization that starts with the neighborhoods around a current solution, and based on available computational time expands, its search to additional areas.

HAP was tested on *Ariel* mission scheduling (Puig et al., 2018; Edwards et al., 2019) to demonstrate its adaptation to details on a real-world example of the problem. This also created an opportunity to compare its performance with the other existing developments for *Ariel* specifics. HAP was compared against a Multi-layer Evolutionary Algorithm (EA) method proposed in Garcia-Piquer et al. (2017a) that consists of a GA and a MOEA, representing the highest level of global optimization but with high computational cost, and against a Hill climbing (HC) algorithm which has a small computational cost but does not offer high optimization. Evaluating against two extreme cases of the solutions to a problem provided us with a good perspective on the performance of the proposed method.

The main contributions of this work can be summarized as follows:

- We propose a reusable scheduling algorithm, called HAP, for different forms of the telescope scheduling problem, capable of tolerating new demands and constraints.
- We adapted the proposal to a real-world problem to demonstrate the process for handling different types of constraints and tasks.
- We conducted experiments on a real-world problem and its current solutions and evaluated the proposal's effectiveness in comparison to well-known state-of-the-art approaches.

The structure of the rest of this paper is as follows. Section 2 describes some of the proposed solutions and implemented systems for astronomical observation scheduling. The proposed algorithm is detailed in Section 3, followed by its evaluation in Section 4. Finally, Section 5, offers a conclusion to the proposal and directions for future work.

2. State-of-the-art

The problem of scheduling for astronomical observations (Yáñez, 2003) has expanded in terms of both the definition of the problem and the variety of studies around the subject. There has been a noticeable growth in the volume of data and the complexity of facilities and the devices involved. This translates to higher dimensions of solution space and more diversified constraints, which makes it impossible to search the entire space for the best solution. On the other hand, changes that come at any stage of the problem require flexibility in the scheduling method in two aspects: the ability to reschedule or repair the current schedule to react to changes related to its input (*input change*), and the feasibility of adaptation to changes regarding the problem definition and its constraints (*problem change*).

There are similarities between the telescope scheduling problem and some variations of well-known CO problems, such as the Vehicle Routing Problem (VRP) (Braekers et al., 2016), the Elevator Dispatching Problem (EDP) (Tartan et al., 2014), and Dial-A-Ride Problem (DARP) (Masmoudi et al., 2017). For example, although telescope scheduling and vehicle routing are different jobs, they share some constraints and generalized objectives. The studies on the variations of routing problem, like, VRP with time windows (VRPTW) (Schneider et al., 2014), multi-objective (Kumar et al., 2014), with precedence constraint (Razali, 2015), and consistency condition (Kovacs et al., 2015), offer a range of solutions to manage their problem efficiently. The existence of all these conditions, besides the details in the proposal-based constraints and evaluation, adds extra difficulty to the telescope scheduling problem.

From heuristic methods to various fields of Artificial Intelligence (AI), different algorithms have been used to create methods that solve the problem of scheduling for astronomical observations with all its complexities. These methods can be categorized as AI and non-AI approach, and examples of both categories are reviewed in this section. Furthermore, their common strategies for handling *input change*, and *problem change* are summarized here. Finally, the algorithms related to the proposal are discussed.

Many telescopes and facilities use non-AI methods. We can mention the scheduling of the Juan Oro Telescope (TJO) (Colomé et al., 2010), which uses a heuristic method to plan the next task at each moment. The small size of the TJO facility, justifies using a fast responding, locally optimized scheduler. Although small facilities are scheduled manually or use non-AI based methods, the feasibility of these approaches is reduced when the number of telescopes or the period to schedule is increased. Different fields of AI offer solutions for these complex problems. For example, the scheduler for the Atacama Large Millimeter/submillimeter Array (ALMA), within its framework of ALMA Common Software (ACS) (Raffi et al., 2002), provides a constraint propagating method to communicate between its telescopes and to plan a schedule for each one. In ACS, emphasize is more on satisfying the variety of constraints arising from the different telescopes and proposals. These approaches have more flexibility in terms of *problem changes*.

One of the main tools, which is still in use in many large projects and was initially developed for the Hubble telescope, is *Spike* (Johnston, 1990). Spike uses a heuristic constraint satisfaction approach to build and update the plans and utilizes artificial neural networks to anticipate changes and keep run-time schedule manipulation to a minimum. The spike scheduling approach has been adapted to other major projects, such as the Very Large Telescope (VLT) operated by European Southern Observatory (ESO), which is one of the world's largest optical telescopes and is run on the adaptation of Spike described in Johnston (1988). Spike is also partially adapted by the Subaru telescope, which expands the VLT variation of Spike, according to its requirements as explained in Sasaki et al. (2000). Mora and Solar in Mora and Solar (2010), have gathered a survey of scheduling tools for operating projects in the field. With constraint satisfaction models like the one implemented in Spike, the main focus is on the availability of valid schedules to execute, and schedule optimization is a second priority.

On the other hand, emphasizing schedule efficiency has led many studies to focus on optimization algorithms. One of the more popular categories of algorithms to solve this problem is Evolutionary Computation (EC), especially Evolutionary Algorithms (EA). EA are competent approaches to large problems, but have high computational costs. In order to handle small *input changes* in a limited time, the system which runs mainly on an EA, and usually also includes a fast heuristic. This strategy has been the subject of several studies (Moisana et al., 2002; van Rooyen et al., 2018), and relies on its heuristic to give flexibility for *input changes*. Garcia et al. in Garcia-Piquer et al. (2017b) add a second EA for scheduling CARMENES (Calar Alto high-Resolution search for M dwarfs with Exo-earths with Near-infrared and optical Échelle Spectrographs). Specifically, his proposal consists of three layers using a Genetic Algorithm (GA), a Multi-Objective Evolutionary Algorithm (MOEA), and a greedy heuristic algorithm In this way, some of the *input changes* are also handled by the MOEA, and not all the responsibility is on its heuristics layer.

The need for a fast response to *input changes* has greater importance for ground-based telescopes. For telescopes based in space, where conditions are more stable, there is no need for a fast heuristic. For example, we can mention the James Webb Space Telescope (JWST) and the *Ariel* mission. The JWST scheduling tool (Giuliano and Johnston, 2008) uses Spike as its core and adds an EA layer on top of it to provide better optimization. The *Ariel* scheduler (Garcia-Piquer et al., 2017a) has similarities with the methods used for CARMENES and uses a two layer structure consisting of a GA and a MOEA.

HAP, the method proposed in this paper, follows a Multi-Start (Martí et al., 2019) hybrid metaheuristics for scheduling and optimization. This trend in algorithms has increasingly become a focus of recent studies of complex multi-objective combinatorial optimization (Blum and Li, 2008). Studies like (Blum et al., 2008; Parpinelli and Lopes, 2011), which illustrate the popularity of these approaches, argue that the new inspirations in hybrid metaheuristic algorithms are a result of their more efficient behavior and greater flexibility. Examples of metaheuristics for multi-objective optimization are Tabu Search (Laguna, 2018), Simulated Annealing (Delahaye et al., 2019), and Variable Neighborhood Search (Hansen et al., 2019). Tabu search has been the subject of numerous studies of the problem. We can mention (Chou et al., 2014; Jemai et al., 2017; Saidi-Mehrabad and Fattahi, 2007), where the authors utilize a Tabu search for its flexibility in design and its relatively low computational cost. These characteristics suit our objectives as defined in Section 1.

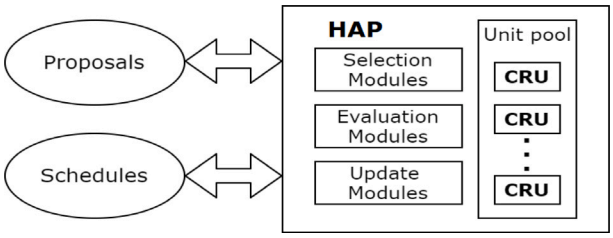

Fig. 1. General overview of HAP components.

SI and multi-start methods both operate by generating a number of stochastic solutions. However SI examples like, the base models of Particle Swarm Optimization (Wang et al., 2018) and Ant Colony Optimization (Dorigo and Socha, 2018) as global optimization algorithms, depend heavily on large populations and extensive numbers of iterations. This reduces their performance on limited available time. To address this issue several variations of these algorithms have been proposed in the literature, including combinations with local optimization algorithms (Han et al., 2017), and constructive approaches (Song et al., 2019). On the other hand, multi-start algorithms are commonly built on local optimizations (György and Kocsis, 2011; Li et al., 2019; Kessaci et al., 2014) to improve their exploration. In this way, the algorithm operates even with limited time and available resources (Martí et al., 2018). Because of this flexibility, HAP uses a multi-start layer to explore additional neighborhoods and comply with various time restrictions.

3. Hybrid accumulative planner

This section details the components and the algorithmic process of HAP. The tasks to schedule, and the constraints which define their dependencies and boundaries, are represented as a set of proposals in the telescope scheduling problem. The HAP strategy to approach this problem is described as follows: First we give an overview of the components that make up the algorithm and summarize the required terminology in Section 3.1. Then, in Section 3.2, the heuristic core and the main component of HAP, called the Conflict Resolution Unit (CRU), is described. Finally, the HAP process is detailed in Section 3.3.

3.1. Overview and components

As a multi-start algorithm, HAP utilizes a number of core metaheuristic task schedulers in its process, which are referred to as CRU. CRU handles the tasks and their constraints at the individual level, while HAP, as the higher layer, controls CRU inputs and evaluates their results at the proposal level. Fig. 1, illustrates a high level scheme of the input and the different parts of HAP.

As can be seen in Fig. 1, HAP takes sets of proposals and schedules as input and updates them based on its process. There are several components in HAP that can be categorized as different control modules (e.g. Selection, Evaluation, and Update) and a unit pool. Control modules consist of selection, evaluation and update. By applying different strategies for these components, HAP handles tasks of different types. The unit pool (*Pool*) contains several instances of its task scheduler, CRU, that HAP can access in order to schedule the tasks of the proposals. We finish this section with Table 1, detailing the terms and denotations that are used in the algorithm descriptions.

Table 1
Description of the HAP denotations.

Denote	Name	Description
C^w	Cost	Sum of priority for members of CG^w which lose completeness.
CG^w	Conflict group	Set of tasks from a schedule, which overlap in time with window w
F_C	Completion fitness	HAP fitness function. On proposal completeness and priority.
F_T	Time fitness	HAP fitness function. On the time dedicated to *sci. tasks*.
k	Number of CRUs	HAP configuration, integer. The value is set based on available resources.
L	Lobby	Set of tasks, waiting to be scheduled by CRU.
max_{iter}	Maximum iteration	CRU configuration, integer. Maximum number of iteration for a CRU main cycle.
p	Proposal	Set of dependent tasks.
P_i		Set of proposals at step i of HAP.
Pri^p	Priority	Priority value of proposal p.
s	Schedule	Sequenced list of non-overlapping tasks with specific window.
S_i		Set of schedules at step i of HAP.
sR	Stochastic range	CRU configuration, integer ≥ 1. The number of options consider for CRU to for stochastic replacement.
t	Task	A user request for a time on schedule.
T		Set of tasks.
$target$		A celestial object which a task is defined on its observation.
w	Window	Possible period of time to schedule a task.
W^t		Set of windows for task t.

3.2. Conflict Resolution Unit (CRU)

The responsibility of the CRU is to add new tasks into a given schedule regardless of their added values and by minimize the damage to the already existing proposals in that schedule. HAP calls a CRU with a non-empty set of tasks T, a schedule s, and its configuration, and it always returns a schedule which includes a window for each member of T and the tasks in s which do not overlap in time with the newly scheduled windows. This process is explained in more detail in algorithm 1.

Algorithm 1, shows the process of a CRU. Beside a set of tasks T and a schedule s, CRU receives its configuration values as input. The default configuration of stochastic range (sR), and maximum iteration (max_{iter}) are specified in algorithm 1 and are explained on their use in the pseudo-code. The first step is to initialize the required variables, including a waiting list called *Lobby* (L). Lobby always contains the tasks that a CRU tries to put into its schedule s. In line 2, we fill the lobby L with the members of T. The other initialization steps are to assume an empty schedule for $best\_schedule$ and set min_score and i to zero. i in line 5 is the iteration counter. The main cycle of CRU spans from line 6 to 26 and its goal, represented as a while loop, is to empty the lobby before it reaches its limitations. The limitation for a CRU is the maximum number of iterations for its main cycle (max_{iter}). After increasing the iteration in line 7, CRU selects a task t from its lobby as indicated in line 8. The default selection strategy is to first select the members of T and then treat the lobby as a stack (last-in first-out). The selected task t is removed from L (line 9) and CRU starts a search to find the best position to put t into s, by going through the windows of t (W^t) one by one in the *For* loop between lines 10 to 17. For a window of t, ($w : w \in W^t$), CRU calculates two values. The first is the Conflict Group of w (CG^w) and the second is its cost (C^w). CG^w is a set of tasks from s whose scheduled window overlaps in time with w (line 11), so in scheduling we can only choose one or the other to be in the solution. After gathering CG^w, we calculate its cost C^w by calling $Cost()$ function (line 12). The default variation of the $Cost()$ function adds up the priority of the members of CG^w whose removal from s damages the completeness of their proposal, multiplied by a value which act as a preference measure between the tasks with the same priority, called

Algorithm 1 CRU algorithm

```
1: procedure CRU(T, s, sR, max_iter)                    ▷ Input.
2:     L ← T                                            ▷ Lobby initialization.
3:     best_schedule ← ∅
4:     min_cost ← 0
5:     i ← 0                                            ▷ Iteration.
6:     while L! = ∅ & i < max_iter do
7:         i ← i + 1
8:         t : t ∈ L                                    ▷ Selects a task from L.
9:         remove t from L
10:        for all w : w ∈ W^t do
11:            CG^w = {t_s : t_s ∈ s & t_s overlaps w}
12:            C^w = Cost(CG^w)
13:            if CG^w = ∅ then
14:                add w to s    ▷ Assigns w as the window to perform task t.
15:                goto line 6
16:            end if
17:        end for
18:        w_st ← StochasticSelection(C^w : ∀w ∈ W^t, sR)
19:        add w_st to s ▷ Assigns w_st as the window to perform task t.
20:        remove CG^{w_st} from s
21:        add CG^{w_st} to L
22:        if Cost(L − T) < min_cost then
23:            min_cost = Cost(L − T)
24:            best_schedule = s
25:        end if
26:    end while
27:    return best_schedule                             ▷ Input of the next cycle.
28: end procedure
```

Impatience (ϱ^t). Eq. (1) describes C^w for task t, calculated by the $Cost()$ function:

$$C^w = \sum_{\tau \in CG^w} (Pri^\tau + \frac{\varrho^\tau * Pri_{min}}{10^\alpha}) \quad (1)$$

In Eq. (1), Pri_{min} represents the minimum priority value in the survey, while α determines the weight of impatience in C^w calculations, with a default value of 1. Impatience is a normalized value ($0 < \varrho^t \leq 1$) and multiplying it by Pri_{min} results in limiting the effect of it to less than priority, thus becoming a secondary attribute to distinguish between the tasks. The default variation of the Impatience of a task t (ϱ^t), from a proposal p, is calculated as shown in Eq. (2):

$$\forall t \in p, \varrho^t = \frac{|p|}{|\bigcup_{\tau \in p} W^\tau|} \tag{2}$$

According to Eq. (2), ϱ is equal for all the tasks in a proposal, and it depends on two values. The first is the number of tasks in proposal p ($|p|$), while the second is the total number of available unique windows for tasks of a proposal. In this way, if a proposal consists of a repetition of a single task, $|\bigcup_{\tau \in p} W^\tau|$ is equal to the size of W^t, because there is only one set of unique windows.

As indicated in algorithm 1, if there is a w whose CG^w is empty (or $C^w = 0$), then CRU adds the task t on w into s (line 12) and goes back to the while loop to pick another task from L (line 13). If there is no window with an empty CG, CRU picks a window w_{st} with a stochastic selection from W^t based on their cost. $StochasticSelection()$ sorts the costs in ascending order, and makes a weighted decision on the first sR number of elements to select one. sR or stochastic range is a configuration integer value of CRU which indicates the number of elements that $StochasticSelection()$ considers in making the weighted selection. Having a lower cost increases the chances of being selected. Definition (3), shows the probability of a window ($prob^w$) in the range of sR, to be picked as w_{st} in $StochasticSelection()$, as shown in line 18 in Algorithm 1:

$$prob^w = \frac{C^w}{\sum_{i=1}^{sR} C^{w_i}} \tag{3}$$

After the stochastic selection of w_{st}, CRU adds the task t on w_{st} into s, removes its conflict group $CG^{w_{st}}$, and adds them into L (lines 19, 20, and 21 respectively). The final check in CRU is to keep track of the best schedule found so far. According to the cost of lobby L without members of T ($Cost(L-T)$), if it is lower than the previously recorded minimum cost (min_{cost}), update the $best\_schedule$. $best\_schedule$ is initiated in line 3 as an empty schedule, and CRU returns it as its solution in line 27. This decision to keep track of the best solution is based on the way CRU explores the solution space. In its methodology, CRU follows a Tabu-search metaheuristic. While it tries to minimize the cost of adding T into s, it allows for increases in the cost in its cycles in order to get out of local minimums and explore surrounding neighborhoods, so the best-assembled solution can be in the intermediate cycles.

3.3. HAP algorithm

HAP has the responsibility of handling tasks at the proposal level, and decides which proposals are added or removed from a schedule. It does so by managing available CRU input and output. In HAP, several CRUs work concurrently in order to explore more of the solution space depending on the availability of the resources in terms of time and computational power. Based on resources, HAP can be configured in different aspects. In addition to the configuration of CRU, including sR and max_{iter}, HAP is set with the number of CRUs (k) that it has access. Although increasing both max_{iter} and k improves search quality, increasing max_{iter} contributes more to the exploitation of the algorithm. On the other hand, a higher number of CRUs or increased k results in a greater exploration of HAP in the solution space. The stochastic range or sR, also contributes more to the exploration and affects the diversity of the solutions from different CRUs. The solutions from higher sR cover a larger range of solutions, and while this can lead to finding better solutions, it is best to increase it when there are enough CRUs available to explore the available range.

HAP as a repair algorithm, processes a subset of all the proposals ($p \subseteq P$) at one time, and at its initial iteration ($i = 0$) starts with an empty set of schedules ($S_0 \leftarrow \varnothing$) and the set of all the proposals ($P_0 \leftarrow P$). Depending on the *Selection* strategy on $p \subseteq P$, and through a certain number of iterations, HAP processes all the proposals in P at least once, and returns when complete. Algorithm 2, describes the algorithm of HAP at iteration i of its process in detail.

Algorithm 2 HAP's algorithm.

```
1:  procedure HAP(p ⊆ P_i, S_i, K, sR, max_iter)   ▷ Input of the ith cycle.
2:      R ← ∅                  ▷ A set of schedules to hold intermediate results.
3:      for all CRU_k : 1 ≤ k ≤ K do
4:          s'_k ← s_k ∈ S_i                ▷ Selects a schedule for the CRU_k.
5:          for all t ∈ p do
6:              s'_k ← CRU_k(t, s'_k, sR, max_iter)          ▷ Calling the CRU
    procedure.
7:              add s'_k to R
8:          end for
9:      end for
10:     S_{i+1} = Evaluation(R ∪ S_i)
11:     P_{i+1} = Update(S_{i+1}, P_i)
12:     return (P_{i+1}, S_{i+1})                    ▷ Input of the next cycle.
13: end procedure
```

According to algorithm 2, the input of HAP for its ith iteration is detailed in line 1 and consists of a subset of remaining proposals at this point ($p \subseteq P_i$), the set of the best schedules found in the previous iteration (S_i), and its configurations. Line 2 initializes R as an empty set of schedules to be filled in later with the results coming from CRUs.

The main loop of HAP is on all the k available CRUs in the system and spans from lines 3 to 9 of the pseudo-code. This loop first selects a schedule for the kth CRU from the available options (line 4). The decision on how to distribute the schedules in S_i between the CRUs depends on the *Selection* strategy. Assuming s_k as the selected base schedule for CRU_k, HAP goes through all the tasks in p and adds them accumulatively to s_k with CRU_k. This is shown in line 6 of algorithm 2, where s_k is continually updated with the results from CRU_k. It then proceeds to add s_k to the results set of R (line 7). This is done to have the option to handle the entry of proposals that have several completion levels. At the end of the main loop in line 9, every task has been processed by all the available CRU. After the CRUs finish their process and all their results are added to R in line 10, HAP proceeds to evaluate all the schedules from the set R and from the current iteration (S_i) with $Evaluation()$. The result of the $Evaluation()$ makes up the set of schedules for the next iteration (S_{i+1}), and it contains the most competent schedules that are selected based on the *evaluation strategy*, and their fitness values on the different objectives of the problem. Different *evaluation strategies* can be applied when selecting the schedules for S_{i+1}. The default approach of HAP is to select not more than the number of available CRUs ($|S_{i+1}| \leq K$), based on an elitist selection. Another strategy is to choose only the best schedule and call all the CRUs on it for the next iteration. By pruning the input diversity of CRUs in the second strategy, we decrease the exploration and increase the exploitation of HAP for a given proposal. The default strategy ranks the schedules in an evaluation based on a single value made up of a combination of the objective fitness with the weights determined by an analytic hierarchy process (AHP) (William and Xin, 2018).

On the other hand, the objectives of the problem require specific fitness functions to assign values on the competence of a schedule. These functions are added to the $Evaluation()$. To satisfy the main common objectives of the problem, two fitness functions are defined in the base setup of HAP. Eq. (4) describes the fitness functions F_C, which reflects the completeness of the proposals with regard to their priority, and F_T, which is based on the time dedicated to *sci. tasks*. These functions are defined on a schedule s, which contains different

tasks (t) of different proposals ($t \in p, p \in P$) with specific windows ($w \in W^t$).

$$F_C(s) = \sum_{p \in (s \cap P)} \vartheta^p * pri^p \qquad F_T(s) = \sum_{w \in s} |w| \tag{4}$$

Eq. (4) reviews the default fitness functions of HAP. A schedule as defined in the algorithm, can be iterated based on p, t, or w. In F_C, for a proposal p, $p \in (s \cap P)$ is true when it belongs to the set of all proposals (P), and has at least one of its tasks with a specific window, scheduled in s. The priority of the proposals is denoted with pri^p, and $0 \leq \vartheta^t \leq 1$ is the completion level multiplier. This value is calculated based on the constraints and the number of tasks from a proposal which are scheduled. The value of ϑ for the proposals that only have a value if all their tasks are scheduled can only switch between the two options: 0 if the number of tasks is less than all and 1 if all the tasks are planned. On the other hand, proposals which have different acceptable levels in their completeness can choose in the range of $[0, 1]$.

F_T iterates over s on the windows ($w \in s$). These windows (w) in s are from the tasks (t) that belong to a proposal ($t \in p$) from the set of proposals ($p \in P$). In Eq. (4) $|w|$ represents the size or time duration of the window.

4. Evaluation

In order to evaluate the performance of HAP, the algorithm is adapted to the *Ariel* space mission. Utilizing a real-world example of the problem in evaluation, provides a complete insight into HAP's algorithm behavior and its adaptation capability. In addition, it allows us to compare HAP with other existing solutions that are designed specifically for this problem and its complexities. In this section first a description of the *Ariel* mission is provided in Section 4.1. After that we detail the available methodological approaches to the *Ariel* problem in Section 4.3 and make a comparison of their test results in 4.4. Finally, a summary of HAP's performance, according to the analysis of the test results is provided in Section 4.5.

4.1. Ariel mission

The Atmospheric Remote-sensing Infrared Exoplanet Large-survey, *Ariel*, is an European Space Agency (ESA) space mission under design study that consists of a 1-m class telescope with a low-resolution spectrograph as the main instrument. Its goal is to study of a large sample of exoplanets to understand the structure of their atmosphere (Tinetti et al., 2018) and to provide information about the planet formation and evolution. *Ariel* is expected to be launched in 2029 and during its 3.5 years of nominal operation phase it will characterize the atmosphere of about 1000 exoplanet (Edwards et al., 2019) targets. To do so, it will observe two types of planetary events: transits, i.e. when the planet passes in front of the star; and occultations, i.e. when it passes behind. Such observations can only be performed in specific times according to the ephemerides and orbital properties of each planetary system. This inflicts strict constraints to the scheduling of the observations. In addition, to ensure the overall quality of the data received from each target, observations are repeated for certain number of times. With the definitions detailed in this paper, a proposal p of *Ariel*, consists of several tasks with the same set of windows ($W^\tau = W^\sigma, \forall \tau, \sigma \in p$). *Ariel* may also be useful to study the planet spectrum variability during its orbit around the host star (hereafter, phase curve) for several systems in order to study atmospheric circulation. Phase Curve (PC) observations have much longer windows, compared to the rest of the *sci. tasks*, and require specific handling strategies to prevent violating constraints of the other tasks.

Beside scheduling the *sci. tasks* of the targets from the proposals, the scheduler for *Ariel* mission should also consider *eng. tasks*, which consist of calibrations and station keeping operations. Calibrations are periodical tasks to observe a stable star in order to check for the precision of the telescope. There are short calibrations which are 1 h *eng. tasks* every 36 ± 12 h, and long calibrations defined as 6 h *eng. tasks* every 30 ± 1 days that monitor the instrumental stability at different scale. As for the station keeping operations, which are regular maintenance tasks, frequency is set to 4 h every 28 ± 3 days. The main difference between the *eng.* and the *sci. tasks* of *Ariel*, is the prior knowledge of the number of required tasks to satisfy the constraints. While this number is predefined as repetition for the sci. tasks of the survey, the final number of eng. tasks in the schedule depends on how close we plan them together. The objective here is to minimize the number of these tasks and comply with their respective frequency request where possible. All the methods in the evaluation, prioritize scheduling of *sci.* tasks over the *eng.* ones. This causes some violation of boundaries for the eng. tasks which has to be mitigated as much as possible. Finally, in the scheduling, the time to re-position the telescope between two consecutive tasks (from hereafter, *Slew*) should also be taken into account. Slew time depends on the coordination of the tasks and speed of the telescope movement.

4.2. Datasets

Table 2, describes the datasets in the evaluation. There are four datasets currently available for *Ariel* mission planning. These datasets are used to simulate different scheduling scenarios to evaluate a final plan for the mission. The main goal of *Ariel* is to complete the survey of 1000 proposals to observe exoplanets. These proposals are provided in the Mission Reference Sample (MRS) dataset. The required tasks for the MRS targets observations cover ∼70% of the mission time, and in order to increase the dedicated time to science, back-up targets are added in MRS_B. MRS_B, includes an additional ∼1100 Tier 1 proposals on top of MRS. Fulfilling all the MRS_B proposals tasks requires ∼200% of the mission time. Phase curves, noted as Tier 4, are included in the datasets MRS^{PC} and MRS_B^{PC} to investigate the cost of adding these very long tasks to the schedule compare to MRS and MRS_B respectively.

According to Table 2, proposals in the datasets are categorized in different tiers. The total number of proposals in the dataset is mentioned in the second column while the number of existing ones from each tier, are represented in columns 3 to 6. Each tier comes with a priority value. The third column in the table, Tier 4, represents the number of phase curve proposals in the dataset. Tier 3 are the benchmark proposals which have the highest priority value, and should be all included in any acceptable schedule. Tier 2 or deep survey proposals valued less than Tier 3, however, the goal is to complete the tasks of these proposals too if they do not disrupt any higher tier tasks. Finally, Tier 1 consists of the survey and back-up targets, which their main purpose of addition is to use the rest of the time unused by the other tiers to investigate less important proposals. The priorities values assigned to different tiers are as follows: 100 to Tier 4, 100 to Tier 3, 10 to Tier 2, and 3 to Tier 1.

The number of repetition for a task of a proposal in *Ariel* problem has a direct link to its tier. To complete a proposal to a higher tier, we have to schedule more tasks for it. Beside Tier 4, for the rest of the tiers, if a proposal cannot be scheduled with the initially request number of tasks, it will get another chance to be scheduled with a downgraded tier and a lower number of repetition.

Coverage time or $Cvg.time$ column in Table 2, shows the percentage of mission duration that is required to complete all the tasks required by the targets of a dataset. Columns 8 and 9 count total number of windows and required tasks in the dataset. Finally, the last column in Table 2, indicates the average conflicts between windows. For example, average conflict of dataset MRS_B^{PC}, shows that at any moment during the mission, on average ∼177 targets are observable and the scheduler can only plan maximum one of them.

Fig. 2, visualizes the distribution of targets windows in the MRS dataset throughout the mission duration. Each row in Fig. 2, represents a target and each bar shows a time window when the observation task can be done.

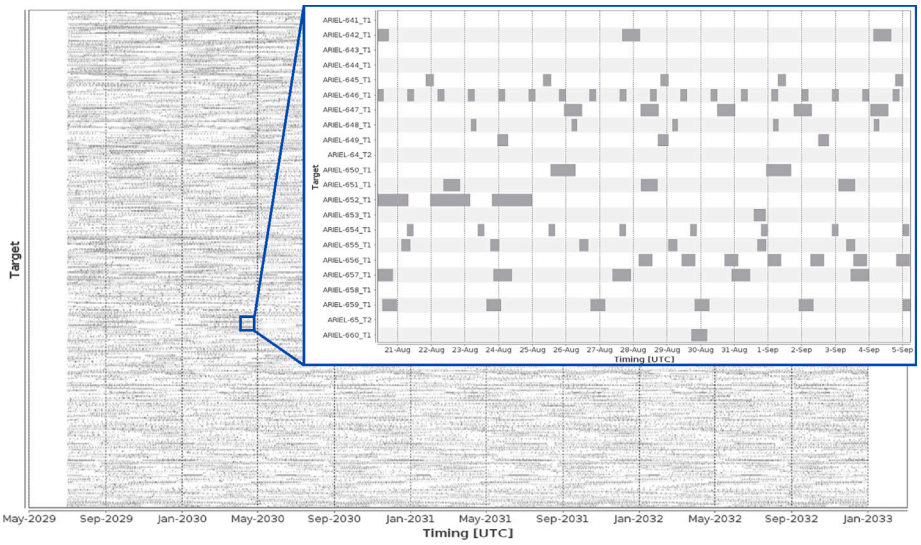

Fig. 2. Targets windows distribution of MRS for the mission duration.

Table 2
Datasets characterization. Considering the following features: The number of proposals containing *sci. tasks*, a count of the proposals of different tiers from Tier 4 to Tier 1, the coverage time of the proposals, the number of windows, the number *sci. tasks*, and finally, the average conflict or overlap between the window.

Dataset name	Total # proposals	Tier 4	Tier 3	Tier 2	Tier 1	Cvg. time	Total # windows	Total #tasks	Avg. conf.
MRS	999	0	50	550	399	69.96%	186,236	3020	78.9
MRS^{PC}	1042	43	50	550	399	81.61%	197,410	3063	112.0
MRS_B	2091	0	50	550	1491	199.37%	304,885	6667	143.9
MRS_B^{PC}	2134	43	50	550	1491	211.02%	316,059	6710	176.9

4.3. Methods in evaluation

There are two developments available for *Ariel* mission scheduling. The first one is based on the examples of Evolutionary Computation algorithms, described in Garcia-Piquer et al. (2017a), and utilizes variations of Evolutionary Algorithms (EA) global optimizations in its process. The second approach uses a greedy hill-climbing (HC) algorithm, a local optimization, to schedule the mission. Greedy algorithms are common solutions for small facilities and projects where there is not much congestion in the schedule. Although greedy and EA approaches are too different, the competence of their final output on *Ariel* datasets are comparable considering all aspects. There are two main reasons to explain this outcome. First is that the solution space has numerous neighborhoods which their local minimums are not far apart. Second, the limitations of the EA design to handle different types of tasks negatively affects its performance. In the rest of this section, the setups for the methods in evaluation are described. Section 4.3.1 reviews the EA method, and Section 4.3.2 details the HC. Finally, Section 4.3.3 sets out the adaptation of HAP to the *Ariel* problem, and its setting.

4.3.1. EA method

In an EA, the key is to generate and evaluate a large number of solutions. In a problem with a variety of constraints, dependencies, and special cases, some compromises should be taken into account to maintain the pace. For the EA used in the *Ariel* example, *sci.* and *eng. tasks* are scheduled in separate processes. Calibrations and station keeping are scheduled at first by a Genetic Algorithm (GA), with the objective to minimize potential conflicts with the *sci. tasks*. In the implementation, in order to match the properties of the algorithm, a fixed number of tasks is assumed and scheduled. This fixed number is estimated from the average cadence of these tasks, to make sure this cadence is fulfilled. The resulting schedule is then passed to a Multi-Objective Evolutionary Algorithm (MOEA), with similar objectives as HAP, to plan the *sci. tasks* without touching the *eng. tasks*. The issue with this adaptation is that by separating the scheduling of different types of tasks, and also by assuming a fixed number of *eng. tasks* instead of dynamically reaching the necessary number, we lose many optimization opportunities. This design decision is enforced by the characteristics of the incorporated EA, and the specification of the problem.

Table 3
Algorithm parameters for the EA method.

Parameter	Value
Max generations	5000
Initial population	500
Max population	1500
Crossover probability	0.9
Mutation probability	$1/\|individual\|$

The genes of the individuals in the utilized MOEA, represent the requested observations of every target, and their allele shows the scheduled window. The algorithm parameters, applied to both GA and MOEA, for the tests are detailed in Table 3, where $|individual|$ means the size of an individual.

4.3.2. HC method

The second method used in the evaluations, is a hill-climbing (HC) greedy algorithm, offering a local optimization to solve the problem. The main advantage of local optimizations is the low computational cost. Also, due to the simplicity of its design compared to EA, it is much more accessible and easier to implement and adapt to new problems and special cases. On the other hand, the main downside of the hill-climbing algorithm is its limitation in optimizing complex problems. Greedy algorithms work on less congested problems but come short in problems where the ratio of available time to the proposal requests

is low. The hill-climbing method in evaluation is directed by four normalized decision variables ($\{d_1, d_2, d_3, d_4\}$) defined on every window of the tasks.

A combination of hierarchical evaluation and weighted aggregation of these values determines the overall competence of a window when building a schedule. At a scheduling moment m, for a target t and a schedule p, decision variables are calculated as described in definition (5).

$$d_1 = \begin{cases} 1 \text{ if has tasks scheduled} \\ 0 \text{ otherwise} \end{cases} \qquad d_3 = \frac{pri}{pri_{max}}$$
$$d_2 = \frac{|t| - |p \cap t|}{|W_t'|}, w \in W_t' : \begin{cases} w \in W_t \\ w.begin > m \end{cases} \qquad d_4 = \frac{d_{max} - d}{d_{max}} \tag{5}$$

The first decision variable, d_1, promotes finishing the tasks that already exist in the schedule. d_2, rates the number of tasks left to schedule to the remaining windows of a target. This is calculated by reducing the number of planned tasks of a target t in the schedule p ($|p \cap t|$) from the number of requested tasks from t ($|t|$), divided by the number of the target's windows that start after m ($d = w.begin > m$). Finally, d_3 and d_4 represent the normalized priority and delay respectively. pri_{max} contains the maximum priority value of the proposals. d is the delay of a window relative to the scheduler time ($w.begin - m$), and d_{max} holds the maximum allowed delay in the execution.

4.3.3. HAP method

The adapted HAP for *Ariel* problem uses the framework defined in Section 3. Different setups are defined to handle different types of tasks within the same process. For the *sci.* tasks of the problem, the *Selection* strategy is selected to make a full constructive method. This translates to having one schedule for each CRU ($\|S_i\| = K$), and HAP, passing the best results of CRUs to themselves from their previous iteration. This distribution of resources allows us to explore a wider range of solutions, as their diversity is not reduced in the intermediate iterations. The input schedule for the CRU_k at iteration $i+1$, and is either s_k, or s'_k from the last call of CRU_k in iteration i.

In *Ariel*, proposals have scientific value when all their tasks are scheduled, so in the *Evaluation* strategy of the adapted HAP, and calculation of the fitness F_C, the value of ϑ is either 0, indicating a proposal is not finished, or 1, representing a fully scheduled one. In the calculation of F_C, the given priorities of different tiers are $\{100, 100, 10, 3\}$ for Tier 4 to Tier 1 respectively. In order to give a more clear perspective, the F_C values that are represented in the evaluation are normalized between $[0, 100]$. In this case, the value of 100 shows all the proposals in the dataset are scheduled. It is worth mentioning that for MRS_B and MRS_B^{PC} datasets, reaching the maximum value is impossible, as scheduling all the proposals require twice the available time for *Ariel*.

In the *Ariel* problem, F_C is the main objective. Growing number of proposals as the survey progresses, lowers the emphasis on the other fitness value, F_T. F_T does not need any modification and is used as the default format, with a quarter weight of the F_C. Finally the *Update* strategy for the *sci.* tasks of *Ariel* is the default approach, where the proposals which are already completed in S_{i+1} will be removed from the set of proposals in P_i, to create P_{i+1}.

The fundamental difference between the *sci.* and the *eng.* tasks of the problem is that the first ones are subjected to maximizing with prior knowledge of required number of tasks, and the later is subject to minimization without the prior knowledge of the place or the required number of tasks. *Eng.* tasks of *Ariel* in HAP, are defined as single task proposals (p_e), and are placed in the schedule after the *sci.* tasks are processed. The initial window of p_e is set at the beginning of the survey. Once the exact place of the task within that window is determined by the CRUs, HAP in its Update module calculates the next appropriate window and creates a new proposal based on it. This cycles continues until the next window of a *eng.* tasks is after the end of the mission. This strategy solves the problem with unknown number of such tasks.

Table 4
Algorithm parameters for the HAP method.

Parameter	Notation	Value
Number of CRUs	k	10
CRU maximum iterations	max_{iter}	200
Stochastic range	sR	3

To handle their required minimization, CRU works in the opposite direction compared to *sci.* tasks. In contrary to the default approach, a CRU starts its search to put a *eng.* task in its window from the end of it. This prioritization allows us to emphasize on scheduling two consecutive *eng.* tasks as far as possible without violating their boundaries.

After all the different types of tasks are scheduled in this setup, HAP filters out the incompetence results by using a Pareto Optimal Front (PF) selection on F_T, F_C, and two other secondary fitness values that represent the quality of *eng.* tasks scheduling. First, is to minimize the number of *eng.* tasks of each type in the final schedule (F_E). Second to have the minimum number of violations from their boundaries (F_V). This detailed setup to handle the specific demands of *Ariel*, also gives an example of how HAP manages different types of tasks in one process, with a few small additions to its default modules. We finish this section with Table 4 that shows HAP algorithm parameters that are used in the tests.

4.4. Analysis of results

To have a better insight into the performance, while HC as a deterministic method requires one simulation per setup, tests on HAP and EA with stochastic mechanisms demand repetition. The results presented in this section for these approaches are aggregated from 25 simulations in each test. In the rest of this section, we first take a closer look at the effect of stochastic selection on the proposed HAP by comparing it with a single CRU and a deterministic version of itself in Section 4.4.1. After that, the comparison results between the different methods on the available datasets are presented. Section 4.4.2 details performance on the MRS dataset, followed by Section 4.4.3, which describes the results of MRS_B. In Section 4.4.4, the outcome of the tests on the datasets that include PC are represented, and finally, Section 4.4.5 contains a computational cost analysis.

4.4.1. HAP stochastic improvement

The stochastic selection in CRU allows HAP to obtain different results by running the same process, thereby exploring more areas of the solution space and providing a variety of solutions to the user. The stochastic selection is weighted toward the options with the lower *Cost* values as defined in Definition (3), and it is not restricted to the lowest one. This strategy helps CRU to not get stuck in a local minimum during its search, however, if only the lowest cost option is to be selected by choosing $sR = 1$, HAP still provides a single competent result. This specific deterministic setup of HAP, which does not use multi-start, and requires only one CRU is referred to as Accumulative Planner (AP), and in this section, we compare it with the HAP as described in Section 4.3. Table 5 displays the performance of AP and HAP on the available datasets.

Table 5, shows that HAP obtains a schedule with better F_C fitness in all the datasets. The value of F_C as the main objective of the problem, represents the scientific return of *Ariel* and it, therefore, receives the most attention in evaluating a schedule, and the selected solutions of HAP for different datasets presented in Table 5 are the best schedule based on it. The solution with the highest F_C does not necessarily provide the best values for the other objectives of the problem. For example, AP on the MRS_B and MRS_B^{PC} datasets completes more proposals and spends more time on *sci. tasks* than HAP, as indicated in columns 3 and 5 in Table 5, respectively. Among all the solutions provided by HAP, they achieve a higher value than AP in all the objectives

Table 5
Comparison of HAP with AP (deterministic single iteration version).

Dataset	Method	Comp.	Obs.	Sci.time (%)	T4	T3	T2	T1	F_C (%)
MRS	AP	998	3016	69.59	0	50	550	398	99.97
	HAP	999	3020	69.97	0	50	550	399	**100**
MRS_B	AP	1507	3035	78.38	0	50	290	1167	76.14
	HAP	1495	3082	78.43	0	50	316	1129	**77.12**
MRS^{PC}	AP	980	2794	75.09	43	50	484	403	95.95
	HAP	994	2762	74.70	43	50	497	404	**96.77**
MRS_B^{PC}	AP	1379	2657	78.32	43	50	289	997	78.77
	HAP	1367	2713	78.50	43	50	315	959	**79.53**

Table 6
Comparison of algorithms performance for the MRS dataset.

Method	# sci.	Tiers			Times (%)		F_C
	proposals	3	2	1	*sci.*	*eng.*	(%)
HAP	998 ± 1	50	550	398 ± 1	69.66 ± 0.31	3.51 ± 0.3	$\mathbf{99.97 \pm 0.03}$
EA	972 ± 3	50	532 ± 3	390 ± 4	65.9 ± 0.4	4.1	98.23 ± 0.35
HC	992	50	534	408	67.79	5.25	98.86

similar to what is shown in the table for F_C. However, the difference, especially in small datasets, is small. AP as a deterministic approach, with a reduced computational cost, provides good reproducible results.

In order to put the results shown in Table 5 into perspective, the F_c values are used in the statistical analysis. Considering the number of available methods and datasets, Student's t-test (De Winter, 2013), was selected for the analysis. T-test comparison between HAP and AP, reveals a p-value of 0.054, which is significant with 90% confidence, however, note that it is very close to the 95% threshold. It is worth mentioning that for MRS and MRS_{PC}, where the F_c values are close to the maximum, the stochastic improvement of HAP is small, yet effective.

Observation 1. *AP as a deterministic approach provides a reliable and competent solution. While the stochastic process of HAP allows it to search more areas of the solution space and find better results, if the computational cost is limited, in order to improve the reliability of HAP, one of the CRUs can be configured as AP.*

4.4.2. MRS Dataset

The MRS dataset, represents the main goal of the *Ariel* mission. Assessing the performance of each method on it has the highest importance. Although scheduling all the requested tasks from MRS takes only 69.96% of the mission duration, strict constraints and a high number of overlaps in the windows makes it challenging to complete all the proposals. Table 6 shows the performance of the methods, on this dataset.

As appears in the second column of Table 6, out of 999 proposals, HAP is the only approach that is able to complete all the tasks in its solutions. This compares to 972 ± 3 by EA, and 991 by HC. According to this objective HAP has the best performance, followed by HC, and with EA in last place. The same ranking is true when comparing F_C fitness values, as shown in column 8. Although EA completes about the same number of Tier 2 proposals as HC (column 4), the greedy approach completes more Tier 1 proposals (column 5). The number of completed Tier 1 proposals for HC is higher than the initial number presented in the dataset. That is because some of the Tier 2 proposals which could not be completed are reintroduced as Tier 1 proposals with a lower number of required tasks and HC manages to put them in the schedule. This strategy is in effect for all the methods under evaluation.

According to Table 6, regarding the time spent on *sci.* tasks as indicated in column 6, HAP still obtains the best result, while HC holds second place, and EA follows the other methods. However, EA assigns less time to calibrations compared to HC. There are several reasons explaining these results. The main cause is the way of handling both *sci.*, and *eng.* tasks in one schedule. In HC, *eng.* tasks are introduced as normal ones with a medium to high priority and one window. The scheduler forces an *eng.* task into the schedule if it is reaching the end of its allowed interval. This strategy ensures these tasks are scheduled with a certain degree of flexibility. HC does not try to minimize the number of *eng.* tasks, resulting in the highest time spent on them (5.25%) in comparison. However in this way HC still manages to outperform EA in all the other categories, due to the fact that EA's strategy is to separate the scheduling of *sci.* and *eng.* tasks. While this means lesser time spent on operations (4.1%) compared to HC, their scheduling prevents the method from completing more targets as they would be scheduled on top of the operation tasks. The other reason for this performance is the low complexity of the dataset. Minimum assumptions and directions on the solutions in EA, on a non-congested schedule with strict constraints, decreases the performance of this approach. On the other hand, smaller problems are more suitable for HC and the results in Table 6 back this up. HAP manages to outperform the other methods in all the fields described in the table. This includes the shortest time spent on operation tasks (3.62%). Table 7, takes a closer look at the details of the scheduled *eng.* tasks, among the results.

According to Table 7, HC and EA schedule more operation tasks than HAP, as shown in columns 2, 6, and 10. Columns 3, 7, and 11 reflect the main reason for this difference by presenting the average distance between two operation tasks of different types. The average distance in HC and EA is less than in HAP in all the tasks.

In the case of violations of boundaries which are indicated in the table in columns 5, 9, and 13, the weakest performance belongs to EA with 135 violations in short calibration (SC) scheduling, and seven in long calibration (LC). While EA is the only approach with LC violation, concretely for SC, HC, and HAP also break the boundaries on a few occasions. In the tests of HAP, there is a small margin assumed for calibrations to be changed after they are planned when there are no better options. As HAP puts *eng.* tasks as far away as possible, there is a minor chance that moving one later within the previously assigned window takes it out of range. This can easily be avoided within the process, although it was allowed under certain scenarios to increase the optimization opportunities of the *sci.* tasks. The violation of boundary for HAP max out is on average at 2.09 (column 4) which is less than ten percent of the maximum allowed of two days for short calibrations. HC exceeds more and has a maximum distance of 2.52. Finally, in EA, the maximum short calibration distance is about twice what is allowed at 3.93. However, the average distance between them is kept close to the SC cadence due to the larger number of tasks that are added to the schedule.

On the stochastic methods applied to MRS, namely HAP and EA, a paired sample t-test is performed for analyzing the statistical significance. The test uses the data produced by the 25 repetitions of scheduling by each method. The result shows that HAP is significantly better than EA on the MRS dataset with over 99% confidence, as in every instance it outperforms EA. The distribution of F_c values in the repeated tests on MRS, by HAP and EA, is demonstrated in a box plot in Fig. 3.

According to Fig. 3, the standard deviation of HAP results is smaller than EA, and its box indicates that there are more instances above the median result.

Observation 2. *For the MRS dataset, HAP consistently achieves the best results in all aspects and objectives. For the relatively small dataset of MRS where ideally all the proposals can be fitted into a schedule, the greedy approach of HC works better than the heavy global optimization that EA provides.*

Table 7
Detailed results for *eng.* tasks on MRS.

Method	Short calibration				Long calibration				Station keeping			
	Plan. tasks	Dist. (day) avg.	Dist. (day) max	Vio.	Plan. tasks	Dist. (day) avg.	Dist. (day) max	Vio.	Plan. tasks	Dist. (day) avg.	Dist. (day) max	Vio.
HC	997	1.24	2.52	5	58	21.81	28.25	0	67	18.92	29.83	0
EA	834	1.48	3.93	135	42	29.36	43.10	7	45	27.21	27.99	0
HAP	723.80	**1.73**	2.09	0.68	36.20	**34.75**	39.75	0	42.60	**29.50**	30.83	0

Table 8
Comparison of algorithms performance for the MRS_B dataset.

Method	Comp. proposals	Tiers 3	2	1	Times (%) *sci.*	*eng.*	F_C (%)
HAP	1488 ± 15	50	310 ± 6	1128 ± 16	77.9 ± 0.6	3.6 ± 0.1	**76.86 ± 0.32**
EA	1110 ± 18	50	399 ± 7	660 ± 11	68.9 ± 0.2	4.1	73.16 ± 0.57
HC	972	47	264	661	67.65	4.47	62.27

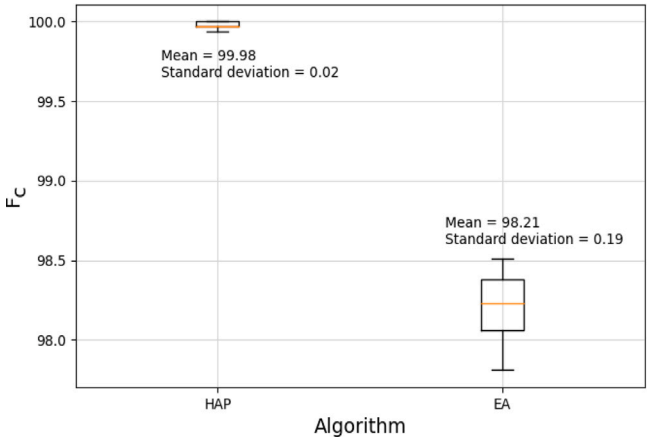

Fig. 3. A box plot of HAP compared to EA on MRS, for 25 repetitions of the test.

4.4.3. MRS_B Dataset

The second dataset that the performance of the methods in evaluation are tested on is MRS_B, which includes the main and the backup proposals. Backup proposals are added to the Tier 1 list, and their priority value is derived from fitness calculations. Unlike MRS, in order to complete all the proposals, MRS_B requires about twice as much time that it is available for *Ariel* (199.37%). This means that a scheduler must leave some proposals out of its solutions. This tips the favor toward EA, since it is more challenging for HC to handle. The results for these two methods along with HAP on MRS_B, are presented in Table 8 in order to give a perspective on their performance.

The most notable performance in Table 8 is for HAP which achieves better results than its rivals. With the same consistency as it had for MRS, HAP obtains better solutions in almost all the fields, except for completed Tier 2 proposals as indicated in column 4. EA completes more Tier 2 proposals than HAP, however, the number of completed Tier 1 shown in column 5 is significantly higher. Fitness value F_C in column 8, gives an overall value to the completed proposals of different tiers. A comparison on F_C shows that the competence of the schedules HAP constantly achieves higher values than EA and HC. Unlike with the MRS dataset, the results on MRS_B from EA are better than those by HC. The difference in competence in their results is notable as can be seen in Table 8.

The results from EA are competent, and there is a small difference between F_C values in EA and HAP. HC on the other hand struggles to keep up with the more sophisticated approaches. As HC cannot decide to remove any tasks that it has already scheduled, selecting proposals to leave out is problematic, and depends mainly on which one comes first. This leads to it finishing only 972 proposals which is not only lower than the other two methods but also lower than the number of completed targets for MRS. All the proposals in MRS also exist in MRS_B, and so this means that an over-populated list of proposals damages the performance of HC in completing proposals, and it is better to prune the dataset and select the more important proposals before passing it to the method. Another observation from Table 8 is on the time on *sci.* tasks in column 6. There is a significant difference between HAP and the two other methods.

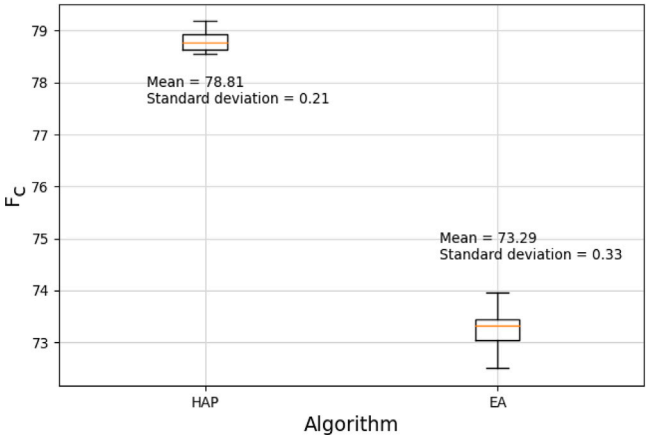

Fig. 4. A box plot of HAP compared to EA on MRS_b, for 25 repetitions of the test.

The scheduling results of *eng.* tasks for MRS_B are similar situation to those of MRS. Column 7 in Table 8 shows the time assigned to these tasks, and Table 9 details the quality of their scheduling.

Although EA has less time assigned to *eng.* tasks than HC, it has a higher number of violations in their boundaries as seen in Table 9. HC still exceeds the maximum allowed time in 101 situations for Short Calibrations (column 5), in two situations for Long Calibrations (column 9), and in one situation for Station Keeping (column 13). The most competent results are achieved by HAP, which has the lowest number of scheduled *eng.* tasks and no violations of boundaries.

Similar to MRS, a statistical analysis of paired sample t-test is performed to analyze the results obtained by HAP, compared to EA. The test on 25 repetitions indicates that HAP achieves significantly better results on MRS_b with over 99% confidence. This is because HAP consistently outperforms EA, as it is illustrated in Fig. 4.

Fig. 4 shows the results from HAP have a lower standard deviation than EA. This is similar to the plot for MRS (see Fig. 3), however, unlike that, the majority of the results are equally around the median. The figure also indicates that the maximum F_c of HAP is further away from the median than the minimum. This aspect improves by increasing the number of CRUs, which is the population of HAP, in each test.

Observation 3. *For the dataset MRS_B, like MRS, HAP provides the best results, but unlike that, EA takes over HC as the second competent method. HAP and EA both provide good results, however, HC falls behind the other two, on a larger dataset.*

Table 9
Detailed results for operation tasks on MRS_B.

Method	Short calibration				Long calibration				Station keeping			
	Plan. tasks	Dist. (day) avg.	Dist. (day) max	Vio.	Plan. tasks	Dist. (day) avg.	Dist. (day) max	Vio.	Plan. tasks	Dist. (day) avg.	Dist. (day) max	Vio.
HC	890	1.39	3.39	101	50	25.21	33.25	2	46	27.11	31.05	4
EA	832	1.49	3.72	137	42	29.26	49.55	4	45	27.21	28.03	0
HAP	747.50	1.67	1.96	0	34.80	35.97	39.75	0	43.20	29.19	30.83	0

Table 10
Comparison of HAP and HC performance in MRS^{PC} and MRS_B^{PC}.

Method	Comp. proposals	Tiers 4	3	2	1	Times (%) *sci.*	*eng.*	F_C (%)
MRS^{PC}								
HAP	986 ± 8	42.5 ± 0.5	50	493 ± 4	401.50 ± 10.5	74.42 ± 0.34	3.46 ± 0.04	96.57 ± 0.2
HC	964	43	50	504	367	75.02	4.61	96.52
MRS_B^{PC}								
HAP	1378 ± 14	42.5 ± 0.5	50	305 ± 10	982 ± 23	78.02 ± 0.9	3.52 ± 0.06	**79.09 ± 0.46**
HC	921	40	48	231	603	68.93	4.48	67.03

4.4.4. MRS^{PC} And MRS_B^{PC} datasets

The addition of abnormally long Phase Curve (PC) proposals to the MRS and MRS_B datasets brings extra challenges and some constraint relaxation. PC proposals were not initially considered in the problem definition and so after their introduction, the methods have to be adapted to the new conditions. The main difficulty of PC is the conflict with SC. Unlike other tasks, which usually have windows with a duration of a few hours, a PC requires a window duration in the order of days. As mentioned in Section 4.1, the SC task has to be repeated every 1.5 ± 0.5 day and therefore has to be an exception in the case of handling PCs, which have a longer duration than SC interval.

Both HAP and HC approaches can handle PC proposals without a structural change. This is due to their strategy of scheduling *eng.* tasks one at a time and within the same process as the other tasks. This gives them enough flexibility to delay SCs if a PC is scheduled. On the other hand, in EA, *eng.* tasks are scheduled separately from the *sci.* ones and before them. Thus, it is not possible to make an exception for the PC proposals in *eng.* task scheduling, when their place in the schedule has not been decided. The necessity to make a fundamental change to EA in order to be able to schedule PC tasks and the high development cost prevented us from testing EA on the datasets that included PC proposals. In *Ariel*, PC proposals have the same priority as Tier 3, and the goal is to schedule all 43 of them. So, HAP and HC, are the only methods that are tested with MRS^{PC} and MRS_B^{PC} datasets.

The selection strategy of HAP is set to pass PC proposals to CRUs after the rest of the *sci.* tasks and the *eng.* tasks. As PC has high priority and cost, if they are passed to CRUs early, they will occupy empty windows whose cost is unknown. For a cost-heavy PC proposal, it is better to delay their scheduling until the last, when most of the new windows have overlapped with the tasks in the schedule, and that determines their true cost. The amount of time that is required to complete all the proposals in MRS^{PC} is equivalent to 81.61% of the *Ariel* survey duration. This value for the MRS_B^{PC} dataset increases to 211.02%. The outcome of the tests on these datasets is presented in Table 10.

Table 10 shows that for dataset MRS^{PC}, HC creates a good solution. The F_C of this solution is slightly below a median schedule produced by HAP, yet it has completed more Tier 2 proposals (column 2) and assigned more time to *sci.* tasks (column 7). HAP still outperforms HC in the rest of the attributes and provides several solutions at a time. A good solution obtained by HAP achieves a distinctively better F_C than HC. This is mainly due to the fact that HAP completes a higher number of Tier 1 proposals.

On the other hand, for MRS_B^{PC}, HC performance drops significantly. If we compare this decrease with the one discussed in Sections 4.4.2 and 4.4.3, from MRS to MRS_B, it is much more noticeable. On the expanded dataset of MRS_B^{PC}, HC completes 5% fewer proposals than MRS^{PC} and spends 6.09% less time on the *sci.* tasks. In relative terms, HAP completes about 33% more proposals and covers 9.09% more of time with *sci.* tasks. The competence difference between the solutions provided by HAP and HC in, HAP_B^{PC}, in the most complex dataset, is severe. This significance is reflected in their corresponding F_C from the last column of Table 10, where the HC schedule obtains a 12.87% lower F_C value than a median schedule by HAP.

Table 11 details the results of operation tasks scheduling on MRS^{PC} data. The first thing to note is the number of short calibration (SC) violations in both methods. As expected, the methods violated SC intervals when, a PC is scheduled. According to column 5 of the table, while HC for MRS^{PC} has violated the boundaries of SC the same number of times as it has scheduled PC (43 times), HAP still manages to schedule the shortest PC proposals without violating SC tasks (37.9 times on average). Other than that, like on the rest of the tests, HAP schedules fewer *eng.* tasks with more distance and with lower violations.

Observation 4. *HAP is able to adapt to dynamic environments while maintaining its good solution competence. This is more significant on the most complex dataset, where EA could not be adapted, and HC obtains significantly worse results.*

4.4.5. Computational cost

So far, we have evaluated HAP and the other methods by their solution competence. However, one major factor in the performance comparison is the computational cost or the execution time, which in most cases determines the limits of the search in the solution space by different approaches. Currently, there is no maximum execution time for *Ariel*, however, a limit will be defined in the later development phases of the problem. Table 12 demonstrates the computational cost of HAP and its special case, AP, along with HC, and EA, on the different datasets under evaluation. The configuration of each method is the same as that described in Section 4.3. The computer that the tests were performed on is equipped with a 3.2 GHz, 8-core CPU, and 16 GB of RAM.

The results for AP are obtained using a single CPU core. While the computational cost of AP can be reduced by multiprocessing, as in HAP implementation, this setup also shows the minimum execution time of HAP. Based on the available processing cores and desired computational cost we can calculate the number of necessary CRUs in HAP.

Table 11
Detailed results for operation tasks on MRS^{PC} and MRS_B^{PC}.

Method	Short calibration				Long calibration				Station keeping			
	Plan. tasks	Dist. (day) avg.	Dist. (day) max	Vio.	Plan. tasks	Dist. (day) avg.	Dist. (day) max	Vio.	Plan. tasks	Dist. (day) avg.	Dist. (day) max	Vio.
MRS^{PC}												
HC	887	1.40	9.12	43	56	22.61	30.00	0	48	26.38	30.17	0
HAP	676.90	1.85	9.53	37.90	36.30	34.65	39.75	0	42.70	29.50	30.83	0
MRS_B^{PC}												
HC	824	1.51	9.53	62	50	25.21	37.25	0	63	20.14	32.02	2
HAP	699.80	1.79	9.56	35.40	35.00	35.62	39.75	0	43.00	29.25	30.83	0

As can be seen in Table 12, HAP significantly outperforms EA in the execution time for the two comparable datasets. While it is expected that an EA should have a higher computational cost than the other two methods under evaluation, these results place this difference into a certain perspective. According to the design of the EA, its execution time can be divided into two separate parts: *eng.* task and *sci.* task scheduling. For MRS dataset, EA spends 1800 s on scheduling the *eng.* tasks and only 240 s for *sci.* tasks. In MRS_B datasets with twice the number of proposals compared to MRS, *eng.* tasks take 3636 s to schedule, while *sci.* tasks require 600 s. These values show that in the EA under evaluation, most of the execution time is spent on processing *eng.* tasks. The scheduling of *sci.* tasks are relatively fast. The reason for this difference, despite the fact that there are more *sci.* tasks, is that while the scheduling of *sci. tasks* is handled as a permutation of different tasks, for *eng.* tasks, EA assumes every minute for the possible start of calibration for the whole duration of the *Ariel* mission (3.5 years).

Compared to HC however, while HAP is faster in the smallest dataset (MRS), for the rest of the datasets it has a higher computational cost. The lower cost of HC has a huge impact on its performance on more complex datasets, as indicated in Sections 4.4.3 and 4.4.4.

On average, the computational cost of HAP stands between HC and EA. It is more sensitive to the number of input proposals than the other methods, however even on the largest dataset, it has a significantly lower computational cost than EA. The main reason for the great leap in execution time for HAP is that the average time spent on processing a proposal by CRUs depends on the available empty spots in the schedule. On a sparse schedule, it takes fewer CRU cycles to empty its Lobby and finish the process, rather than when the schedule is almost full when it usually takes the maximum number of cycles. For the MRS and MRS^{PC} datasets, the coverage time of all the proposals is less than the available time, so the schedule will never be full and HAP works with a low computational cost. On the other hand, for the MRS_B and MRS_B^{PC} datasets, with a coverage time twice as much as the available time, HAP works on a full schedule.

AP, as the smallest HAP, has the better timing overall, with only 3.02 s for MRS, and 233.49 s for MRS_B^{PC}. However, it should be taken into account that the tested HAP has 10 CRU, thus exploring ten times more than AP in the solution space.

Observation 5. *Compared to the other methods in evaluation, the computational cost of HAP is more input dependent. As the schedule becomes more congested, HAP takes longer time to process a proposal. With that in mind, HAP has less computational cost than HC for the smallest dataset, and it is significantly faster than EA, even on the largest dataset.*

4.5. Performance summary

The *Ariel* mission, as a developing project, has seen several expansions to its initial definitions. This is noticeable first in the difference between the MRS and the MRS_B in the number of proposals and second in the differences between the normal and PC tasks. PC tasks are added later to the problem, so the existing methods had to be adapted in line with the changes. While the EA method fails to handle them without major changes to its structure, HAP and HC are able to adapt to the new conditions. On the other hand, HC performance dropped significantly when the number of proposals doubled in MRS_B, compared to the initial proposal set in MRS, whereas both HAP and EA managed that situation well.

Table 12
Comparison of the computational cost (in seconds) of the methods.

Method	MRS	MRS_B	MRS^{PC}	MRS_B^{PC}
HAP	12.09	561.42	69.78	602.76
AP	3.02	230.82	13.34	233.49
HC	20.31	31.97	18.90	30.94
EA	2040	4236	–	–

HAP was able to adapt to the new demands with minimum changes in all situations and consistently outputs the best results. This reliability was not present in the other approaches in the evaluation. These characteristics of HAP satisfy the main objectives of the proposal. The architecture of HAP allows for the definition of different types of tasks and proposals, independently, and while performing scheduling optimization, processes all of them together. On the other hand, regarding the computational cost of HAP, although it is more input dependent than the other methods in comparison, it still takes less time than EA for all the datasets, and it is even faster than HC in smaller datasets. The input-dependent computational cost of HAP is beneficial for a simulation tool, where the user is able to apply and test many small changes without spending too much time. It also facilitates the adaptation of HAP to more extreme time constraints.

5. Conclusion

The inter-disciplinary scheduling problem that is investigated in this paper, requires a flexible and adaptive solution. We demonstrated the capabilities of the HAP proposal, by adapting the algorithm to the *Ariel* mission scheduling problem and its various setups. The quality of a solution is determined by the fitness values defined in the objectives of the problem. The evaluation on the performance of HAP and other available methods for *Ariel* shows that, while it has a much smaller computational cost than an EA, even on the largest dataset, it produces better solutions with higher quality and it offers wider customizability to adapt to different setups. Although HAP computational cost on more complex tests is larger than a greedy HC, the quality of the results from HC is significantly lower, which justifies the extra time needed for HAP. The HAP algorithm is defined based on the general definitions of the problem and provides a good balance between strict solution creation in local optimizations like greedy approaches, and the more free generation of solutions in global optimization algorithms like EA. This is done by introducing parts of the problem knowledge to guide its metaheuristics in its search in the solution space. In many cases, the

complexity of the problem is not enough to justify the use of a fully global optimization approach, especially when there are a high number of hard and soft constraints, which makes avoiding searching in invalid sections of the solution space more costly.

The consistency of HAP results in outperforming the other approaches and its flexibility makes it a suitable choice for users to test and simulate different scenarios and to adapt to new conditions presented by the problem. This paper has been written for researchers in both the fields of computer science and astrophysics. *Ariel* is a good example that demonstrates the steps required to adapt an algorithm to a real-world problem. We propose a reliable approach to the telescope scheduling problem and provide an in-depth analysis of the specifics of the problem and the performance of different methods on it. We consider that HAP is defined generally enough to be easily adapted to scheduling problems.

The are several directions to continue further with this study. The main path is to adapt HAP to other variations of the telescope scheduling problem to get more insight into its performance, adaptation, limitations, and into the problem itself. Between a simple metaheuristic and a Swarm Intelligence (SI) algorithm, HAP stands as a multi-start algorithm. There are different ways to investigate improving HAP by using the strategies from the SI field, especially from the Ant Colony Optimization (ACO), without losing the customizability of the current design. For instance, experimenting with incorporating a value like a pheromone, defined in ACO, to increase the exploitation of HAP would be beneficial.

Another path for future work is to further study the effects of the order in which the proposals are processed by CRUs. HAP configures each of its CRUs independently and this can be used to diversify its solutions by re-ordering the input for each CRU.

CRediT authorship contribution statement

Nariman Nakhjiri: Conceptualization, Investigation, Software, Data curation, Writing – original draft, Writing – review & editing. **Maria Salamó:** Conceptualization, Investigation, Supervision, Writing – review & editing, Resources in computer science. **Miquel Sànchez-Marrè:** Conceptualization, Investigation, Supervision, Writing – review & editing, Resources in computer science. **Juan Carlos Morales:** Investigation, Supervision, Data curation, Writing – review & editing, Resources in astrophysics.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

References

Blum, C., Li, X., 2008. Swarm intelligence in optimization. In: Swarm Intelligence. Springer, pp. 43–85.

Blum, C., Roli, A., Sampels, M., 2008. Hybrid Metaheuristics: An Emerging Approach to Optimization, Vol. 114. Springer.

Braekers, K., Ramaekers, K., Van Nieuwenhuyse, I., 2016. The vehicle routing problem: State of the art classification and review. Comput. Ind. Eng. 99, 300–313.

Chou, Y.-H., Kuo, S.-Y., Chen, C.-Y., Chao, H.-C., 2014. A rule-based dynamic decision-making stock trading system based on quantum-inspired tabu search algorithm. IEEE Access 2, 883–896.

Colomé, J., Casteels, K., Ribas, I., Francisco, X., 2010. The TJO-OAdM robotic observatory: the scheduler. In: Software and Cyberinfrastructure for Astronomy, Vol. 7740. International Society for Optics and Photonics, p. 77403K.

De Winter, J.C., 2013. Using the Student's t-test with extremely small sample sizes. Pract. Assess. Res. Eval. 18 (1), 10.

Deb, K., 2015. Multi-objective evolutionary algorithms. In: Springer Handbook of Computational Intelligence. Springer, pp. 995–1015.

Delahaye, D., Chaimatanan, S., Mongeau, M., 2019. Simulated annealing: From basics to applications. In: Handbook of Metaheuristics. Springer, pp. 1–35.

Dorigo, M., Socha, K., 2018. An introduction to ant colony optimization. In: Handbook of Approximation Algorithms and Metaheuristics, second ed. Chapman and Hall/CRC, pp. 395–408.

Edwards, B., Mugnai, L., Tinetti, G., Pascale, E., Sarkar, S., 2019. An updated study of potential targets for ariel. Astron. J. 157 (6), 242.

Garcia-Piquer, A., Morales, J.C., Colomé, J., Ribas, I., 2017a. Evolutionary Computation for the ARIEL mission planning tool. In: 2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT). IEEE, pp. 101–106.

Garcia-Piquer, A., Morales, J., Ribas, I., Colomé, J., Guàrdia, J., Perger, M., Caballero, J.A., Cortés-Contreras, M., Jeffers, S., Reiners, A., et al., 2017b. Efficient scheduling of astronomical observations-Application to the CARMENES radial-velocity survey. Astron. Astrophys. 604, A87.

Giuliano, M.E., Johnston, M.D., 2008. Multi-objective evolutionary algorithms for scheduling the James Webb Space Telescope. In: ICAPS. pp. 107–115.

Glover, F., Sörensen, K., 2015. Metaheuristics. Scholarpedia 10 (4), 6532.

György, A., Kocsis, L., 2011. Efficient multi-start strategies for local search algorithms. J. Artificial Intelligence Res. 41, 407–444.

Han, Z., Liu, S., Yu, F., Zhang, X., Zhang, G., 2017. A 3D measuring path planning strategy for intelligent CMMs based on an improved ant colony algorithm. Int. J. Adv. Manuf. Technol. 93, 1487–1497.

Hansen, P., Mladenović, N., Brimberg, J., Pérez, J.A.M., 2019. Variable neighborhood search. In: Handbook of Metaheuristics. Springer, pp. 57–97.

Jemai, M., Dimassi, S., Ouni, B., Mtibaa, A., 2017. A metaheuristic based on the tabu search for hardware-software partitioning. Turk. J. Electr. Eng. Comput. Sci. 25 (2), 901–912.

Johnston, M.D., 1988. Automated observation scheduling for the VLT. In: Very Large Telescopes and their Instrumentation, Vol. 2. p. 1273, vol. 30.

Johnston, M.D., 1990. Spike: Ai scheduling for nasa's hubble space telescope. In: Sixth Conference on Artificial Intelligence for Applications. IEEE Computer Society, pp. 184–185.

Katoch, S., Chauhan, S.S., Kumar, V., 2021. A review on genetic algorithm: past, present, and future. Multimedia Tools Appl. 80, 8091–8126.

Kessaci, Y., Melab, N., Talbi, E.-G., 2014. A multi-start local search heuristic for an energy efficient VMs assignment on top of the OpenNebula cloud manager. Future Gener. Comput. Syst. 36, 237–256.

Kovacs, A.A., Golden, B.L., Hartl, R.F., Parragh, S.N., 2015. The generalized consistent vehicle routing problem. Transp. Sci. 49 (4), 796–816.

Kumar, V.S., Thansekhar, M., Saravanan, R., Amali, S.M.J., 2014. Solving multi-objective vehicle routing problem with time windows by FAGA. Procedia Eng. 97, 2176–2185.

Laguna, M., 2018. Tabu search. In: Handbook of Heuristics. Springer, pp. 741–758.

Lange, J., Werner, F., 2019. On neighborhood structures and repair techniques for blocking job shop scheduling problems. Algorithms 12 (11).

Li, R., Hu, S., Liu, H., Li, R., Ouyang, D., Yin, M., 2019. Multi-start local search algorithm for the minimum connected dominating set problems. Mathematics 7 (12), 1173.

Martí, R., Aceves, R., León, M.T., Moreno-Vega, J.M., Duarte, A., 2019. Intelligent multi-start methods. In: Handbook of Metaheuristics. Springer, pp. 221–243.

Martí, R., Lozano, J.A., Mendiburu, A., Hernando, L., 2018. Multi-start methods. In: Handbook of Heuristics. Springer, pp. 155–175.

Masmoudi, M.A., Braekers, K., Masmoudi, M., Dammak, A., 2017. A hybrid genetic algorithm for the heterogeneous dial-a-ride problem. Comput. Oper. Res. 81, 1–13.

Moisana, S., Boërb, M., Thiebautb, C., Tricoirea, F., Thonnata, M., 2002. A versatile scheduler for automatic telescopes. In: Proceedings, Vol. 4844.

Mora, M., Solar, M., 2010. A survey on the dynamic scheduling problem in astronomical observations. In: IFIP International Conference on Artificial Intelligence in Theory and Practice. Springer, pp. 111–120.

Pardalos, P.M., Du, D.-Z., Graham, R.L., 2013. Handbook of Combinatorial Optimization. Springer.

Parpinelli, R.S., Lopes, H.S., 2011. New inspirations in swarm intelligence: a survey. Int. J. Bio-Inspired Comput. 3 (1), 1–16.

Puig, L., Pilbratt, G., Heske, A., Escudero, I., et al., 2018. The phase a study of the ESA M4 mission candidate ARIEL. Exp. Astron. 46 (1), 211–239.

Raffi, G., Chiozzi, G., Glendenning, B., 2002. The ALMA common software as a basis for a distributed software development. In: Astronomical Data Analysis Software and Systems XI, Vol. 281. p. 103.

Razali, N.M., 2015. An efficient genetic algorithm for large scale vehicle routing problem subject to precedence constraints. Procedia-Soc. Behav. Sci. 195, 1922–1931.

Saidi-Mehrabad, M., Fattahi, P., 2007. Flexible job shop scheduling with tabu search algorithms. Int. J. Adv. Manuf. Technol. 32 (5–6), 563–570.

Sasaki, T., Kosugi, G., Kawai, J.A., et al., 2000. Observation scheduling scheme for the Subaru telescope. In: Advanced Telescope and Instrumentation Control Software, Vol. 4009. International Society for Optics and Photonics, pp. 350–354.

Schneider, M., Stenger, A., Goeke, D., 2014. The electric vehicle-routing problem with time windows and recharging stations. Transp. Sci. 48 (4), 500–520.

Song, A., Chen, W.-N., Gu, T., Zhang, H., Zhang, J., 2019. A constructive particle swarm optimizer for virtual network embedding. IEEE Trans. Netw. Sci. Eng. 7 (3), 1406–1420.

Tartan, E.O., Erdem, H., Berkol, A., 2014. Optimization of waiting and journey time in group elevator system using genetic algorithm. In: 2014 IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA) Proceedings. IEEE, pp. 361–367.

Tinetti, G., Drossart, P., Eccleston, P., et al., 2018. A chemical survey of exoplanets with ARIEL. Exp. Astron. 46 (1), 135–209.

van Rooyen, R., Maartens, D.S., Martinez, P., 2018. Autonomous observation scheduling in astronomy. In: Observatory Operations: Strategies, Processes, and Systems VII, Vol. 10704. SPIE, pp. 393–408.

Wang, D., Tan, D., Liu, L., 2018. Particle swarm optimization algorithm: an overview. Soft Comput. 22, 387–408.

William, H., Xin, M., 2018. The state-of-the-art integrations and applications of the analytic hierarchy process. European J. Oper. Res. 267, 399–414.

Yáñez, J., 2003. Optimization of telescope scheduling-Algorithmic research and scientific policy. Astron. Astrophys. 403 (1), 357–367.