IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 September 2024, accepted 22 October 2024, date of publication 5 November 2024, date of current version 26 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3492100

RESEARCH ARTICLE

Forgetful Swarm Optimization for Astronomical Observation Scheduling

NARIMAN NAKHJIRI[1,2,3], MARIA SALAMÓ[1,4], MIQUEL SÀNCHEZ-MARRÈ[5], CHRISTIAN BLUM[6], AND JUAN CARLOS MORALES[2,3]

[1]Faculty of Mathematics and Computer Science, Universitat de Barcelona (UB), 08007 Barcelona, Spain
[2]Institute of Space Sciences (ICE, CSIC), Campus UAB, 08193 Bellaterra, Spain
[3]Institute of Space Studies of Catalonia (IEEC), Campus PMT-UPC, 08860 Castelldefels, Spain
[4]Institute of Complex Systems (UBICS), Universitat de Barcelona (UB), 08007 Barcelona, Spain
[5]Intelligent Data Science and Artificial Intelligence Research Centre (IDEAI-UPC), Department of Computer Science, Universitat Politècnica de Catalunya (UPC), 08034 Barcelona, Spain
[6]Artificial Intelligence Research Institute (IIIA-CSIC), Campus UAB, 08193 Bellaterra, Spain

Corresponding author: Nariman Nakhjiri (nnakhjna21@alumnes.ub.edu)

This work was supported in part by the Ministerio de Ciencia e Innovación (MCIN)/Agencia Estatal de Investigación (AEI)/10.13039/501100011033 through Spanish Grants under Project PID2021-125627OB-C31, in part by the MCIN/AEI through "European Regional Development Fund (ERDF) A way of making Europe" under Grant PID2020-120375GB-I00, in part by the Programme Unidad de Excelencia María de Maeztu under Grant CEX2020-001058-M, and in part by the Generalitat de Catalunya/Centres de Recerca de Catalunya (CERCA) Programme. The work of Maria Salamó was supported by MCIN/AEI/10.13039/501100011033/ Fondo Europeo de Desarrollo Regional (FEDER), European Union (UE) under Grant PID2021-124361OB-C33; and in part by the Catalan Agency of University and Research Grants (AGAUR), Generalitat de Catalunya under Grant 2021 SGR 00313. The work of Miquel Sànchez-Marrè was supported by the Consolidated Research Group "Intelligent Data Science and Artificial Intelligence Research Centre (IDEAI-UPC)" from AGAUR, Generalitat de Catalunya under Grant 2021 SGR 01532.

ABSTRACT In this paper, we propose a novel metaheuristic algorithm called Forgetful Swarm Optimization (FSO) for Astronomical Observation Scheduling (AOS), a type of combinatorial optimization problem defined by the tasks and constraints assigned to the telescopes and other devices involved in astrophysical research. FSO combines local optimization, Destroy and Repair, and Swarm Intelligence methodologies to create a flexible and scalable global optimization algorithm to handle the challenges of AOS. The proposal is adapted to the well-justified scenarios of the Ariel Space Mission problem, a particular example of AOS, and compared with previous algorithms that are applied to it including an Evolutionary Algorithm (EA), an Iterated Local Search (ILS), a multi-start metaheuristic, a Tabu Search, and a Hill-Climbing greedy algorithm. The experimental evaluation demonstrates that FSO consistently outperforms other algorithms in objective completeness, up to 8.4% on average, for all instances of the problem regardless of dimensions and complexity. Additionally, it has significantly less computational cost than ILS and the base models of a global optimization algorithm such as EA.

INDEX TERMS Metaheuristics, combinatorial optimization, swarm intelligence, destroy and repair, telescope scheduling.

I. INTRODUCTION

Combinatorial Optimization (CO) is the process of finding an optimal object, usually, a subset or permutation, from a finite set [1]. A particular CO problem, with a variety of constraints and tasks, is Astronomical Observation Scheduling (AOS) that applies to scheduling single or multiple telescopes to maximize the scientific return of the involved devices in a mission or project [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Daniel Augusto Ribeiro Chaves.

The schedule in AOS is a sequence of non-overlapping tasks with their execution times. A task (t) is generally the observation of a celestial object, but other activities such as maintenance operations of the facilities or auxiliary calibration observations should be commonly taken into account. Each task is defined by their *type*, their *duration*, *astrophysical attributes* (such as coordinates and orbital period), and *task constraints* (C_t). These parameters determine the possible periods of time (*windows*) in which it can be executed. The smallest unit of scientific value in AOS however, is not an individual task, but a set of

dependent ones referred to as a Scheduling Block (B). Only if all the dependencies in a scheduling block, B, are resolved, it contributes to the scientific return of a schedule, proportionally to its assigned priority value (pri_{B}).

In general, the constraints in AOS are categorized into two types depending on the level they affect: *task constraints* (C_t) which are applied to individual tasks, and *scheduling block constraints* (C_{B}) which are the dependencies between the tasks in a scheduling block. As mentioned, C_t is used to calculate the task windows and could be processed before the scheduling. For instance, a minimum required elevation of an object, or a maximum sky brightness for observation. On the other hand, C_{B} defines the rules for scheduling the tasks in B to increase the scientific return. Examples of C_{B} include simple logical relations and set operators such as intersection or union, and attribute-specific relations between tasks, like cadence and precedence. With C_{B}, it is also possible to define different patterns to schedule a scheduling block. For example, with XOR assigned with different priorities on its alternatives.[1] In addition, the number of tasks in a B can be fixed or be determined by the constraints during the scheduling (e.g. periodic tasks with minimum and maximum interval times). It is important to note that the scheduling block structure and its diverse range of constraints, make it difficult to draw parallels with AOS and other more well-known examples of CO.

Considering the base concepts, the AOS problem is defined as follows. The input is a set of scheduling blocks ($\mathcal{B}_{in}$) and a base schedule that might be empty or contain some scheduled tasks. The output is a schedule (s) that consists of some or all the scheduling blocks in $\mathcal{B}_{in}$ with their execution times. The objectives in AOS vary depending on the specific problem. However, in general, apart from satisfying the constraints, there are two main objectives: **(1)** to propose a schedule that maximizes the accumulated priority of scheduling blocks in the schedule without breaking any C_t or C_{B}; **(2)** to minimize the idle time of the involved devices.

Besides the diversity of scheduling blocks and the high number of special cases, which severely limit the competent and valid results in the solution space and increase the computational cost of their evaluation, there are two main challenges that contribute to AOS difficulty:

1) *Scalability challenge*: The scheduler of AOS should be able to handle different instances of the same problem (scenario) that can significantly vary in dimension, complexity, and available computational time.
2) *Flexibility challenge*: Introducing new tasks and constraints is not rare in AOS. Thus the scheduler should be able to take on new conditions without going into major or structural changes.

[1] consider the example of a scheduling block consisting of the three tasks t_1, t_2, and t_3. A scheduling block constraint C_{B}: $(t_1 \oplus (t_2 \wedge t_3))$, with priority p_1 if t_1 is scheduled, and p_2 if t_2 and t_3 are both scheduled. In this scenario, the scheduler tries to schedule only one of the XOR alternatives, preferably the one with the higher priority.

Researchers have addressed these challenges in two main ways. Some proceed with one method that suits the most crucial scenarios of the problem and compromise on marginal ones, whereas others develop several scheduling methods in a system for different needs. The first approach facilitates handling the flexibility challenge, but it might lack the optimization power in less common scenarios. On the other hand, the second approach excels at the scalability challenge, but it is more difficult to adapt it to new conditions. Research into a single-method approach that is flexible for all scenarios is more beneficial in the long term, as it is easier to maintain and update. In the literature, the solutions to AOS are widely tailored to a specific type of problem or project, which can make it challenging to find a universally applicable and reusable solution.

In this paper, to address the challenges of AOS, we propose a single algorithm approach with the FSO algorithm to handle different variants of the problem. FSO combines the Swarm Intelligence (SI) [3] framework with a Destroy and Repair (D&R) strategy to utilize a constructive metaheuristic with a local search and create a scalable algorithm with general concept definitions. The algorithm uses the fundamental concept of SI as through interactions of rather simple agents, like sharing some information or experiences directly or via the environment, the emergent collective behavior is capable of solving more complex problems. The constructive metaheuristic of FSO, which progressively builds a solution from an initial state by adding components iteratively until the full solution is formed, applies a local search to individual task scheduling, to ensure the satisfaction of related C_t and C_{B}, and the combined SI framework optimizes the schedule on the objectives and high-level constraints. FSO competence in complex scenarios is reinforced by high-level exploration and exploitation in SI methodology, and in less elaborate or smaller scenarios, by its metaheuristic and local search. The combination of SI and local search in FSO aims to combine the benefits of the two approaches and create a scalable and flexible algorithm. In summary, the contribution of this paper is as follows:

1) Propose a flexible and scalable algorithm for AOS.
2) Introduce novel Swarm Intelligence interaction to improve exploration and exploitation of interesting parts of the solution space.
3) Demonstration of performance and adaptability in well-founded real problem scenarios, along with their associated challenges.
4) Comparison with different algorithms and methodologies.

The proposal is evaluated on different scenarios of a real example of the AOS problem, and its performance is compared with the other approaches that are specifically designed for this problem. These approaches are an Evolutionary Algorithm (EA) [4], a hybrid metaheuristic [5], an ILS [6], A Tabu Search [7], and a Greedy Hill-Climbing algorithm. The value of evaluating real problem scenarios lies

in their complexity. In contrast to generic problems, which are typically well-defined and stable, real cases often consist of a combination of challenges, leading to disparity between them and generic solutions. Testing the methods on real problems provides insight into how this gap is closed in different ways, and how the final outputs compare.

II. STATE-OF-THE-ART

The optimization problems can be divided into two main categories, *Combinatorial* and *Continuous* [8]. While combinatorial optimization is defined on discrete possibilities, usually subset or permutation, continuous optimization applies to continuous domains and often real numbers. The algorithms applied to the problems in these categories share many similarities, but their fundamental difference in the structure of the solution spaces require distinct approaches. This section reviews the solutions to the combinatorial optimization problems that are relevant to AOS, and details the methodologies related to FSO mainly as part of *metaheuristic* research [9].

In the literature, different approaches are applied to solve the AOS problem. Although *metaheuristics* are more commonly used, there are successful applications of other methods. For example, to schedule for the Hubble Space Telescope [10], a constraint-satisfaction system supported by artificial neural networks is used, which is derived from an earlier version proposed for the Very Large Telescope [11]. This approach is made possible by exploiting the heavily constrained environment to break down the search and scheduling into small decision rules. The same system is later transformed and adapted to other projects with heavily constrained environments, such as the Subaru Telescope [12].

Metaheuristics in general, have some advantages over direct optimization in problems with high dimensions and non-linear constraints [13] and find sufficiently good solutions in a limited time to problems that are too large to explore entirely [14]. Additionally, metaheuristics with few assumptions about the solution space facilitate their adaption to different problems [1]. These characteristics make them suitable for AOS and have led many researchers in this field to apply various metaheuristic algorithms to the problem with success, from less computationally expensive algorithms, such as Tabu Search [7] used to schedule the SOLSTICE instrument [15], to heavier population-based algorithms like MOEA [16] operating on James Webb Space Telescope (JWST) [17], Evolutionary Algorithm (EA) [18] on the LOFAR radio telescope [19], and Genetic Algorithm (GA) [20] on the CARMENES spectrograph [21] mounted on the telescope at the Calar Alto Observatory and the Canada France Hawaii Telescope (CFHT) [22].

The popularity of hybrid metaheuristics grows as the focus is shifted from *algorithm-based* research toward *problem-based* [23]. This approach exploits the good features of different algorithms to create competent solutions for specific problems with real-world complexity [24], [25]. Without hybrid algorithms, researchers use either a single algorithm that fits the most important scenarios of the problem [4], or break it down into *long-term* and *short-term* scheduling and use different algorithms depending on the scale of the scenarios. While *long-term* scheduling utilizes computationally heavy algorithms with high optimization capability to plan over a large period and large instances of the problem, *short-term* applies a fast algorithm with lower optimization power to the smaller scenarios. This approach is mainly used in ongoing projects where the initial long-term schedule must be updated nightly, based on the previously executed tasks [21], [26], and varying environmental (e.g. weather) conditions.

Metaheuristics can be categorized into *single-solution* algorithms, such as Simulated Annealing [27] and Variable Neighborhood Search [28], and *population-based* approaches [29], like algorithms in Evolutionary Computation (EC) [30] and Swarm Intelligence (SI). SI is a category of metaheuristics that is inspired by biological patterns and behavior in boosting success through iterations and is widely applied to a variety of CO problems. As population-based algorithms, they work based on the principle of creating initial solutions and gradually improving them by iteratively searching the solution space with a group of simple agents (Population or Swarm). In particular, the agents in SI algorithms regularly interact with the environment or each other, so that an improvement found by one affects the search of the others in future iterations. This interaction takes the form of pheromones in Ant Colony Optimization (ACO) [31], affects the velocity and position of particles in Particle Swarm Optimization (PSO) [32], and tips the odds on the selection of candidate solutions by onlooker bees in Artificial Bee Colony (ABC) [33], which are all well-known examples of SI.

Population-based global optimization algorithms, such as SI and EA in their base formats, depend on less specific knowledge about the solution space. Although this decreases the quality of individual solutions, it allows these algorithms to produce large numbers of solutions and optimize not only by the process of solution-making itself but also with the information gathered from the agents through iterations. Having a few assumptions about the solution space helps these algorithms in dealing with complex problems. However, their performance is not optimal on smaller problems due to the high computational cost and difficulty in converging on competent solutions. This is a long-known issue [34], which has led many studies to explore different approaches to handle it [35]. One approach to improve these algorithms' performance on small problems is to combine them with local search algorithms. Memetic Algorithms (MA) [36] is an example of such a hybrid approach that combines GA with local search, and outperforms the base version in certain types of problems. The local search used in MA is commonly designed to solve specific problems, objectives, and constraints, and it helps the GA incorporate more knowledge of the domain into its process [37]. This type of hybrid algorithm has global and local optimization characteristics, which

put it alongside well-known metaheuristics like Simulated Annealing and Tabu Search.

A well-known strategy that is utilized in metaheuristics is D&R, which prevents local search algorithms from getting stuck in local minimums and promotes exploration. D&R strategies are a form of ILS [6] that have been successfully applied to CO problems with close characteristics to AOS [38], [39]. Large Neighborhood Search (LNS) [40] is a good example of the algorithms that utilize these strategies with a diverse range of applications [41], [42] and academic studies [43], [44].

III. FORGETFUL SWARM OPTIMIZATION (FSO)

FSO is a population-based metaheuristic optimization algorithm consisting of a swarm ($\mathcal{A}$) of agents ($agent \in \mathcal{A}$) that use a *D&R* strategy iteratively (through generations) to optimize solution schedules. The specific form of *D&R* in FSO is referred to as Forget & Repair to distinguish it from the general strategy. The Forget strategy of FSO differs from the general D&R as it takes into account optimization parameters of FSO in removing parts of the intermediate solutions and includes modification on the environment to guide the next iterations. On the other hand, the Repair strategy of FSO uses a specific constructive metaheuristic, designed on the demands of AOS and its complexities.

The main FSO optimization parameters that set the pace of the algorithm include two values. First is the *forget probability*, which is denoted as λ_{max} ($0 \leq \lambda_{max} \leq 1$), and is initially assigned to every scheduling block, λ_B. Second, is the *decay rate* or δ ($0 \leq \delta \leq \lambda_{max}$), which is shared between the agents and is utilized for convergence. With these parameters, a brief description of the key features of FSO is as summarized in Figure 1.

Figure 1 illustrates a high-level description of FSO and its **Forget** strategy. In the initialization step, FSO creates the agents, sets the elite schedules, and adjusts the λ_B values. The elites ($\mathcal{S}_{elite}$) are the schedules with the highest fitness found at any point during the optimization process. Initially, $\mathcal{S}_{elite}$ is set with the input schedule which might be empty. The λ_B of scheduling blocks are set equal initially, with the predefined value, however, they are changed independently in the process. In a generation, FSO calls its agents' Forget and Repair processes to create new solutions to the problem, evaluates their results, and updates the elite schedules if necessary.

however, they decrease independently by the agents' **Forget** process. The λ_B reduction is applied on the forgotten set of every agent, whereas its increase is only on the forgotten sets of the successful agents. If **Forget** and **Repair** of an Agent leads to a schedule with higher fitness than its input, it is successful. The λ_B values decrease by a fixed value (δ), contrary to increment situations where the amount depends on the number of successful agents (in increasing the fitness) in that generation. Having fewer successful attempts results in a higher rise to λ_B. In this way, success is more impactful toward the end of the optimization when it is rare. Lowering

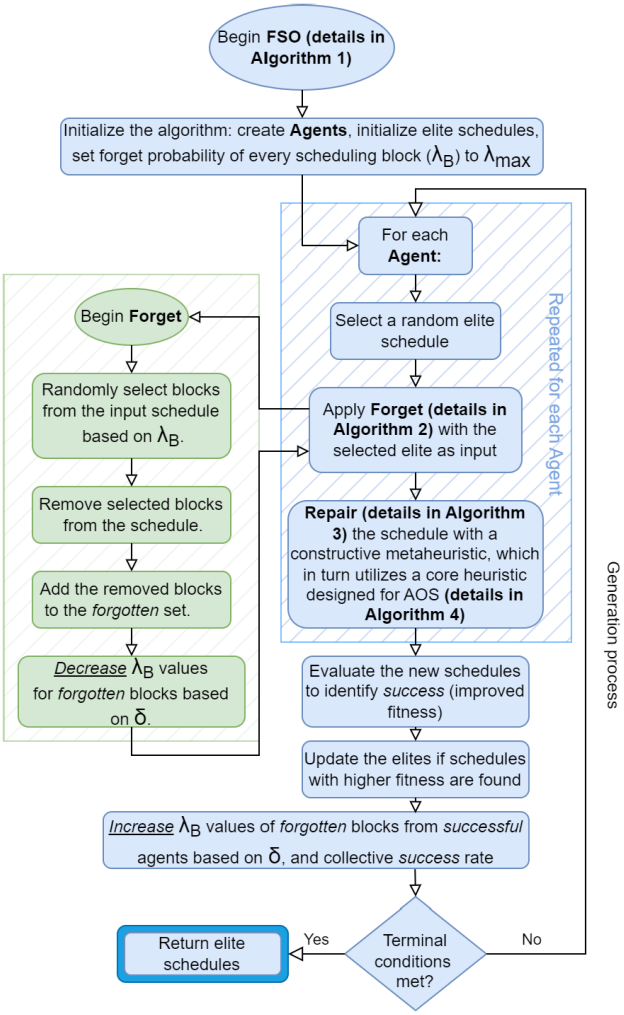

FIGURE 1. Flowchart of the FSO key features.

λ_B on each pass of an agent distributes the search evenly across different Bs to give them a fair chance at breaking into the schedule. On the other hand, the encouragement system increases the exploitation of successful attempts.

Figure 1 illustrates a high-level description of FSO and its **Forget** strategy. In the description, elite schedules ($\mathcal{S}_{elite}$) are the highest fitness schedules found at any point during the optimization process. Initially, $\mathcal{S}_{elite}$ is set with the input schedule which might be empty. The λ_B of scheduling blocks are set equal initially, however, they decrease independently by the agents' **Forget** process. The λ_B reduction is applied on the forgotten set of every agent, whereas its increase is only on the forgotten sets of the successful agents. If **Forget** and **Repair** of an Agent leads to a schedule with higher fitness than its input, it is successful. The λ_B values decrease by a fixed value (δ), contrary to increment situations where the amount depends on the number of successful agents (in increasing the fitness) in that generation. Having fewer successful attempts results in a higher rise to λ_B. In this way, success is more impactful toward the end of the optimization

when it is rare. Lowering λ_B on each pass of an agent distributes the search evenly across different Bs to give them a fair chance at breaking into the schedule. On the other hand, the encouragement system increases the exploitation of successful attempts.

FSO starts with a relatively high value λ_{max} and thus takes significant steps in the solution space during the early generations. This helps the algorithm focus more on exploration to find competent neighborhoods. The decrement strategy outweighs the increase, and the average forgetting probability in a schedule goes down over generations. Lower forgetting probability reduces the step size of FSO in later generations, which along with the higher impact of success, changes the algorithm focus to exploitation. FSO reaches the terminal conditions when in a generation, the average forgetting probability drops below a predefined threshold and no agent is successful in increasing the fitness of its input schedule.

Appendix describes the set of conventions for notations that are followed in this paper and includes Table 13, summarizing the definitions of the important denotations. The FSO algorithm is further explained in Section III-A (Algorithm 1), and to facilitate following its details, Section III-B describes the Forget (Algorithm 2) and Repair (Algorithm 3) of the agents. Finally, Section III-C describes the base heuristic of FSO (Algorithm 4), which the agents utilize to traverse the solution space.

A. FSO ALGORITHM

This section explains the FSO algorithm and the functions to assess the results of the swarm at each generation and manipulate the environment to guide the search. On the generic problem that is defined in Section I, Algorithm 1 describes FSO, where $x.y(z)$ means calling the y function of the object x with the input of z.

According to Algorithm 1, FSO takes a set of scheduling blocks ($\mathcal{B}_{in}$) and a base schedule (s_{base}) as input and starts by initializing the necessary variables. The set of elite schedules ($\mathcal{S}_{elite}$) is initialized with s_{base}. f_{avg} and f_{avg}^{prev}, which are the average fitness of the current and previous generation, are respectively set to the fitness of s_{base} and any value below that initially. λ_{avg} holds the average forgetting probability of the scheduling blocks in $\mathcal{S}_{elite}$ schedules and is initialized with λ_{max}. In line 2, the algorithm sets the forgetting probability of every input scheduling block to λ_{max} which is defined in the optimization parameters.

The main cycle of the algorithm from lines 3 to 16 represents a generation cycle and it iterates until the terminal conditions are met. Each generation begins with calling the Forget (details in Algorithm 2) and the Repair (details in Algorithm 3) functions of each agent of the swarm ($\mathcal{A}$). The Forget function receives a schedule s_{in} which is a random member of the elite schedules and the input scheduling blocks to produce a temporary schedule (s_{temp}) with some scheduling blocks removed (line 6). s_{temp} is then passed to the Repair function to produce a new full solution as s_{out}

Algorithm 1 Forgetful Swarm Optimization

```
1  Algorithm FSO( B_in, s_base ) :
1      initialize S_elite, f_avg, f_avg^prev, and λ_avg;
2      set λ_B of every B to λ_max;
3      while λ_avg > λ_min || f_avg > f_avg^prev
       // Generation cycle.
4      do
5          foreach agent ∈ A do
6              s_temp ← agent.Forget(s_in, B_in);   // s_in
                 is a random member of S_elite.
7              s_out ← agent.Repair(s_temp, B_in);
8          end
9          success_counter ← count successful agents;
10         foreach successful agent do
11             foreach B ∈ forgotten_set_agent do
12                 λ_B ← λ_B + δ*(||A||/success_counter);
                   // Rewarding.
13             end
14         end
15         update S_elite, f_avg, f_avg^prev, and λ_avg;
16     end
17     return S_elite;
```

(line 7). Once all agents produced their s_{out}, FSO counts the number of successful agents as indicated in line 9. An agent is considered successful if the fitness of its s_{out} is higher than its input schedule, s_{in}. The cycle spanning from lines 10 to 14 is for rewarding the successful agents. It is doing so by going through every scheduling block in the *forgotten_set* of the successful agents and increasing their λ_B value proportionate to the optimization parameter of δ, and the rate of the swarm size to the *success_counter*. The maximum value of the reward matches the maximum drop of a λ_B in a generation ($\|\mathcal{A}\| * \delta$). In this way, in the earlier generations of optimization where success is more common, the reward is small, and in the later generations, where success rarely appears, a larger value is rewarded.

After the rewarding process, to finish a generation and prepare for the next one, FSO updates the necessary variables as shown in line 15. The schedules with the highest fitness between current members of $\mathcal{S}_{elite}$ and every s_{out} produced by the agents create a new $\mathcal{S}_{elite}$ to be used by the next generation of agents. f_{avg}^{prev} is copied from f_{avg} and the value of f_{avg} and λ_{avg} is updated base on the new $\mathcal{S}_{elite}$. Once the generation cycle stops and the terminal conditions are reached, FSO returns the latest $\mathcal{S}_{elite}$ as its output in line 17.

B. AGENT

The main responsibility of an agent in FSO is to build new solutions for the algorithm. To achieve this, the agent stochastically removes parts of the input schedule using the Forget function (Algorithm 2). With the results from the Forget function, the agent subsequently builds a new schedule using the constructive metaheuristic of the Repair function

(Algorithm 3), and returns it to the main FSO algorithm (see Section III-A, Algorithm 1).

Algorithm 2 Agent - Forget

```
1  Algorithm Forget(s_in, B_in):
1      initialize forgotten_set and s_temp;
2      foreach B ∈ s_temp do
3          x ← RandomNumber(0, 1);
4          if x < λ_B then
5              s_temp ← s_temp − {b}; // Removes all
                tasks of B from s_temp
6              λ_B ← LambdaDecay(λ_B, δ);
7              forgotten_set ← forgotten_set ∪ {B};
8      end
9      return s_temp;
```

According to Algorithm 2the agent requires a base schedule (s_{in}) and a set of scheduling blocks ($\mathcal{B}_{in}$) to start Forget. As previously mentioned, each agent has a *forgotten_set* which keeps track of the scheduling blocks it removes from s_{in}. This is initialized as an empty set. The other initialization of Forget is for s_{temp} set to s_{in}. s_{temp} is the schedule that is manipulated in the forgetting process. The main cycle between lines 2 to 8, iterates over every scheduling block in s_{temp}. For each scheduling block B, the agent picks a random number between 0 and 1 as x (line 3), and if this number is less than the forgetting probability of B (λ_B) it removes all the scheduled tasks of that scheduling block from the schedule as shown in line 5. The λ_B of the forgotten block is then reduced with LambdaDecay function, which by default subtracts δ from λ_B to the minimum of λ_{min}. This linear variation does not favor exploration or exploitation, whereas concentrating on the low λ values promotes exploitation, and focusing on high values contributes to wider exploration. The final step is to add the forgotten B to the agent's *forgotten_set* in line 7. Forget returns s_{temp} as output which will be used as input for the second main function of *Agent*, that is Repair.

The objective of Repair is to maximize the fitness of the schedule that has been passed to it as input using the scheduling blocks that are not in it. The agents utilize a constructive metaheuristic algorithm, which is derived from the Hybrid Accumulative Planner (HAP) [5], with some modifications to its optimization parameters to allow the algorithm to search a larger space suitable for the SI framework. The reason behind utilizing this algorithm for the repair strategy of FSO is the relatively good results obtained by HAP on AOS, and its lack of proper global optimization limiting its search capabilities on large and complex problems. HAP is specifically designed for the AOS problem, which takes into account its required flexibility and has performed well in its evaluations [5], [45].

Repair processes every scheduling block that does not exist in its input schedule (s_{temp}), sequentially in random order and adds its tasks to the best schedule found so far in the search. If the result of the additions has higher fitness than the current best schedule, it replaces the best to be used as the base for the process of the next B. A single execution of Repair performs a greedy local optimization to check and evaluate the specific constraints of different B and its tasks more directly. FSO however, operates a global optimization as the result of the random order of input processing in Repair, the Forget function of the agents, and numerous repetitions of them through generations. Algorithm 3 details the process of Repair.

Algorithm 3 Agent - Repair

```
1  Algorithm Repair(s_temp, B_in):
1      B_valid ← {B : B ∈ B_in & B ∉ s_temp};
2      foreach random B ∈ B_valid do
3          S_B ← {s_temp}; // Set of schedules.
4          foreach t ∈ B   // Scheduling block
            scheduler cycle.
5          do
6              set lobby, iter, cost_t, and s_next;
7              while iter < ι_max & lobby ≠ ∅ // Task
                scheduler cycle.
8              do
9                  s_next ← CRU(s_next, lobby); // See
                    algorithm 4.
10                 cost_next ←
                    CalculateCost(s_next, lobby);
11                 if cost_next < cost_t then
12                     s_temp ← s_next;
13                     cost_t ← cost_next;
14                 end
15                 iter ← iter + 1;
16             end
17             S_B ← S_B ∪ s_temp;
18         end
19         s_temp ←
            the schedule with the highest fitness in S_B;
20     end
21     return s_temp;
```

According to line 1 in Algorithm 3, the Repair function first identifies the scheduling blocks B that are members of $\mathcal{B}_{in}$ but are not in s_{temp}, and saves them in $\mathcal{B}_{valid}$. The main cycle between lines 2 to 20 goes through all members of $\mathcal{B}_{valid}$ in random order. The first step of the cycle is to initialize S_B, which is a set to hold the schedules produced by accumulated tasks of B, by the sole member of s_{temp}. The secondary cycle spans from lines 3 to 17 and is responsible for going through the tasks (t) of B. It is worth mentioning that while by default it goes through all the tasks, the conditions of this cycle could be adopted for handling specific scheduling block constraints (C_B). Line 5, sets the required variable for scheduling t by setting *lobby* (a set of tasks) to t, iteration counter *iter* to zero, temporary cost of adding t to the schedule to the maximum possible value ($+\infty$), and s_{next} to s_{temp}. The

s_{next} schedule holds the result of the heuristic of FSO, the Conflict Resolution Unit (CRU, see Section III-C), and the set *lobby* keeps the tasks that the algorithm must schedule at any time.

The task scheduler cycle between lines 6 to 15 schedules t into s_{next} and while there are tasks in *lobby* and *iter* has not reached its maximum value ι_{max} (optimization parameter) it tries to reduce its cost. The cost of adding a task is defined as the fitness loss of the schedule if the new addition breaks the constraints or replaces any previously scheduled tasks. The CRU function, mentioned in line 8 and detailed in Algorithm 4, takes a schedule s_{next} and *lobby* as input and returns an updated schedule, which always contains the initial task from *lobby*, but might have lost one or several other tasks that were already in the input schedule and caused conflicts with the new addition.

The cost of each iteration is calculated in line 9, and if it is lower than $cost_t$, the algorithm updates it and replaces s_{temp} with s_{next} (lines 10 to 13). In this way $cost_t$ and s_{temp} always hold the best results found in the task scheduler loop. An important note here is that once CRU schedules a task, it will not replace it in the next iterations to avoid falling into a vicious cycle or losing t. there are two possibilities to exit the loop, reaching an empty *lobby* meaning task t is scheduled without a cost, or iterations reach the maximum number and there are still tasks in *lobby*. However, in both scenarios task t is in s_{next}, and the schedule with minimum cost is saved in s_{temp} which is used as the base for the next task of B.

In line 16, the algorithm adds s_{temp} to $\mathcal{S}_\mathrm{B}$ so it would have the best schedules found by CRU on every accumulated task of B. After all the tasks are accumulated in the schedule, the algorithm finds the highest fitness between the members of $\mathcal{S}_\mathrm{B}$ to determine which one to keep for the next scheduling block (line 18). As $\mathcal{S}_\mathrm{B}$ is initially populated with s_{temp} without any tasks of the current B (as shown in line 2), it is possible that **Repair** ignores that B completely, if no combination of its tasks manages to increase the overall fitness. When every B from $\mathcal{B}_{valid}$ are processed, the last s_{temp} is returned as the algorithm output as represented in line 20.

C. CONFLICT RESOLUTION UNIT

The CRU that is used in Algorithm 3, is the core heuristic of FSO, and is how the agents traverse the solution space. It removes a task (t) from *lobby*, stochastically selects a window of t, removes the conflicting tasks, and adds t to the schedule and the conflicts to *lobby*. The stochastic selection of CRU is set to favor the windows that have lower conflict costs with the tasks that are already on the schedule. In this way, an empty spot without any conflicts has the highest selection priority. The algorithm calculates the costs of all the t windows in the schedule and assigns a normalized weight to them based on that, in a way that higher costs translate to lower weights. The stochastic selection favors higher weights which is illustrated in Algorithm 4.

According to Algorithm 4, CRU starts with taking a task from *lobby* to schedule in s as t_{entry} and continues by initializing the *conflict_set* variable to an empty set. *conflict_set* holds the conflicts (the tasks that need to be removed to schedule t_{entry}) for every window of t_{entry} along with their costs. After these steps, CRU enters a cycle from lines 3 to 7 that goes through every window of t_{entry}. In this cycle, the conflicts of each window (w) are calculated in line 4 as $\mathcal{T}_{conflict}$ (a set of tasks), line 5 calculates the cost of removing $\mathcal{T}_{conflict}$ from the schedule s, and these two values are added to *conflict_set*.

Algorithm 4 Conflict Resolution Unit

```
1  Algorithm CRU(s, lobby):
1      t_entry ← lobby.RemoveFromSet();
2      conflict_set ← ∅;
3      foreach w ∈ W_{t_entry} do
4          T_conflict ← Conflict(s, w);
5          cost_conflict ← CalculateCost(s, T_conflict);
6          conflict_set ←
           conflict_set ∪ {(T_conflict, cost_conflict)};
7      end
8      weights ← NormalizedWeight(conflict_set);
9      T_replaced ←
       StochasticSelection(conflict_set, weights);
10     remove T_replaced from s;
11     lobby ← lobby ∪ T_replaced;
12     add t_entry to s;
13     return s
```

After the conflicts and the costs of all windows are added to *conflict_set*, CRU calculates the weights of each member with the **NormalizedWeight** function. The weight (a positive real number) of a member has an inverse relation to its cost, so a higher cost means a lower weight.

With the *conflict_set* and based on the calculated weights, the **StochasticSelection** function selects a random member, favoring higher weights, and extracts its set of conflicts to save in $\mathcal{T}_{replaced}$ in line 9. The tasks in $\mathcal{T}_{replaced}$ are removed from the schedule s and added to *lobby*, as shown in lines 10 and 11. Finally, the task t_{entry} is added to s in the place of $\mathcal{T}_{replaced}$, and the updated s is returned as the CRU output.

The CRU description completes the details of the FSO algorithm. This Section along with Sections III-A and III-B illustrate how the components are utilized to schedule tasks of scheduling blocks and a set of scheduling blocks altogether.

IV. EXPERIMENTAL EVALUATION

This section illustrates the adaptation of FSO to the particular case of the Ariel mission [46] scheduling that allows algorithm flexibility and scalability testing on various scenarios and datasets. Furthermore, FSO performance is compared against the other approaches that have been developed for the Ariel problem that consists of a greedy HC (section IV-C1), an EA [4] (section IV-C2), HAP [5] (section IV-C3), an ILS (section IV-C4) with a similar D&R Strategy as FSO, but without the Swarm Intelligence

framework, and a deterministic variation of the base heuristic of FSO and HAP, which is a type of Tabu Search referred to as Accumulative Planner (AP) (section IV-C5). These five algorithms cover a wide range of complexity, optimization power, and computational cost, to put FSO performance into perspective.

A. ARIEL PROBLEM

The Atmospheric Remote-sensing Infrared Exoplanet Large-survey (Ariel) mission that is used for the evaluation is a European Space Agency (ESA) space telescope that is planned to be launched into L2 orbit in 2029. It will study what exoplanets are made of, how they formed, and how they evolve, by surveying a diverse sample of exoplanets with a low-resolution infrared spectrograph. The Ariel problem is a specific example of AOS, which aims to schedule a space telescope for time-constrained observations such as exoplanet transits, occurring when the planet passes in front of the star, over 3.5 years of operation. A transit event occurs periodically at specific time windows depending on the ephemerides of each exoplanet target. The number of windows for each target, which varies between a few to hundreds, is determined by its orbital period, ranging from ∼1 day to a few hundred days, and the telescope's accessible sky at each moment (see Section IV-B and [45]). Thus the scheduler of Ariel typically has different opportunities to plan the observations of each target within the mission lifetime. However, to ensure the quality of the data received from the telescope, each transit observation has to be repeated several times, otherwise, it does not provide a scientific value or contribute to the fitness. This information creates the definition for the *scientific task* ($t_{sci.}$) of Ariel and the B that contains it, where $t_{sci.}$ is to observe a transit and its B is the repetition of $t_{sci.}$ for a specific number of times. The specific number of repetitions for each transit is declared in the input as a set of three values $Rep_{\mathrm{B}} = \{L_1, L_2, L_3\}$ ($L_i \in \mathcal{N}, L_1 < L_2 < L_3$). Each value represents the number of repetitions to reach a level of signal-to-noise ratio (SNR), where L_3 is the highest level and L_1 is the lowest. The transits based on their importance are requested to reach one of these levels. However, if that cannot be achieved, reaching lower levels is also acceptable but with a reduced added value to the fitness.

The transits in the Ariel datasets are divided into three Tiers to represent their scientific importance. The most valuable tier is Tier 3 ($T3$), or benchmark, in which the transits require L_3 number of observations. The transits in Tier 2 ($T2$), or deep survey, have the second importance after $T3$ and require L_2 repetitions. The least important tier is Tier 1 ($T1$), or reconnaissance survey, where the transits are only requested to be repeated L_1 times. In this way, there are three ways to gain fitness by the scheduling of $T3$ transits (up to L_1, L_2, or L_3), two for $T2$ (up to L_1 or L_2), and one for $T1$ (up to L_1). The fitness gain depends on the highest reached level of SNR and the *Priority* (*pri*) value assigned to it.

The objectives in the Ariel problem are broken down into two parts. The first is to maximize the accumulated fitness of the scientific tasks in the schedule, and the second is to maximize the time that the telescope spends on them. For both objectives, there shouldn't be any overlap between the different scheduled tasks. Each $t_{sci.}$ has its specific duration that should be considered by the scheduler for this objective. The first objective is measured by the priority fitness (F_P), and the other by the time fitness (F_T). The functions to calculate these fitness values on a schedule s are described in Equations 1 and 2.

$$F_P(s) = \frac{\sum_{B \in s} pri_{\mathrm{B}}}{\sum_{B' \in \mathcal{B}} pri_{B'}} \tag{1}$$

$$F_T(s) = \frac{\sum_{t_{sci.} \in s} \|t_{sci.}\|}{\|mission\|} \tag{2}$$

In Equation 1, F_P of s is calculated on the sum of the priorities of the B in the schedule, normalized by the sum of priorities of all the input. The priority values assigned to the completion of different tiers are 1000, 100, and 10 for $T3$, $T2$, and $T1$ respectively. Equation 2 notes that F_T is obtained by dividing the accumulated duration of each scientific task ($\|t_{sci.}\|$) in s by the mission duration ($\|mission\|$), which is defined as 3.5 years. The overall fitness of a schedule ($Fitness(s)$) for Ariel is the mean value of F_P and F_T and is represented in Equation 3. The single objective format is preferred in the Ariel problem definition, and it also creates a fair comparison between the algorithms that do not effectively support multi-objective optimization, namely HC and AP.

$$Fitness(s) = \frac{F_P(s) + F_T(s)}{2} \tag{3}$$

Besides the scientific ones, there are *engineering tasks* ($t_{eng.}$) in the Ariel problem to make sure that the telescope is working as intended and monitor its performance. These tasks consist of *Short Calibrations*, *Long Calibrations*, and *Station-keeping operations*. Like any task, $t_{eng.}$ requires time on the schedule, however, contrary to $t_{sci.}$ their constraint is to be repeated with a specific cadence throughout the mission. The *Short Calibration* is defined as an observation of any known celestial body from a long list of suitable ones for 1 hour every 1.5±0.5 days, and *Long Calibration* is the same type of observation but for 6 hours every 30±1 days. As for the *Station-keeping* operations, which are regular maintenance tasks, frequency is set to 4 hours every 28±3 days. The main difference between $t_{sci.}$ and $t_{eng.}$ is that the desired number of the latter is not known prior to the scheduling, and as the fitness is defined on the $t_{sci.}$, the scheduler seeks to satisfy the constraint of $t_{eng.}$ but also to reduce the number of them. To handle this specific constraint, the CRU (see Section III-C) of FSO is set to calculate the next window of a $t_{eng.}$ when it schedules one and repeats until it reaches the end of the mission time. Furthermore, to reduce the number of $t_{eng.}$, the stochastic selection of CRU favors distancing consecutive $t_{eng.}$.

The exoplanet transits are associated with a host star. Therefore, they occur at different locations in the sky. To perform a new observation, the Ariel telescope must first move to the sky coordinates of the star. The time the telescope takes to point from the star of the previous task to that in the next task and stabilize the pointing is referred to as the *Slew Time*, and it depends on the angle between the coordinates of successive tasks, and the rotation speed. The scheduler of Ariel takes into account the slew time between the tasks when it checks for conflicts in CRU.

B. DATASETS

There are four available datasets for the Ariel mission scheduling. Each dataset creates a scenario with specific characteristics to simulate the different demands of the scheduler. The first and main dataset is the Mission Reference Sample (MRS) which consists of information on 999 exoplanets to observe. MRS includes the most interesting set of exoplanets that define the main scientific goal of Ariel in surveying around a thousand targets, however, even scheduling all its tasks only covers around 70% of the mission time. This time margin is allowed to leave space in the schedule for engineering tasks ($t_{eng.}$) consisting of short and long calibrations, and station-keeping operations, as well as to take into account that, inevitably, some idle time will remain between the time-constrained scientific tasks. The second dataset, referred to as MRS_B, extends *MRS* by adding 1092 T1 exoplanets to its set, to push the limits of the time dedicated to scientific observations. These additions increase the coverage time from around 70% to 200%, meaning that at least some tasks will not be in the final solution. Another way to increase the time on science is proposed in the third dataset called MRS^{PC} which adds information on 43 exoplanets to *MRS* and requests a special $t_{sci.}$ denoted as Phase Curve (*PC*) on them. What differentiates *PC* tasks from normal transits is their significantly longer duration which disrupts some $t_{eng.}$, specifically short calibrations that have a shorter cadence than the average duration of a *PC*, and specific handling should be adapted. *PC* observations were not initially defined for the Ariel problem, and they challenged the flexibility of the algorithms to adapt to new demands. The fourth and final dataset referred to as MRS_B^{PC}, gathers all aforementioned data from *MRS*, MRS_B, and MRS^{PC} in one set. The characteristics of these datasets are summarized in Table 1.

The available datasets create different scenarios with diverse dimensions and complexities to test the algorithms in the evaluation, in both the flexibility and scalability challenges of AOS.

C. ALGORITHMS SETUP

Since the definition of the Ariel scheduling problem, five approaches have been developed and applied to it. These approaches include a local optimizer with HC and four with different global optimization capabilities. The simplest approach with some global optimization capabilities is the deterministic AP based on a Tabu Search, followed by its expansion, HAP [5] with its multi-start stochastic strategy. The more capable global optimizers in the evaluation are ILS, and EA [4]. These methods cover a wide range of optimization power. Additionally, AP, HAP, and ILS with largely similar base heuristics as the proposal, put the effectiveness of their methodologies versus FSO into perspective.

TABLE 1. Datasets characterization. Considering the following features: The number of proposals containing $t_{sci.}$, a count of the proposals of *PC* and different tiers from *T3* to *T1*, the coverage time of the proposals, the number of windows, the number of $t_{sci.}$, and finally, the average number of overlapping tasks (conflicts) at any time.

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

The FSO parameters in the evaluation find a balance between computational cost and an effective increase in fitness. The values are set based on a series of empirical tests on each parameter to determine the optimal point at which, whether the rate of fitness gain with respect to the computational cost is 1 (derivative equal to 1, or the slope of 45 degrees), or there is a clear maximum. The AOS problem often comes with a time limitation, which emphasizes the lower computational cost besides the schedule's fitness. The values of these parameters are presented in Table 2.

TABLE 2. Algorithm parameters for the FSO method.

Parameter	Notation	Value
forget probability	λ_{max}	0.5
decay rate	δ	0.05
Number of agents	$\|\mathcal{A}\|$	16
Number of elites	$\|S_{elite}\|$	4
CRU maximum iterations	ι_{max}	20
Minimum λ (stop criterion)	λ_{min}	0.01

For more insight into the effects of different optimization parameter values on the FSO performance, numerous setups with 10 repetitions each have been tested. The tests on $\|\mathcal{A}\|$, $\|S_{elite}\|$, λ_{max}, and δ are evaluated on the average number of generations and Fitness. These parameters do not affect a single generation time directly, so to measure their effect on computational cost the number of generations is considered. On the other hand, the direct effect of ι_{max} in CRU is on the time it takes to execute a single generation (generation time), thus it is evaluated on this time for computational cost effect. The summary of these tests is illustrated in Figures 2 and 3. The red points in the figures indicate the default values.

According to Figure 2, increasing ι_{max} (*iter*) leads to higher fitness, however, its effectiveness diminishes in higher values. That means while low values (e.g. $\iota_{max} = 5$) do not fully use

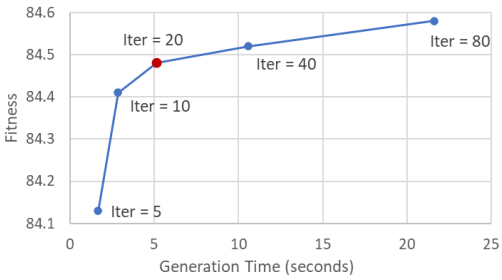

FIGURE 2. Effect of ι_{max} (iter) on fitness and generation time in seconds.

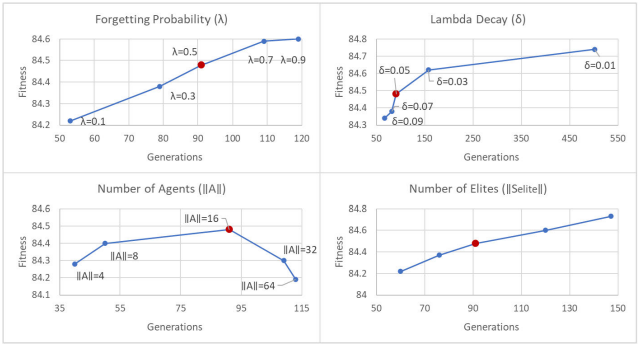

FIGURE 3. Effect of $\|\mathcal{A}\|$ (upper left), $\|S_{elite}\|$ (upper right), λ_{max} (bottom left), and δ (bottom right) on fitness and the number of generations.

the CRU search capabilities, high numbers (e.g. $\iota_{max} > 80$) exhausts CRU and does not lead to much better results. On the other hand, the effect on generation time is steady throughout all tested values.

The plots in Figure 3 show a consistent increase in fitness with more generations, except for $\|\mathcal{A}\|$. The fitness increase for the number of generations in $\|S_{elite}\|$ is relatively linear, but for λ_{max} and δ, it diminishes toward their extremum values, where λ_{max} is closer to 1 and δ reaching 0. The clearest example is on δ, where the slight fitness gain between $\delta = 0.03$ and 0.01 requires more than three times generation.

The $\|\mathcal{A}\|$ plot shows a clear optimal value for this parameter because its effect on the generations and fitness is reliant on the other parameters. δ and to some extent λ_{max} determine the effective $\|\mathcal{A}\|$ value. This is due to the dependency of the rewarding system of FSO on δ and $\|\mathcal{A}\|$. By selecting a high value for $\|\mathcal{A}\|$, the algorithm converges prematurely leading to lower fitness. In conclusion, to select suitable values for the optimization parameters, it is important to look into the limitations and restrictions imposed on specific projects.

1) HILL-CLIMBING

The Greedy algorithms' main benefits are their low computational cost and due to their simplicity, the relative ease of adaptation to a problem and implementation. The benefits come at the cost of weak optimization by its strictly local search. The adapted version of the algorithm to the Ariel problem uses a combination of four decision variables ($\{d_1, d_2, d_3, d_4\}$) to schedule the next task at any moment in the timeline. The variable τ in the HC scheduler always holds the latest time that the scheduled tasks end. Starting with an empty schedule and τ set to the beginning of the timeline, HC schedules the tasks sequentially based on its decision variables and moves τ forward on each step accordingly. The process ends when no more tasks fit into the schedule between τ and the end of the mission timeline. The decision variables of HC are calculated on every viable B based on the current schedule and the value of τ, and are defined in Equation 4:

$$
\begin{aligned}
d_1(B) &= \begin{cases} 1 & \text{if } \mathrm{B} \text{ has tasks scheduled} \\ 0 & \text{otherwise} \end{cases} \\
d_2(B) &= \frac{\| \textit{ remaining tasks of } B \|}{\| \textit{ remaining windows of } B \|} \\
d_3(B) &= \frac{pri_i}{pri_{max}} \\
d_4(B) &= 1 - \frac{dist_{\mathrm{B}}^{\tau}}{dist_{max}}
\end{aligned} \tag{4}
$$

The first decision variable, d_1, promotes completing the B that already exists in the schedule, whereas d_2 evaluates the difficulty of the B scheduling by calculating the ratio between the number of unscheduled tasks in B and the number of windows from its tasks that are after the current τ value. The decision variable of d_3 holds a normalized priority value to emphasize scheduling more important B, and finally, d_4 establishes a measure to minimize the delay between two consecutive tasks. $dist_b^{\tau}$ shows the difference in time between τ and the start of the closest window of a task in B. This value is divided by $dist_{max}$ which is the maximum delay between two tasks in the schedule and is defined in the algorithm configuration. The combination of the four decision variables determines which B gets its task scheduled after τ.

2) EVOLUTIONARY ALGORITHMS

The approach on Ariel based on EA divides the problem into two parts [4]. First, is the scheduling of the engineering tasks which is done using a Genetic Algorithm (GA), with a specific variation known as Non-dominated Sorting Genetic Algorithm 2 (NSGA-II). Second is the scheduling of the scientific tasks on top of the results from the previous part, using a Multi-Objective Evolutionary Algorithm (MOEA), which also uses non-dominated sorting. The reason behind this division is the inherent difference between the two types of tasks and their objectives in scheduling. This demands specific behavior of the scheduler depending on the type. The incorporated GA for engineering tasks starts on an empty schedule and assumes a window for each repetition based on the requested cadence and searches for places to put the tasks within those windows where it has the minimum conflict with the potential scientific tasks that could occupy the same time period. The approach utilizes a Pittsburgh-style GA [35] where an individual (*ind*) consists of $\|HC\| + \|LC\| + \|SK\|$ genes of real values that each indicates the shift from the

center of its window. The fitness function to minimize an individual's competence is defined in Equation 5.

$$F_{\text{eng.}}(\text{ind}) = \sum_{g \in \text{ind}} \sum_{B \in \mathcal{B}} \frac{\|B\| \cdot \text{pri}_{\text{B}}}{\|\text{remaining w of } B\|} \text{if conflict}(g, B) = \text{true} \quad (5)$$

Equation 5 describes the calculation of engineering fitness ($F_{eng.}$) for an individual (ind), where if there is a conflict between a gene of ind and a B from the list of all available ones ($\mathcal{B}$), $F_{eng.}$ increase. The increase is proportional to the priority of the conflicting B, and a measure to estimate the difficulty of its completion. This is represented as the number of required tasks in B ($\|\text{B}\|$) divided by the number of remaining windows after the conflict point for B.

After completing the scheduling of $tasks_{eng.}$, the best solutions are passed to the MOEA to add the $tasks_{sci.}$. Each individual of MOEA consists of $\sum_{\text{B} \in \mathcal{B}} \|\text{B}\|$ genes, and each gene can take a value between $[-1, \|\mathcal{W}_{\text{B}}\| - 1]$. The value of -1 for a gene is interpreted as unscheduled, and the rest of the values in the range indicate the window where B has a task scheduled. The objectives for this part are as defined in the problem definition. However, the EA implementation works on minimization and its fitness functions are defined to present the complement of the normalized objectives. Also during its cycles, MOEA treats the objectives with a Pareto Optimal Front to further diversify the solutions. 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. The parameter values are set based on the results presented in [4].

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	$1/\|individual\|$

3) HYBRID ACCUMULATIVE PLANNER

HAP is a Multi-Start metaheuristic algorithm designed on the AOS problem [5], and it is used as the base of FSO development. HAP focuses on local optimization, whereas FSO concentrates on global optimization. The local position of HAP is its sensitivity to the input order as it processes every scheduling block once and decides on whether to keep them in the schedule or not based on a local greedy benefit to the fitness. FSO on the other hand, with numerous attempts to schedule the scheduling blocks in random orders, through Forget and Repair, breaks the sensitivity of HAP and explores much wider neighborhoods in the solution space. HAP searches fewer areas in the solution space than FSO but to make up for it, spend more time creating better individual solutions. The heuristic step of both algorithms share many similarities, however, HAP uses a much higher ι_{max} and in its *stochastic replacement*, uses a configuration value of Stochastic Range (η) which limits the replacement to a few of the best options rather than all of them as in FSO. HAP runs several instances of its metaheuristic simultaneously (instances) to create different solutions and makes an elitist selection at the end to return the best ones. HAP uses a similar strategy as FSO to schedule $tasks_{eng.}$ in the same routine as $tasks_{sci.}$ while distancing them to reduce their number. The algorithm parameters in the test are described in Table 4. These values are obtained based on the tuned results presented in [5].

TABLE 4. Algorithm parameters for the HAP method.

Parameter	notation	Value
instances	k	16
CRU maximum iterations	ι_{max}	200
Stochastic range	η	3

In the evaluation, HAP is set with $\iota_{max} = 200$, $\eta = 3$, and 16 simultaneous instances. In comparison, FSO also runs on 16 agents, though with $\iota_{max} = 20$ and tens of repetitions (generations). As a result, HAP has increased exploitation and low exploration. These characteristics make it most suitable for small-sized problems with low complexity where large exploration has a limited impact on the results.

4) ITERATED LOCAL SEARCH

The ILS used in the evaluation, is derived from a similar heuristic and D&R of FSO. The main goal of including this algorithm is to gain insight into the Swarm Intelligence (SI) framework of FSO, and how it compares with iterations without agent interaction. As a result, all the scheduling blocks are assigned with a constant forgetting probability (λ) which does not increase nor decrease over the iterations. The stopping criteria consist of two conditions: reaching a minimum number of iterations and stagnation on the fitness improvement for certain consecutive iterations. Once both these conditions are reached, the algorithm stops and returns the elite schedules of the last iteration as final solutions. The values of the optimization parameters of the ILS in the evaluation are selected as closest to FSO, to create a better comparison between the two. Table 5 shows these values.

TABLE 5. Algorithm parameters for the ILS method.

Parameter	Notation	Value
forget probability	λ	0.5
Number of agents	$\|\mathcal{A}\|$	16
Number of elites	$\|S_{elite}\|$	4
CRU maximum iterations	ι_{max}	20
Maximum stagnation	$stag_{max}$	10
Minimum ILS iterations	$iteration_{min}$	The average generation of FSO on each dataset.

According to Table 5, the first four parameters and their values are the same as FSO, however, the last two rows for

Maximum stagnation ($stag_{max}$) and *Minimum ILS iterations* ($iteration_{min}$) which are the stop criteria of ILS are set to improve its competitiveness. The value of $iteration_{min}$ is set dynamically on the average generations that FSO takes to reach its solutions, to avoid early convergence of the algorithm. The value for $stag_{max}$ is set to 10 meaning that the condition is reached if ILS does not see an improvement in fitness in ten consecutive iterations. The performance of this ILS singles out the effects of guided forgetting and the reward strategy of FSO through manipulation of λ.

5) ACCUMULATIVE PLANNER

The final method included in the evaluation is based on a Tabu Search algorithm that is modified to be used in HAP and FSO. This method referred to as the AP, as opposed to HAP and FSO is a single solution and deterministic approach. The deterministic strategy of AP comes from its simplified CRU (see Section III-C) which instead of stochastic replacement, always selects the option with the lowest cost to replace a task from the schedule with a member of the *lobby*. The search in this way reduces the overall cost generally, however, if there is no replacement option available to achieve that, it continues with the next best option. The optimization parameter of AP consists of a single parameter which is the CRU maximum iterations (ι_{max}). Similar to HAP, AP is also sensitive to the input scheduling block order and the blocks that are processed earlier have a slight advantage in being in the final solution. Table 6 shows the value of ι_{max} that is used in the tests, which is similar to HAP and derived from [5].

TABLE 6. Algorithm parameter for the AP method.

Parameter	Notation	Value
CRU maximum iterations	ι_{max}	200

D. RESULTS

In this section, test results with the Ariel problem datasets on FSO, ILS, HAP, EA, AP, and HC are analyzed. Every test is repeated 10 times on non-deterministic methods (FSO, ILS, HAP, and EA). The values in all the tables show the median results, except for the computational cost table in which the mean result is reported. The tests on every dataset are detailed in the next sections.

1) *MRS* DATASET

The *MRS* dataset contains the tasks that define the main goal of Ariel in repeated observation of about 1000 exoplanets' transit. Thus, it has the highest importance in the scientific aspect. Scheduling all the tasks in *MRS* requires less time than is available, though it is not guaranteed that all of them could be scheduled due to time constraints, slews, and $tasks_{eng.}$. Table 7 summarizes the results of the different algorithms used to schedule the *MRS* tasks, where the second to the fourth rows show the number of completed B from the different tiers, and the fifth row displays the amount of time dedicated by the algorithms to the $tasks_{eng.}$. The last three rows indicate the values of time fitness (F_T), priority fitness (F_P), and overall fitness (F) in the percentage format.

According to Table 7, FSO achieves better results than the rest of the algorithms, as indicated in the fitness rows. This is a result of consistently completing all the 999 scheduling blocks provided by the dataset. The second-best performance is shared between ILS and HAP, which also in some repetitions of the test schedules all the tasks, however in other cases it comes short in one or two Tier 1 B. Tier 1 has the lowest priority value, thus missing a few of them does not reduce F_P significantly. The results from AP closely trail the top three which shows its high competence in solving the problem.

HC algorithm ranks fourth in the evaluation with $F = 83.2$, and *EA* scores the lowest fitness with $F = 81.95 \pm 0.2$. *MRS* dataset is relatively simple yet has an extremely large number of possible solutions. As a result, a fully global optimization, like EA, without many assumptions about the solution space is not efficient in narrowing down its search to more competent areas. On the other hand, this type of problem suits a greedy algorithm like HC, which has a strictly narrow and deterministic search. All the methods based on variations of CRU base heuristic achieve notably better results than the other two.

Observation 1: On the less complex *MRS* dataset, FSO performs similarly to its closest algorithms (ILS and HAP), and its global search does not negatively affect its competence as in the EA, for this relatively small scenario.

2) MRS_B DATASET

The MRS_B dataset represents a more complex scenario than *MRS*, where there is a far larger number of B in the dataset than there is time available (over-subscribed). This affects the local greedy algorithm like HC negatively by a significant degree, and at the same time, improves the problem conditions for global optimizations. Table 8 shows the performance of the algorithms on the MRS_B dataset.

The best performance on MRS_B, as indicated in Table 8, is from FSO with a noticeable difference. The gap in fitness with ILS and HAP is more significant than in *MRS*, specifically in F_T, as the result of including about many more Tier 1 that does not add to F_P much (due to their low priority) but occupy more time on the schedule. The FSO search also manages to spend the least amount of time on $tasks_{eng.}$ which leaves more availability for $tasks_{sci.}$. The difference between ILS and HAP on MRS_B is larger than on *MRS* and while ILS schedules fewer scheduling blocks than HAP, it selects longer blocks that contribute to higher F_T. AP achieves a lower overall fitness in its solution but remains competitive to larger global optimizations based on CRU heuristics. Unlike *MRS*, in MRS_B, EA performs much better than HC, but the number of Tier 2 B that it completes is below FSO, ILS, and HAP. There are two main reasons for this performance. First, is the method that uses the algorithm to schedule the problem. Separating the scheduling of $tasks_{sci.}$ and $tasks_{eng.}$, besides

TABLE 7. Algorithms performance on the *MRS* dataset.

Method	FSO	ILS	HAP	AP	EA	HC
Tiers 3	50	50	50	50	50	50
Tiers 2	550	550	550	549	532 ± 3	534
Tiers 1	399	398 ± 1	398 ± 1	396	390 ± 4	408
eng. time (%)	3.61 ± 0.05	3.63 ± 0.05	3.51 ± 0.33	3.64	4.1	5.25
F_T (%)	**69.97**	69.78 ± 0.19	69.66 ± 0.31	69.19	65.9 ± 0.42	67.79
F_P (%)	**100**	99.99 ± 0.01	99.99 ± 0.01	99.88	98.21 ± 0.11	98.61
F (%)	**84.98**	84.90 ± 0.08	84.89 ± 0.09	84.53	81.95 ± 0.18	83.2

TABLE 8. Algorithms performance on the MRS_B dataset.

Method	FSO	ILS	HAP	AP	EA	HC
Tiers 3	50	50	50	50	50	47
Tiers 2	550	549 ± 1	549.5 ± 0.5	548	399 ± 7	264
Tiers 1	545 ± 10	468 ± 7.5	491 ± 4	480	660 ± 11	661
eng. time (%)	3.24 ± 0.03	3.68 ± 0.05	3.6 ± 0.11	3.67	4.1	4.47
F_T (%)	$\mathbf{76.87 \pm 0.38}$	74.09 ± 0.24	73.64 ± 0.23	72.93	68.9 ± 0.20	67.65
F_P (%)	$\mathbf{92.11 \pm 0.08}$	91.41 ± 0.10	91.66 ± 0.03	91.14	80.49 ± 0.62	66.72
F (%)	$\mathbf{84.49 \pm 0.23}$	82.79 ± 0.12	82.65 ± 0.15	82.03	74.72 ± 0.38	67.19

TABLE 9. Algorithms performance on MRS^{PC} dataset.

Method	FSO	ILS	HAP	AP	HC
PC	43	43	42 ± 1	41	43
Tiers 3	50	50	50	50	50
Tiers 2	549 ± 1	547 ± 2	527 ± 2	528	504
Tiers 1	330 ± 6	262 ± 5	319 ± 3	315	364
eng. time (%)	3.28 ± 0.09	3.17 ± 0.09	3.43 ± 0.03	3.45	4.61
F_T (%)	$\mathbf{75.82 \pm 0.22}$	73.23 ± 0.22	73.60 ± 0.19	72.94	75.02
F_P (%)	$\mathbf{99.5 \pm 0.07}$	98.94 ± 0.09	97.71 ± 0.10	97.34	96.76
F (%)	$\mathbf{87.68 \pm 0.12}$	86.10 ± 0.14	85.77 ± 0.14	85.14	85.69

TABLE 10. Algorithms performance on MRS_B^{PC} dataset.

Method	FSO	ILS	HAP	AP	HC
PC	43	43	42 ± 1	42	40
Tiers 3	50	50	50	50	48
Tiers 2	549 ± 1	547 ± 2	515 ± 3	516	231
Tiers 1	355 ± 5	292 ± 7	374 ± 4	603	374
eng. time (%)	3.35 ± 0.05	3.26 ± 0.08	3.47 ± 0.05	3.47	4.48
F_T (%)	$\mathbf{77.68 \pm 0.21}$	74.76 ± 0.27	74.74 ± 0.15	74.20	68.93
F_P (%)	$\mathbf{93.03 \pm 0.05}$	92.49 ± 0.11	90.43 ± 0.14	90.12	77.06
F (%)	$\mathbf{85.35 \pm 0.12}$	83.66 ± 0.16	82.58 ± 0.11	82.16	72.99

not allowing them to undergo a homogeneous process, takes some opportunities for optimization away from the final solution. The second reason is the early convergence of the EA set by the time limits assigned to the execution of the scheduler. The flexible structure of FSO allows coherent optimization across different tasks of the problem, and this is more significant as the problem scenario grows larger.

Observation 2: The global optimization framework of FSO and its SI interactions of the agents greatly increases the performance, compared to other CRU-based methods, namely ILS, HAP, and AP, in an over-subscribed dataset such as MRS_B. Although purely global optimization algorithms like EA are significantly better than the greedy local search algorithm of HC, the difference between EA and the hybrid approaches of FSO and HAP is much larger.

3) MRS^{PC} AND MRS_B^{PC} DATASETS

The last two datasets of Ariel create scenarios where a new type of task, PC, with specific constraints and conflicts, is added to the problem. MRS^{PC} and MRS_B^{PC} represent these scenarios on small and large datasets respectively. The EA could not be adapted to PC without going under design change, and extensive work, so it was omitted from the test on these datasets. The most undemanding adaptation to PC is for HC due to its simple architecture and strict local search which allows schedule manipulation on a precise level. The structure of handling tasks and scheduling blocks shared between AP, HAP, ILS, and FSO also gives the algorithms an entry point to add specific rules for some tasks. By enforcing the scheduling of the Bs containing $tasks_{eng.}$ before PCs, which have a high priority value, these algorithms make sure the cadence

of $tasks_{eng.}$ are as requested except the situations where a task of PC occupies its windows. The results of the tests on MRS^{PC} and MRS_B^{PC} datasets are detailed in Tables 9 and 10, respectively.

Tables 9 and 10 indicate that, like on the other datasets, FSO has the best performance between the algorithms in the evaluation on MRS^{PC} and MRS_B^{PC}. Unlike the results from MRS (Section IV-D1) where FSO, HAP, and ILS have almost similar performances, MRS^{PC} results show a larger gap in fitness between them. The deterministic approach of AP struggles to keep up with the other methods and it has the lowest performance below HC.

As for MRS_B^{PC}, FSO covers about 3% more time with scientific tasks, reflected in F_T, than ILS, HAP, and AP. Combined with higher priority fitness (F_P), it has a clear advantage over the other methods. Unlike in MRS^{PC}, on the larger and more complicated dataset, AP makes a good solution, close to HAP, which puts it well above HC. Similar to MRS_B, HC does not handle large datasets well.

Observation 3: FSO consistently obtains better results than the rest of the algorithm on all datasets including MRS^{PC} and MRS_B^{PC}, demonstrating the algorithm's adaptation to new conditions. The core heuristic and solution building of FSO which is mostly shared with ILS, HAP, and AP provides the necessary flexibility to these algorithms to handle the challenges of AOS.

4) COMPUTATIONAL COST

The computational cost of FSO can be viewed from two perspectives. One view is to assess the worst-case time complexity (Order), and the other is to compare the execution times of the results detailed in Sections IV-D1, IV-D2, and IV-D3. The time complexity of FSO is estimated for a single generation, as described below:

$$O(FSO) = \|\mathcal{A}\| * O(\|\mathcal{T}\|) * (1 + iter_{max} * O(\overline{\|W_t\|})), \quad (6)$$

where $\|\mathcal{A}\|$ displays the number of agents, $\|\mathcal{T}\|$ indicates the total number of tasks in all the scheduling blocks of $\mathcal{B}$, and $\overline{\|W_t\|}$ shows the average number of windows for a task in $\mathcal{T}$.

According to Equation 6, beside the configuration parameters of $\|\mathcal{A}\|$ and $iter_{max}$, the time complexity of a single generation of FSO is bound to $O(\|\mathcal{T}\|)$ and $O(\overline{\|W_t\|})$. While the value of $\overline{\|W_t\|}$ does not necessarily change by adding or removing scheduling blocks, the number of tasks has a direct relation with the size of the input and it impacts linearly on the computational cost.

The worst-case scenario of FSO, assumes that CRU always reaches its maximum iteration ($iter_{max}$), which usually does not happen when the schedule is rather empty. As a result, the time complexity assessment in Equation 6 applies to the process on a full schedule. The number of generations, which was omitted from the time complexity assessment, depends on several aspects of the inputs such as the data complexity and constraints, however, it can be controlled by adjusting the configuration parameters of λ_{max} and δ.

Assessing the execution time of the scheduling methods on a specific problem depends on factors besides their algorithm, such as implementation details, and configuration of optimization parameters. Despite that, the computational cost of the methods in evaluation are compared, to illustrate a perspective on the execution time of FSO. Table 11 shows the average execution time, in seconds, of the tests performed for Sections IV-D1, IV-D2, and IV-D3.

TABLE 11. Comparison of the computational cost (in seconds) of the methods.

Method	MRS	MRS_B	MRS^{PC}	MRS_B^{PC}
FSO	4.80	431.55	45.37	507.01
ILS	7.97	536.25	83.92	816.98
HAP	0.70	32.8	2.9	35.98
AP	0.34	20.47	2.20	22.33
HC	5.06	7.87	4.82	8.21
EA	1012	2118	-	-

Based on the values displayed in Table 11, the lowest average time of the scheduling belongs to HC, despite not being the fastest on MRS and MRS^{PC}. HC computational cost is the least affected by the size of the dataset, as it just increases the number of options to choose from in each step of the algorithm and does not affect its overall cycles. AP has the lowest time on MRS and MRS^{PC}, and it is the second fastest method on average. Despite having 16 instances in HAP, compared to the single-solution AP, its computational cost does not increase more than double, due to parallelization. The EA method in evaluation, due to lack of a local guide takes much longer time to reach higher fitness.

FSO and ILS as more global optimization algorithms have access to similar parts of solution space. The average number of generations in FSO for MRS, MRS^{PC}, MRS_B, and MRS_B^{PC} are 31, 44, 87, and 101 respectively. These values are set for $iteration_{min}$ optimization parameter of ILS (see Section IV-C4). This is to avoid early convergence of ILS, but usually, it takes more iterations to converge. It is apparent in the time comparison on each dataset, where ILS on average takes about 59% more time than FSO. The EA method in evaluation, due to the large solution space and lack of a local guide takes a much longer time to reach the potent neighborhoods in the solution space and converge.

Observation 4: The combination of local and global optimization in FSO allows the algorithm to have a competitive computational cost with smaller algorithms like HC and HAP on the simpler scenarios of the problem, and be faster than ILS and EA on more complex ones.

V. STATISTICAL ANALYSIS

To evaluate the significance of the differences between the stochastic methods, FSO, ILS, HAP, and EA based on the repeated tests on the available datasets, we used the Friedman-Nemenyi [47] test. The Friedman test is commonly used to identify if there are significant differences between multiple groups of related data, and the Nemenyi post hoc test follows it up to pin down which pairs differ significantly. The

TABLE 12. Friedman-Nemenyi paired p-values on different datasets.

Pair	MRS	MRS_B	MRS^{PC}	MRS_B^{PC}
(FSO, ILS)	0.091	0.065	0.077	0.077
(FSO, HAP)	0.065	**0.023**	**0.023**	**0.023**
(FSO, EA)	**0.016**	**0.007**	-	-
(ILS, HAP)	0.376	0.156	0.077	0.077
(ILS, EA)	0.065	**0.026**	-	-
(HAP, EA)	0.091	0.076	-	-

outcome of the test, or p-value, is compared with a confidence threshold to determine the significance. Table 12 shows the p-value of the Friedman-Nemenyi test between algorithm pairs.

According to Table 12 and considering the 90% confidence level (p-value < 0.10), the results from FSO on all datasets outperform the other methods by a statistically significant margin. The ILS approach has the closest results to FSO and the significance does not reach the 95% (p-value < 0.05) threshold. However, considering that ILS on average takes about 59% more time to schedule (see Table 11), FSO has a clear advantage. The paired comparison with HAP indicates a more significant difference from FSO with 95% confidence on all datasets except *MRS*. On *MRS*, both algorithms can reach maximum fitness, limiting the comparison capability. The improved results of FSO compared to EA on the *MRS* and MRS_B datasets are significant with 95% confidence.

To provide more context to the statistical analysis, Table 12 also includes the p-values of the pairs ILS-HAP, ILS-EA, and HAP-EA. These results show no significant difference between ILS and HAP on *MRS* and MRS_B. However, on MRS^{PC} and MRS_B^{PC}, ILS significantly outperforms HAP with 90% confidence. The comparisons between ILS and HAP with EA also show significant differences, particularly between ILS and EA on MRS_B.

Finally, another Friedman-Nemenyi test, not shown in Table 12, has been performed on the average results from each dataset across all evaluated methods, including AP and HC. It shows statistically significant differences between FSO and AP (p-value = 0.022) and between FSO and HC (p-value = 0.014) with 95% confidence.

VI. DISCUSSION

The experimental evaluation of FSO shows the persistent competence of the algorithm in handling problems of different complexities and adaptation to new constraints and conditions. In objective completeness, it consistently outperforms all the other algorithms including the other methods based on similar heuristics, AP, HAP, and ILS. On average, FSO obtains higher total fitness than ILS, HAP, AP, EA, and HC, by 1.3%, 1.7%, 2.2%, 4.8%, and 8.4%, respectively. Figure 4 breaks down this advantage on priority fitness (F_P) and time fitness (F_T), and compares relative computational costs.

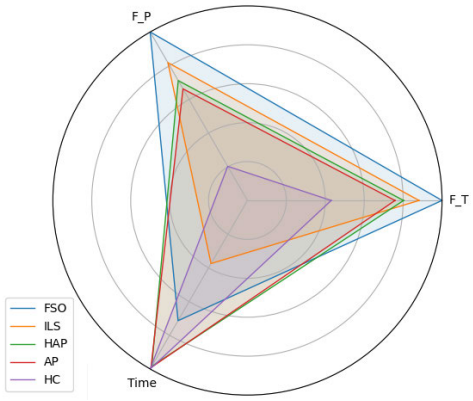

FIGURE 4. Algorithms average performance on F_T, F_P, and the computational cost. To better distinguish the differences, relative normalized values are used in the figure. Higher points in every measure, F_T, F_P, and *Time*, represent better performance.

Based on the evaluation results, the gradual improvement from AP to HAP, HAP to ILS, and ILS to FSO, indicates a growth in optimization power with efficient search in larger accessible solution space. The multi-start expansion and limited stochastic selection (by optimization parameter of ρ (See Section IV-C3)) of HAP are effective in improving the single-solution determinism of AP. Repetition of a modified HAP (without limitation on ρ) through a *D&R* strategy also leads to better results in ILS. However, FSO with its added SI framework not only improves the ILS results in fitness, but also in a much shorter time. The directed exploration of FSO with forgetting probability decay strategy, in combination with rewarding the success to encourage exploitation of potent areas in the solution space, proves the effectiveness of FSO in using SI methodology.

The flexibility of the proposed constructive metaheuristics combined with the design of constraints handling in FSO, provides the algorithm with a vast range of possibilities to define special tasks and scheduling blocks, as demonstrated in scheduling the $tasks_{eng.}$ and Phase Curves. CRU manages C_t by its Tabu Search, Repair maintains C_B, and the objectives are optimized by the swarm of agents in FSO.

Due to the internal cycles of Repair, building solutions in FSO is computationally costlier than the similar process in global optimization approaches like EA (with the crossover and mutation), and in return, it builds better individual solutions and reduces the number of generations. This results in overall less computational cost than a typical global optimization algorithm. The good results of the proposal in the evaluation are achieved by the combination of building high-quality starting points and a robust interaction strategy of the swarm.

FSO consists of novel features to create a population-based optimization from a *D&R* strategy that distributes the search evenly across the tasks and exploits interesting areas found by its agents to improve the solutions. Although it is designed for the AOS problem, the algorithm is adaptable

to other Combinatorial Optimization problems with similar characteristics that aim for performance and flexibility. The internal components of the proposal, Repair, and CRU, contain most of the problem-specific adjustments that fit FSO into AOS, and the swarm behavior has a generic definition. Thus, in order to adapt FSO to other CO problems, the focus should be on the internal components.

In addition, Repair and CRU can be replaced with algorithms that do the same job. CRU follows a Tabu Search methodology, as it takes steps to worsen the immediate solutions if no improvement is found. Tabu Search is a fast and well-established algorithm that suited AOS characteristics the most, however, other algorithms based on different methodologies such as Simulated Annealing (SA) can replace CRU to fit a problem. As a constructive metaheuristic, Repair can also be swapped with alternative algorithms, and as long as the agents of FSO can perform their Forget and Repair process, the swarm interaction manages to direct the search into competent neighborhoods of the solution space.

VII. CONCLUSION

The Astronomical Observation Scheduling (AOS) problem relies more on computer science solutions, as its complexity grows with new technologies and connectivities. The specific challenges of AOS, being scalability and flexibility, add extra criteria to evaluate an algorithm used to solve it. The FSO algorithm presented in this paper is designed to handle this problem and its specifics. FSO's solution to the *scalability challenge* of the AOS problem is to rely on its *global optimization* of the high-level components and *local optimization* of its agents' metaheuristics. This combination reduces the computational cost of the algorithm compared to fully global optimizations to reach competent results, especially on problems with less complexity.

The answer to the *flexibility challenge* is the different components of FSO, which have the responsibility to maintain solutions on different levels of the problem constraints and facilitate the introduction of new ones. The core heuristic component of FSO (CRU), schedules the individual tasks and handles their constraints. This feature is used in scheduling the engineering tasks of the Ariel problem to create a cadence between them. The metaheuristic component of FSO (Repair), has the responsibility to schedule dependant tasks or *scheduling blocks* (B), taking into account their constraints. Repetition of the scientific tasks in the Ariel problem is an example of B constraints handled by Repair. Finally, the objectives or the high-level constraints are optimized globally by the agents of FSO.

The proposed algorithm outperforms the other implementations of the Ariel scheduler on *objective completeness*, consistently throughout different scenarios. The *computational cost* of it, though heavier than local optimizations is meaningfully less than the fully global optimizers such as EA. In addition, the in-depth analysis of the FSO configuration demonstrates how to balance the objectives' completeness and the computational cost, and further tune the algorithm for the different requirements of the problem.

There are several directions to continue this study. As discussed in Section VI, exploring alternative algorithms to Repair and CRU of FSO is deemed valuable. The bulk of the operations in FSO takes place in the CRU component, thus lowering the computational cost of the CRU while maintaining its competence is a promising path to improve FSO. Furthermore, the configuration input of the algorithm can be embedded as a part of the scheduler and optimization process to tune the algorithm to the desired settings automatically. Finally, the applications of FSO on other variations of the Combinatorial Optimization problems could be evaluated. Comparison between a generalized version of the proposal and the state-of-the-art solutions for well-known examples of Combinatorial Optimization would provide more insight into its performance.

TABLE 13. *FSO* terminology and denotation.

Denote	Name	Type	Description
λ	forget probability	optimization parameter	Probability of removal for a B.
δ	decay rate	optimization parameter	The amount subtracted from λ whenever it drops.
ι_{max}	maximum iteration	optimization parameter	The maximum number of iterations that the heuristic does in its search.
$\|\mathcal{A}\|$	number of agents	optimization parameter	The number of agents in the swarm of FSO.
$\|\mathcal{S}_{elite}\|$	number of elites	optimization parameter	The number of top solutions kept in a generation to be used for next.
s	schedule	object	Timeline where the tasks are planned.
$\mathcal{S}$	schedule set	object	A set of schedules.
t	task	object	Operation that requires space in the schedule timeline.
$\mathcal{T}$	task set	object	A set of tasks.
B	scheduling block	object	Unit of value, a set of dependant tasks and their constraints.
$\mathcal{B}$	scheduling block set	object	A set of scheduling blocks.
pri_{B}	priority	object	Priority value of B.
w	window	object	The time when a task can be scheduled abiding by constraints.
$agent$	agent	object	Operator of FSO that forgets and repairs a schedule.
$\mathcal{A}$	swarm	object	Set of interacting agents in FSO.

APPENDIX
TERMINOLOGY AND DENOTATION

To improve the clarity and consistency of the algorithm description, a set of conventions is followed. The algorithm parameters are denoted using Greek letters (e.g. λ), while commonly used objects, which may comprise multiple variables and functions, are denoted using Latin letters (e.g. $\mathcal{A}$). Other variables have descriptive or abbreviated names with all lowercase letters. The functions are also descriptively denoted with words that start with capital letters. Table 13 describes the important denotations presented in this paper.

REFERENCES

[1] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, Sep. 2003.

[2] A. I. G. de Castro and J. Yáñez, "Optimization of telescope scheduling: Algorithmic research and scientific policy," *Astron. Astrophys.*, vol. 403, no. 1, pp. 357–367, May 2003.

[3] X.-S. Yang, S. Deb, Y.-X. Zhao, S. Fong, and X. He, "Swarm intelligence: Past, present and future," *Soft Comput.*, vol. 22, no. 18, pp. 5923–5933, Sep. 2018.

[4] A. Garcia-Piquer, J. C. Morales, J. Colomé, and I. Ribas, "Evolutionary computation for the ARIEL mission planning tool," in *Proc. 6th Int. Conf. Space Mission Challenges Inf. Technol. (SMC-IT)*, Sep. 2017, pp. 101–106.

[5] N. Nakhjiri, M. Salamó, M. Sànchez-Marrè, and J. C. Morales, "A hybrid multi-start metaheuristic scheduler for astronomical observations," *Eng. Appl. Artif. Intell.*, vol. 126, Nov. 2023, Art. no. 106856.

[6] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search: Framework and applications," in *Handbook of Metaheuristics*. Cham, Switzerland: Springer, 2019, pp. 129–168.

[7] V. K. Prajapati, M. Jain, and L. Chouhan, "Tabu search algorithm (TSA): A comprehensive survey," in *Proc. 3rd Int. Conf. Emerg. Technol. Comput. Eng., Mach. Learn. Internet Things (ICETCE)*, Feb. 2020, pp. 1–8.

[8] N. Andréasson, A. Evgrafov, and M. Patriksson, *An Introduction To Continuous Optimization: Foundations and Fundamental Algorithms*. New York, NY, USA: Dover, 2020.

[9] F. Glover and K. Sörensen, "Metaheuristics," *Scholarpedia*, vol. 10, no. 4, p. 6532, Apr. 2015.

[10] M. D. Johnston, "Spike: AI scheduling for NASA's Hubble space telescope," in *Proc. 6th Conf. Artif. Intell. Appl.*, 1990, pp. 184–190.

[11] M. Johnston, "Automated observation scheduling for the VLT," *Very Large Telescopes Instrum*, vol. 30, p. 1273, Aug. 1988.

[12] T. Sasaki, G. Kosugi, J. A. Kawai, T. Kusumoto, N. Koura, R. Hawkins, L. Kramer, A. P. Krueger, and G. E. Miller, "Observation scheduling scheme for the subaru telescope," *Proc. SPIE*, vol. 4009, pp. 350–354, Jun. 2000.

[13] D. E. Kvasov and M. S. Mukhametzhanov, "Metaheuristic vs. Deterministic global optimization algorithms: The univariate case," *Appl. Math. Comput.*, vol. 318, pp. 245–259, Feb. 2018.

[14] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Comput.*, vol. 8, no. 2, pp. 239–287, Jun. 2009.

[15] N. Thalman, T. Sparn, L. Jaffres, D. Gablehouse, D. Judd, and C. Russell, "AI techniques for a space application scheduling problem," in *Proc. NASA Conf. Publication*, vol. 3110, 1991, p. 83.

[16] A. L. Jaimes and C. A. C. Coello, "Multi-objective evolutionary algorithms: A review of the state-of-the-art and some of their applications in chemical engineering," in *Multi-Objective Optimization: Techniques and Application in Chemical Engineering*. Singapore: World Scientific, 2017, pp. 63–92.

[17] M. E. Giuliano and M. D. Johnston, "Multi-objective evolutionary algorithms for scheduling the James Webb space telescope," in *Proc. Int. Conf. Automated Planning Scheduling*, 2008, pp. 107–115.

[18] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *Proc. Int. Conf. Global Trends Signal Process., Inf. Comput. Commun. (ICGTSPICC)*, Dec. 2016, pp. 261–265.

[19] R. Grim, M. Jansen, A. Baan, J. Van Hemert, and H. D. Wolf, "Use of evolutionary algorithms for telescope scheduling," *Proc. SPIE*, vol. 4757, pp. 51–61, Jul. 2002.

[20] L. J. Eshelman, "Genetic algorithms," in *Evolutionary Computation 1*. Boca Raton, FL, USA: CRC Press, 2018, pp. 102–118.

[21] A. Garcia-Piquer, J. C. Morales, I. Ribas, J. Colomé, J. Guàrdia, M. Perger, J. A. Caballero, M. Cortés-Contreras, S. V. Jeffers, A. Reiners, P. J. Amado, A. Quirrenbach, and W. Seifert, "Efficient scheduling of astronomical observations: Application to the CARMENES radial-velocity survey," *Astron. Astrophys.*, vol. 604, p. A87, Aug. 2017.

[22] W. Mahoney and K. Thanjavur, "Artificial intelligence in autonomous telescopes," in *Telescopes Afar*. San Francisco, CA, USA: Astronomical Society of the Pacific, 2011, p. 38.

[23] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid metaheuristics in combinatorial optimization: A survey," *Appl. Soft Comput.*, vol. 11, no. 6, pp. 4135–4151, Sep. 2011.

[24] H. Garg, "A hybrid PSO-GA algorithm for constrained optimization problems," *Appl. Math. Comput.*, vol. 274, pp. 292–305, Feb. 2016.

[25] C. Blum, "Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling," *Comput. Oper. Res.*, vol. 32, no. 6, pp. 1565–1591, Jun. 2005.

[26] G. Giannone, A. M. Chavan, D. R. Silva, A. P. Krueger, and G. E. Miller, "Long and short term scheduling tools in ESO," in *Astronomical Data Analysis Software and Systems IX*, vol. 216. San Francisco, CA, USA: Astronomical Society of the Pacific, 2000, p. 111.

[27] D. Delahaye, S. Chaimatanan, and M. Mongeau, "Simulated annealing: From basics to applications," in *Handbook of Metaheuristics*. Berlin, Germany: Springer, 2019, pp. 1–35.

[28] P. Hansen, N. Mladenović, J. Brimberg, and J. A. M. Pérez, "Variable neighborhood search," in *Handbook of Metaheuristics*. Cham, Switzerland: Springer, 2019, pp. 57–97.

[29] M. N. Omidvar, X. Li, and X. Yao, "A review of population-based metaheuristics for large-scale black-box global optimization—Part i," *IEEE Trans. Evol. Comput.*, vol. 26, no. 5, pp. 802–822, Oct. 2022.

[30] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary Computation 1: Basic Algorithms and Operators*. Boca Raton, FL, USA: CRC Press, 2018.

[31] M. Dorigo and T. Stützle, *Ant Colony Optimization: Overview and Recent Advances*. Cham, Switzerland: Springer, 2019, pp. 311–351.

[32] J. C. Bansal, "Particle swarm optimization," in *Evolutionary and Swarm Intelligence Algorithms*. Berlin, Germany: Springer, 2019, pp. 11–23.

[33] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: Artificial bee colony (ABC) algorithm and applications," *Artif. Intell. Rev.*, vol. 42, no. 1, pp. 21–57, Jun. 2014.

[34] S. A. Vavasis, "Complexity issues in global optimization: A survey," in *Handbook of Global Optimization*. Cham, Switzerland: Springer, 1995, pp. 27–41.

[35] A. Eiben, J. Smith, A. Eiben, and J. Smith, "Popular evolutionary algorithm variants," in *Introduction to Evolutionary Computing*, 2015, pp. 99–116.

[36] C. Cotta, L. Mathieson, and P. Moscato, "Memetic algorithms," in *Handbook of Heuristics*. Cham, Switzerland: Springer, 2018, pp. 607–638.

[37] E. K. Burke and J. L. Silva, "The design of memetic algorithms for scheduling and timetabling problems," in *Recent Advances in Memetic Algorithms*. Cham, Switzerland: Springer, 2005, pp. 289–311.

[38] J. Michallet, C. Prins, L. Amodeo, F. Yalaoui, and G. Vitry, "Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services," *Comput. Oper. Res.*, vol. 41, pp. 196–207, Jan. 2014.

[39] M. Dell, M. Iori, S. Novellani, and T. Stützle, "A destroy and repair algorithm for the bike sharing rebalancing problem," *Comput. Oper. Res.*, vol. 71, pp. 149–162, Jul. 2016.

[40] D. Pisinger and S. Ropke, "Large neighborhood search," in *Handbook of Metaheuristics*. Berlin, Germany: Springer, 2019, pp. 99–127.

[41] V. Schmid and J. F. Ehmke, "An effective large neighborhood search for the team orienteering problem with time windows," in *Proc. Int. Conf. Comput. Logistics*. Cham, Switzerland: Springer, 2017, pp. 3–18.

[42] M. Wen, E. Linde, S. Ropke, P. Mirchandani, and A. Larsen, "An adaptive large neighborhood search heuristic for the electric vehicle scheduling problem," *Comput. Oper. Res.*, vol. 76, pp. 73–83, Dec. 2016.

[43] S. T. W. Mara, R. Norcahyo, P. Jodiawan, L. Lusiantoro, and A. P. Rifai, "A survey of adaptive large neighborhood search algorithms and applications," *Comput. Oper. Res.*, vol. 146, Oct. 2022, Art. no. 105903.

[44] D. S. Altner, R. K. Ahuja, Ö. Ergun, and J. B. Orlin, "Very large-scale neighborhood search," in *Search Methodologies*, E. K. Burke and G. Kendall, Eds., Cham, Switzerland: Springer, 2014, pp. 339–367. [Online]. Available: https://ideas.repec.org/h/spr/sprchp/978-1-4614-6940-7_13.html

[45] J. C. Morales, N. Nakhjiri, J. Colomé, I. Ribas, E. García, D. Moreno, and F. Vilardell, "Ariel mission planning: Scheduling the survey of a thousand exoplanets," *Experim. Astron.*, vol. 53, no. 2, pp. 807–829, Apr. 2022.

[46] E. Pascale, P. Eccleston, and G. Tinetti, "The ARIEL space mission," in *Proc. 5th IEEE Int. Workshop Metrol. Aerosp. (MetroAeroSpace)*, Jun. 2018, pp. 31–34.

[47] D. G. Pereira, A. Afonso, and F. M. Medeiros, "Overview of Friedman's test and post-hoc analysis," *Commun. Stat. Simul. Comput.*, vol. 44, no. 10, pp. 2636–2653, 2015.

NARIMAN NAKHJIRI received the B.S. degree in computer hardware engineering from Shahid Beheshti University, Tehran, Iran, in 2012, and the M.S. degree in artificial intelligence from Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 2019. He is currently pursuing the Ph.D. degree in computer science with the University of Barcelona, Barcelona.

He has been with the Institute of Space Studies of Catalonia (IEEC), Barcelona, as a Research Engineer, since 2019. He is involved in international projects, such as the Ariel Space Mission of the European Space Agency (ESA) and the Cherenkov Telescope Array (CTA) of the European Southern Observatory (ESO), and national projects in Spain, such as the Joan Oro Robotic Telescope (TJO). His research interests include optimization and scheduling problems focusing on metaheuristics and their applications in real-world projects. Additionally, he has conducted research in the machine learning field, utilizing methodologies, such as decision trees and case-based reasoning.

MARIA SALAMÓ received the B.S. degree in computer science and the Ph.D. degree from Universitat Ramon Llull, Barcelona, Spain, in 1999 and 2004, respectively. She is currently a Professor with the Department of Mathematics and Computer Science, Universitat de Barcelona. She has published more than 60 peer-reviewed papers (including book chapters and papers in leading international conferences and journals). Her research covers a broad range of topics within AI, including, machine learning, recommender systems, user modeling, and personalization techniques. She received the Award for her Ph.D. Project on Information and Communication Technologies.

MIQUEL SÀNCHEZ-MARRÈ received the B.Sc. and M.Sc. degrees in computer science from Barcelona School of Informatics (FIB), in 1988 and 1991, respectively, and the Ph.D. degree in computer science (artificial intelligence) from Universitat Politècnica de Catalunya (UPC), Barcelona, in 1996.

He has been an Associate Professor (tenure) with the Computer Science (CS) Department, UPC, since 1997. He is currently a member with the Intelligent Data Science and Artificial Intelligence Research Centre (IDEAI-UPC). He co-founded the spin-off company ''Sanejament Intelligent S.L. (SISLtech)'' devoted to advanced control and supervision of environmental systems. He has authored more than 250 peer-reviewed publications and is the author of ten books. His main research interests include case-based reasoning, intelligent decision support systems, recommender systems, machine learning, data science, knowledge engineering, integrated AI architectures, and AI applied to environmental, industrial, and health systems. He has been a fellow of the International Environmental Modelling and Software Society (iEMSs), since 2005. He is a Pioneer Member of the Catalan Association of Artificial Intelligence (ACIA) and a member of Spanish Association of Artificial Intelligence (AEPIA). He co-organized the first Environment and Artificial Intelligence Workshop in Europe: Binding Environmental Sciences and Artificial Intelligence (BESAI 98) at ECAI Conference, in 1998. He has organized some international events in the AI field related to the Environment (BESAI-ECAI, AAAI, and iEMSs).

CHRISTIAN BLUM received the Ph.D. degree in applied sciences from the Free University of Brussels, Brussels, Belgium, in 2004. He is currently a Senior Research Scientist with the Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain. His research interests include solving difficult optimization problems using swarm intelligence techniques and combinations of metaheuristics with exact techniques.

JUAN CARLOS MORALES received the B.S. and Ph.D. degrees in physics from the University of Barcelona, Spain, in 2006 and 2010, respectively.

He has been a Senior Postdoctoral Researcher with the Institute of Space Studies of Catalonia (IEEC) and the Institute of Space Sciences (ICE, CSIC), Barcelona, Spain, since 2015. He has participated in more than 120 papers in peer-reviewed journals and several international conferences. The main research line is the study of low-mass stars and the exoplanets they host. His research interests include the determination of the fundamental properties of low-mass stars (i.e., stars with a mass below that of the Sun) and the effects that stellar magnetic activity causes on the structure of such stars. He also works on the discovery of exoplanetary systems, mainly by means of the radial velocity method. He is also participating in the development of future space missions devoted to exoplanet research, in particular coordinating the development of scheduling tools.