IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2022.Doi Number

BAGESS: A software module based on a genetic algorithm to sequentially order load-balancing evaluation scenarios over smartphone-based clusters at the Edge

Virginia Yannibelli[1], Matías Hirsch[1], Juan Toloza[1], Tim A. Majchrzak[2], Tor-Morten Grønli[3], Alejandro Zunino[1] and Cristian Mateos[1]

[1] ISISTAN (UNICEN-CONICET), Tandil, Buenos Aires, Argentina
[2] University of Agder, Kristiansand, Norway
[3] Kristiania University College, Oslo, Norway

Corresponding author: Tim A. Majchrzak (e-mail: timam@uia.no).

ABSTRACT Due to the increasing interest in employing smartphones as first-class citizens in high-performance Edge computing environments, the necessity of software to facilitate the evaluation of load-balancing strategies for smartphone-based clusters has emerged. Regarding this, to select the best strategy for a cluster with m smartphones, usually a number of g candidate strategies are evaluated based on a number of r scenarios that contain these smartphones, which differ in terms of the start battery levels required for these smartphones. Thus, each of the r scenarios must be *prepared* before evaluating each of the g strategies on each r_i, so that the smartphones have the required start battery levels pre-configured for r_i, which requires discharging or charging smartphones. This leads to a number of $e = r*g$ scenario preparation events that must be sequentially developed, considering that the time required to develop each event depends on the previous event. Thus, the single-objective problem addressed here implies finding out the sequential order in which the events should be developed, so that the total time required to develop them is minimized. This problem is modeled as the ATSP (Asymmetric Traveling Salesman Problem), since defining the sequential order to develop the events is equivalent to defining the sequential order to visit the cities, and therefore, is an NP-Hard problem. Given the complexity of this problem, the novel software module BAGESS (Battery Aware Green Edge Scenario Sequencer) is proposed, which uses a genetic algorithm for defining the sequential order to develop the events. BAGESS's performance outperforms those of the methods currently used for the problem, reaching significant savings regarding the time required to develop the events in the range [12, 85]%.

INDEX TERMS edge computing, smartphone, profiling, benchmarking, evolutionary computing.

I. INTRODUCTION

In the last few years, the use of smartphones has increased significantly, and continues to increase [6]. Contemporary smartphone models have high processing power and memory capacity, and long-lasting rechargeable batteries, among other valuable features. Due to these reasons, high-performance Edge computing environments consider smartphones as valuable computing resources, and promote the creation and use of smartphone-based clusters to distribute the execution of the tasks inherent to different kinds of Edge applications on the smartphones in the cluster. One example is distributing the tasks from AI (Artificial Intelligence) applications aimed to identify objects in images taken in urban environments via neural network models.

In this context, to distribute the tasks inherent to a given application on the smartphones that compose the cluster, it is necessary to use an appropriate load-balancing strategy. The load-balancing strategies usually decide which tasks of the application are assigned to each smartphone, so that a given optimization objective is reached. In this respect, many load-balancing strategies have been proposed in the

literature for smartphone-based clusters [6], which differ in several aspects, including the optimization objectives considered (e.g., to minimize the time required to execute the tasks), the kind of algorithm utilized to decide the assignment of the tasks (e.g., heuristic and metaheuristic algorithms), and the smartphone attributes considered to decide such assignment (e.g., battery level, and CPU load).

For selecting the best load-balancing strategy for a given cluster with *m* smartphones, the performance of a number of *g* candidate strategies should be evaluated on a number of *r* different scenarios inherent to the cluster. These *r* scenarios represent different initial states of the cluster. Specifically, these scenarios are composed by the *m* smartphones in the cluster, but differ in relation to the start battery levels, which are required for the *m* smartphones. Therefore, each of the *r* scenarios must be *prepared* before evaluating each of the *g* strategies on it. With respect to this, as detailed in [1], the preparation of a given scenario involves applying battery charge/discharge actions on the smartphones in the scenario, in order to reach the start battery levels required for these smartphones from the corresponding current battery levels, at the minimal possible time for the set of *m* smartphones.

Thus, considering that each one of the *r* scenarios must be prepared before evaluating each of the *g* strategies on it, this leads to a number of $e = r*g$ scenario preparation events. Since these *e* events aim to prepare scenarios composed by the same *m* smartphones, these *e* events must be sequentially developed one at a time. In addition, when these *e* events are sequentially developed, the current battery levels of the *m* smartphones in the scenario of each event are affected by the previous event in the sequential order. As a consequence of this, the time required for developing each event depends on the previous event in the sequential order. Therefore, the problem of defining the sequential order in which these *e* events should be developed, considering that the total time required to develop them should be minimized, is relevant in the context of facilitating the evaluation of diverse candidate load-balancing strategies for smartphone-based clusters.

In connection with the ideas previously mentioned, different software/hardware tools [5, 6, 7, 8] have been proposed in the literature for simulating smartphone-based clusters. These tools allow researchers to simulate the creation of a cluster composed by smartphones of different models, and to define the attributes of such smartphones (e.g., current battery level, CPU load, screen state, and charge/discharge profiles). Moreover, these tools simulate the execution of a given load-balancing strategy on a created smartphone-based cluster, considering the attributes defined for the smartphones, and also the tasks of a given application. To complement the tools presented in recent work [7, 8], a software module was recently proposed [1], which aims at preparing a given scenario inherent to a created smartphone-based cluster. This module defines the battery charge/discharge actions that should be applied on the smartphones in the scenario, to reach the start battery levels required for these smartphones in such a scenario, from the corresponding current battery levels. However, to the best of our knowledge, the tools proposed so far in the literature do not contain software aimed to facilitate the evaluation of a number of candidate load-balancing strategies on a number of diverse scenarios inherent to a smartphone-based cluster created.

Thus, again, the problem addressed in this paper implies defining the sequential order in which a given number of *e* scenario preparation events should be developed, in such a way that the total time required to develop them is minimized. This problem is modeled here as the well–known ATSP (Asymmetric Traveling Salesman Problem) [2]. As the events to be sequentially developed can be modeled as the cities to be visited, the time required to develop each of these events can be modeled as the cost of visiting each of the cities, and thus defining the sequential order for developing the events is equivalent to defining the sequential order to visit the cities. By modeling the addressed problem as the ATSP, the complexity of the addressed problem is equivalent to the complexity of the ATSP; ATSP is a NP-Hard problem [2].

A novel software module named BAGESS (Battery Aware Green Edge Scenario Sequencer) is proposed here, which uses a single-objective genetic algorithm to determine the best time-effective sequential order to develop the *e* events. We utilize a genetic algorithm specifically since the addressed problem is modeled as the ATSP. Genetic algorithms have been shown to be effective to solve the ATSP, achieving near-optimal solutions, and sometimes optimal solutions, for medium and large ATSP instances [2]. Therefore, a genetic algorithm is an appropriate alternative for the addressed problem.

The main contributions of this paper are:

– A mathematical model for the addressed problem of determining the sequential order in which a given number of *e* scenario preparation events should be developed, so that the total time required to develop them is minimized. This model is defined based on the recognized mathematical model proposed by Dantzig et al. [19] for the Traveling Salesman Problem (TSP) and then extended for the ATSP.

– An introduction to the software module BAGESS, which utilizes a genetic algorithm for defining the sequential order to develop the given *e* events. This genetic algorithm has been designed to: *a*) explore different feasible sequential orders for developing the *e* events one at a time, and *b*) identify the sequential order that allows developing these *e* events at the minimal possible time.

– An experimental evaluation of the genetic algorithm utilized by BAGESS. Specifically, the genetic algorithm's performance was evaluated on 540 different instances of the addressed problem, and then was compared with those of the two methods currently used to determine the sequential order to develop a given number of *e* scenario preparation events. The obtained results indicate that the algorithm outperforms the two mentioned methods,

achieving significant savings in terms of the total time required to develop the *e* events (i.e., savings on [12, 85]%). The significance of these results was validated with the Mann–Whitney U statistical test [4], with a confidence level of $\alpha = 0.001$.

– BAGESS's source code in Java 1.8, which includes the genetic algorithm's code, publicly available for reuse and adaptation. Moreover, the above-mentioned 540 problem instances are publicly available for experimentation.

The remainder of the paper is organized as follows. In Section II, the addressed problem is described in detail, and then the mathematical model of this problem is presented. In Section III, the software module BAGESS is presented, and then the genetic algorithm used by BAGESS is described in detail, and the computational complexity of this algorithm is analyzed. In Section IV, the computational experiments that were developed for evaluating the performance of the genetic algorithm used by BAGESS are presented, and after that the results obtained by these experiments are presented and analyzed in detail. In Section V, the main implications and limitations of BAGESS are discussed. In Section VI, related works from literature are reviewed. Finally, in Section VII, the conclusion of this work and future work are presented.

II. PROBLEM DESCRIPTION: SEQUENCING LOAD-BALANCING SCENARIOS

A. LOAD-BALANCING SCENARIOS

Suppose that a given number *g* of load-balancing strategies are being considered as candidates to distribute the workload on a cluster of *m* smartphones. To determine the best of these strategies for achieving a specific objective on this cluster (e.g., complete a set of tasks in the minimum time), the load balancing performance of each strategy *j* (*j* = 1, …, *g*) should be evaluated on a given number *r* of different scenarios inherent to the cluster. Regarding this, each scenario *i* (*i* = 1, …, *r*) is composed of the *m* smartphones belonging to the cluster. In each scenario *i*, there are predefined target battery levels for the *m* smartphones, which are required to evaluate any strategy *j* on the scenario *i*. Besides, in each scenario *i*, the *m* smartphones have the current battery levels associated with them. To evaluate the load balancing performance of any strategy *j* on scenario *i*, it is necessary to prepare such a scenario.

The preparation of the scenario involves applying sequences of battery charge/discharge actions on the *m* smartphones in the scenario, to reach the target battery levels predefined for these *m* smartphones from the corresponding current battery levels, at the minimal possible time for the set of *m* smartphones [1]. In this respect, the minimal possible time to reach the target battery levels predefined for the *m* smartphones depends on the difference between the current and target battery levels of these smartphones. Specifically, the higher the difference between the current and target battery levels of these *m* smartphones, the higher the time to reach the target battery levels predefined for these smartphones. Once the *m* smartphones in the scenario have reached their target battery levels, the scenario is considered ready (or prepared) to evaluate any of the strategies. For a detailed description of the scenario preparation problem, and the most recent approach proposed for it, we refer to [1].

Considering the above-mentioned, each one of the *r* scenarios should be prepared before evaluating each of the *g* strategies over it. This means that each one of the *r* scenarios should be prepared as many times as the number *g* of strategies to be evaluated on it. This leads to a number of $e = r * g$ scenario preparation events. These *e* events differ in terms of the scenario *i* to be prepared (i.e., events differ in terms of the target and current battery levels considered for the *m* smartphones) and/or the strategy *j* to be evaluated. In addition, given that these *e* events aim to prepare scenarios composed by the same *m* smartphones, the *e* events should be developed sequentially one at a time. In this context, the sequential order in which the *e* events are developed is really important because the development of the *k*-th event depends on the development of the (*k*-1)-th event, considering *k* = 2, …, *e*. Specifically, suppose that the *k*-th event aims to prepare the scenario sce_a in order to evaluate the strategy str_c, whereas the (*k*-1)-th event aims to prepare the scenario sce_b in order to evaluate the strategy str_f, considering $sce_a \neq sce_b$ and/or $str_c \neq str_f$. In the *k*-th event, to prepare the scenario sce_a, first it is necessary to determine the current battery level of each one of the *m* smartphones, and after that, the scenario preparation approach proposed in [1] is used to determine the sequences of battery charge/discharge actions to be applied on the *m* smartphones, in order to reach the target battery levels predefined for these smartphones from the current battery levels, at the minimal possible time for the set of *m* smartphones. Thus, the time required to prepare the scenario sce_a is equivalent to the time amount required to apply the sequences of battery charge/discharge actions indicated by the mentioned approach on the *m* smartphones.

In order to determine the current battery level $cbl_{s,sce(k)}$ of each smartphone *s* (*s* = 1, …, *m*) in the scenario sce_a to be prepared by the *k*-th event, it is necessary to consider that $cbl_{s,sce(k)}$ depends on two factors inherent to the (*k*-1)-th event. The first factor $tbl_{s,sce(k-1)}$ refers to the target battery level predefined for each smartphone *s* in the scenario sce_b to be prepared by the (*k*-1)-th event. This factor means once the scenario sce_b is prepared, the battery level of each smartphone *s* is equal to $tbl_{s,sce(k-1)}$. The second factor $d_{s,str(k-1)}$ refers to the battery level variation that each smartphone *s* suffers once the strategy str_f considered in the (*k*-1)-th event is evaluated on the prepared scenario sce_b. This factor is relevant because the scenario sce_b is prepared in order to evaluate the strategy str_f over it. The evaluation of str_f on sce_b implies that str_f will distribute workload on the *m* smartphones, and thus, the battery level $tbl_{s,sce(k-1)}$ of each of the *m* smartphones will be subject to a decrease $d_{s,str(k-1)}$. Therefore, once str_f is evaluated on sce_b, the battery level of each smartphone *s* will be equal to $tbl_{s,sce(k-1)} - d_{s,str(k-1)}$. Then,

given that the k-th event is carried out immediately after that the (k-1)-th event is finished, the current battery level $cbl_{s,sce(k)}$ of each smartphone s will be equivalent to $tbl_{s,sce(k-1)} - d_{s,str(k-1)}$. It is necessary to mention that the values of the terms $cbl_{s,sce(k)}$, and besides the predefined values of the terms $tbl_{s,sce(k-1)}$, are integer values on the range [0, 100]%. In addition, the values of the terms $d_{s,str(k-1)}$ are predefined integer values on the range [0, 100]%.

The sequential order in which the e scenario preparation events are developed determines the current battery levels considered by each of these events for the m smartphones to be prepared, and consequently, impacts the time required by each of these events to prepare the m smartphones. Having this in mind, the scenario sequencing problem addressed here involves defining the sequential order in which the e scenario preparation events should be developed, in such a way that the total time required to develop these e events is minimized.

B. INSTANCE OF THE PROBLEM

Fig. 1 shows an instance of the addressed problem. In this instance, a number of r = 3 scenarios, which are composed by the same 3 smartphones, and a number of g = 2 load balancing strategies, are considered. Then, from these scenarios and strategies, a number of e = 6 scenario preparation events are considered, in order to prepare each of the 3 scenarios before evaluating each one of the 2 strategies on it.

Fig. 1.a presents the three considered scenarios and details the target battery level $tbl_{s,i}$ predefined for each smartphone s in each scenario i. Then, Fig. 1.b presents the two strategies considered, and indicates the battery variation $d_{s,j}$ each smartphone s can suffer after strategy j is evaluated on the set of 3 smartphones. Fig. 1.c presents the six events defined from the three scenarios and the two strategies, indicating the scenario to be prepared by each event, and the strategy to be evaluated on the scenario prepared by each event. Finally, Fig. 1.d details the current battery level $cbl_{s,sce(1)}$ to be considered for each smartphone s in the first event to be developed. In this sense, it is important to note that the first event to be developed will be determined after the sequential order of the e events is determined. This is described below in detail.

Fig. 2.a shows a feasible sequential order to develop the six events of Fig. 1.c one at a time. This order indicates that the first event to be developed is event 3. Then, the events to be developed in second and third place are events 4 and 5, respectively. After that, the events to be carried out in fourth and fifth place are 6 and 1, respectively. Finally, the sixth event to be developed is event 2.

To develop the six events following the sequential order given in Fig. 2.a, it is required to define the current battery level to be considered for each smartphone s in the scenario of each event, considering the place given to each event in the sequential order. In the case of the first event in the sequential order (i.e., event 3), the current battery level $cbl_{s,sce(1)}$ of each smartphone s is a predefined integer value on the range [0, 100]% (as detailed in Fig. 1.d, with the purpose of representing the battery level of the smartphone s at the moment of developing the first event). Unlike this, in the case of the k-th event in the sequential order, considering k = 2, …, 6, the current battery level $cbl_{s,sce(k)}$ of each smartphone s is $tbl_{s,sce(k-1)} - d_{s,str(k-1)}$, as was previously described. For example, in the case of the second event (i.e., event 4), the current battery level $cbl_{1,sce(2)}$ of the smartphone 1 is calculated as $tbl_{1,sce(1)} - d_{1,str(1)}$. Note that $tbl_{1,sce(1)}$ represents the target battery level of the smartphone 1 in the scenario of the first event (i.e., scenario 2), and is 90% (as detailed in Fig. 1.a). Then, $d_{1,str(1)}$ represents the decrease in the battery level $tbl_{1,sce(1)}$ of smartphone 1 once that the strategy considered in the first event (i.e., the strategy 1) is evaluated on the scenario of the first event, and is 10% (as detailed in Fig. 1.b). Thus, the current battery level $cbl_{1,sce(2)}$ of the smartphone 1 in the second event is 90% – 10% = 80%. Fig. 2.b presents in detail the current battery level $cbl_{s,sce(k)}$ determined for each smartphone s in the k-th event of the sequential order, for all k.

Once the current battery levels of the 3 smartphones in the scenario of each event are defined, the events are developed following the sequential order given in Fig. 2.a. This means that, for each event, the scenario inherent to the event is prepared, according to the scenario preparation approach proposed in [1]. Specifically, this approach is applied to the scenario, and indicates the sequences of battery charge/discharge actions to be applied on the three smartphones, in order to reach the target battery levels of these 3 smartphones from the current battery levels. Then, the sequences of actions indicated by the approach are applied on the three smartphones in the scenario. Thus, the time required to prepare the scenario is equivalent to the time required to apply the mentioned sequences of actions. Fig. 2.c shows the time required to prepare the scenario of each one of the 6 events, according to the sequences of actions indicated by the mentioned approach.

It is important to mention that other feasible sequential orders can be defined to develop these six events. Given that there are no precedence relationships between the events, each one of the six events can be developed immediately after any of the other 5 events. Thus, a number of 6! feasible sequential orders can be defined to develop the six events. Fig. 3.a shows other feasible sequential order to develop the 6 events, which is different from the one detailed in Fig. 2.a. Then, Fig. 3.b details the current battery level $cbl_{s,sce(k)}$ determined for each smartphone s in the scenario of the k-th event of this order, for all k. Finally, Fig. 3.c presents the time required to prepare the scenario of each event, according to the sequences of actions indicated by the previously mentioned approach. Note that, as was previously explained, the lower the difference between the current and target battery levels of the three smartphones in the scenario of an

event, the lower the time required by the sequences of actions indicated to reach the target battery levels of these smartphones from the current battery levels, and thus, the lower the time required to prepare the scenario. For example, in the case of the scenario of the event 4 (i.e., scenario 2), the target battery levels $tbl_{s,i}$ of the smartphones (i.e., 90%, 96% and 34% for smartphone 1, 2 and 3, respectively) are closer to the current battery levels $cbl_{s,sce(2)}$ defined for these smartphones in Fig. 2.b (i.e., 80%, 86% and 17% for smartphone 1, 2 and 3, respectively) than the current battery levels $cbl_{s,sce(5)}$ defined for these smartphones in Fig. 3.b (i.e., 22%, 10% and 21% for smartphone 1, 2 and 3, respectively). For this reason, the time required to prepare the scenario of the event 4 in Fig. 2.c is much lower than that required to prepare this scenario in Fig. 3.c.

a)

$tbl_{s,i}$	Scenario		
Smartphone	$i = 1$	$i = 2$	$i = 3$
$s = 1$	42%	90%	92%
$s = 2$	28%	96%	51%
$s = 3$	35%	34%	22%

b)

$d_{s,j}$	Strategy	
Smartphone	$j = 1$	$j = 2$
$s = 1$	10%	20%
$s = 2$	10%	18%
$s = 3$	17%	14%

c)

Event	Scenario	Strategy
1	1	1
2	1	2
3	2	1
4	2	2
5	3	1
6	3	2

d)

Smartphone	$cbl_{s,sce(1)}$
$s = 1$	65%
$s = 2$	95%
$s = 3$	49%

FIGURE 1. **Instance of the addressed scenario sequencing problem.**

a)

	1st	2nd	3rd	4th	5th	6th
Event	3	4	5	6	1	2

b)

Smartphone	$cbl_{s,sce(1)}$	$cbl_{s,sce(2)}$	$cbl_{s,sce(3)}$	$cbl_{s,sce(4)}$	$cbl_{s,sce(5)}$	$cbl_{s,sce(6)}$
$s = 1$	65%	90%-10%=80%	90%-20%=70%	92%-10%=82%	92%-20%=72%	42%-10%=32%
$s = 2$	95%	96%-10%=86%	96%-18%=78%	51%-10%=41%	51%-18%=33%	28%-10%=18%
$s = 3$	49%	34%-17%=17%	34%-14%=20%	22%-17%=5%	22%-14%=8%	35%-17%=18%

c)

Time per Event (minutes)	***time*(3)**	***time*(4)**	***time*(5)**	***time*(6)**	***time*(1)**	***time*(2)**	**Total time**
	43.36	19.12	83.99	11.16	90.25	5.21	253.09

FIGURE 2. **Feasible solution for the problem instance shown in Fig. 1.**

a)

	1st	2nd	3rd	4th	5th	6th
Event	1	3	5	2	4	6

b)

Smartphone	$cbl_{5,sce(1)}$	$cbl_{5,sce(2)}$	$cbl_{5,sce(3)}$	$cbl_{5,sce(4)}$	$cbl_{5,sce(5)}$	$cbl_{5,sce(6)}$
$s = 1$	65%	42%-10%=32%	90%-10%=80%	92%-10%=82%	42%-20%=22%	90%-20%=70%
$s = 2$	95%	28%-10%=18%	96%-10%=86%	51%-10%=41%	28%-18%=10%	96%-18%=78%
$s = 3$	49%	35%-17%=18%	34%-17%=17%	22%-17%=5%	35%-14%=21%	34%-14%=20%

c)

Time per Event (minutes)	*time*(1)	*time*(3)	*time*(5)	*time*(2)	*time*(4)	*time*(6)	**Total time**
	213.64	134.20	108.85	117.59	139.67	83.99	797.94

FIGURE 3. Another feasible solution for the problem instance shown in Fig. 1.

C. MATHEMATICAL FORMULATION OF THE PROBLEM

The previously described scenario sequencing problem has significant similarities with the known ATSP (Asymmetric Traveling Salesman Problem) [2]. Such similarities are explained below in detail.

- The e scenario preparation events to be sequentially developed are considered as equivalent to the e cities to be sequentially visited in the ATSP.
- Each of the e events can be developed immediately after any of the other events (i.e., there are no precedence relationships between events). This is similar to the fact that each one of the e cities can be visited immediately after any of the other cities (i.e., a fully connected network of e cities is considered in ATSP).
- The time required to develop the event h, immediately after developing event q, is considered as equivalent to the cost of visiting the city h, immediately after visiting the city q ($h, q \in \{1, \ldots, e\}$ and $h \neq q$).
- The sequential order to be defined for developing the e events, which includes each event only once, is considered as equivalent to the sequential order to be defined for visiting the e cities, which includes each city only once. However, these sequential orders have the following difference: In the sequential order for visiting the e cities, it is considered that once the e-th city is visited, it is necessary to go back to the first visited city. Unlike this, in the sequential order for developing the e events, once developed the e-th event indicated in this order, no other event must be developed.
- The total time required to develop the e events, according to a sequential order given for these e events, is considered as equivalent to the total cost of visiting the e cities, according to a sequential order given for these e cities. Nevertheless, to calculate the total time required for developing the e events, it is considered the time to develop the first event indicated in the given order, and then the time required to develop the k-th event in the given order immediately after of developing the (k-1)-th event, for all k (k = 2, …, e). In contrast to this, to calculate the total cost of visiting the e cities, it is considered the cost of visiting the k-th city indicated in the giver order immediately after visiting the (k-1)-th city, for all k, and also the cost of going back to the first visited city once visited the e-th city in the order.
- Determining the sequential order in which the e events should be developed, so that the total time required for developing these events is minimized. This is considered equivalent to determining the sequential order in which the e cities should be visited, in such a way that the total cost of visiting these cities is minimized.

Therefore, the mathematical formulation of the scenario sequencing problem is defined based on the formulation proposed by Dantzig et al. [19] for the TSP and then extended for the ATSP, and is presented below.

$$min\left(t_z + \sum_{h \in E}\sum_{q \in E} t_{hq} * x_{hq}\right) \qquad z \in E; u_z = 1; h \neq q \tag{1}$$

subject to:

$$\sum_{h \in E} x_{hq} = 0 \qquad q \in E; q \neq h; u_q = 1 \tag{2}$$

$$\sum_{h \in E} x_{hq} = 1 \qquad q \in E; q \neq h; 2 \leq u_q \leq e \tag{3}$$

$$\sum_{q \in E} x_{hq} = 1 \qquad h \in E; h \neq q; 1 \leq u_h \leq e - 1 \tag{4}$$

$$\sum_{q \in E} x_{hq} = 0 \qquad h \in E; h \neq q; u_h = e \tag{5}$$

$$\sum_{h \in S} \sum_{q \in S} x_{hq} \leq |S| - 1 \qquad q \neq h; S \subset E; 2 \leq |S| \leq e - 2 \tag{6}$$

In this mathematical formulation, E is the set of e events to be sequentially ordered, $E = \{1,\ldots,e\}$.Then, t_z is the time required to develop the event z, when z is the first event in the sequential order, and thus the development of z does not depend on a previous event, and t_{hq} is the time required to develop the event q, when q is immediately preceded by the event h in the sequential order, and so the development of q depends on the development of h. Recall that, as described in Sections II.A and II.B, the time required to develop any of the e events refers to the amount of time required to prepare the scenario inherent to the event (i.e., to apply the sequences of battery charge/discharge actions indicated by the scenario preparation approach [1] on the m smartphones of the scenario inherent to the event).

This mathematical formulation considers binary decision variables x_{hq} and integer decision variables u_z. In this respect, the variables x_{hq} indicate if the event q is directly preceded by the event h in the sequential order ($x_{hq} = 1$) or not ($x_{hq} = 0$). Besides, the variables u_z indicate the position of the event z in the sequential order, considering $u_z = 1,\ldots, e$.

Moreover, this formulation has one objective function that is defined by Eq. (1). This function aims to minimize the total time required to sequentially develop the e events one at a time, and is subject to the constraints defined by Eq. (2)-(6). In this respect, the constraints defined by Eq. (2)-(5) guarantee that each of the e events is included exactly once in the sequential order. This is guaranteed if each of the events in the positions [2, e] of the sequential order has exactly one immediate predecessor event (Eq. (3)), and each of the events in the positions [1, e-1] of the sequential order has exactly one immediate successor event (Eq. (4)). Besides, the event in the position 1 of the sequential order must not be preceded by other event (Eq. (2)), and the event in the position e of the sequential order must not be succeeded by other event (Eq. (5)). Finally, the constraints defined by Eq. (6) guarantee a single sequential order that includes the e events, preventing partial sequential orders that include a subset of the e events. These constraints are the well-known sub-tour elimination constraints of Dantzig et al. [19].

The presented mathematical formulation of the addressed scenario sequencing problem is based on the mathematical formulation proposed by Dantzig et al. [19] for the TSP and then extended for the ATSP, but differs from the latter in the following aspects. Firstly, in addition to considering the variables x_{hq} of the formulation by Dantzig et al., the variables u_z are considered. Secondly, to define the objective function (Eq. (1)), the term t_z and the variables u_z were included in the objective function of the formulation by Dantzig et al. Finally, since it is necessary to guarantee that the first event in the sequential order is not preceded by other event, and the e-th event in the sequential order is not succeeded by other event, all the constraints of the formulation by Dantzig et al. (excepting the sub-tour elimination constraints) were adapted by including the variables u_z, which resulted in the constraints defined by Eq. (3) and the constraints defined by Eq. (4). Besides, the constraints defined by Eq. (2) and the constraints defined by Eq. (5) were added to the formulation.

D. COMPLEXITY OF THE PROBLEM

Due to the similarities of the addressed scenario sequencing problem with the ATSP, it is possible to claim that the complexity of the scenario sequencing problem is equal to the complexity of the ATSP. In this respect, the ATSP is known to be an NP-Hard problem [2].

Given the complexity of the ATSP, during the last two decades, different kinds of algorithms have been proposed in the literature for solving this problem, including exact and meta-heuristic algorithms. In this sense, the exact algorithms (e.g., branching and shearing, and dynamic programming) guarantee optimal solutions for the ATSP instances. However, the runtime of these algorithms grows exponentially with the number of cities to be visited, and therefore are usually considered for solving only small ATSP instances [20]. On the other hand, the meta-heuristic algorithms (e.g., tabu search algorithms, simulated annealing algorithms, genetic algorithms, and swarm intelligence algorithms) are aimed to obtain high-quality solutions (not necessarily the optimal solutions) for the ATSP instances, in a reasonable runtime. Because of this reason, these algorithms are generally considered by the research community and practitioners to solve medium and large ATSP instances [20].

With regard to the mentioned meta-heuristic algorithms, genetic algorithms proposed in the last years for solving the ATSP have reached near-optimal solutions, and sometimes optimal solutions, for medium and large ATSP instances in a reasonable runtime [2]. Besides, unlike the other meta-heuristic algorithms, these genetic algorithms propose a very natural and simple encoding for the ATSP solutions (i.e., solutions are encoded as permutations of the e cities to be visited), and also propose search operators (i.e., crossover and mutation operators feasible for permutations of the e cities) that can be easily implemented, require a very low runtime (i.e., the computing time complexity of these operators is $O(e)$), and allow an effective exploration and exploitation of the solution space inherent to the ATSP instances..

Considering all the above-mentioned, a genetic algorithm is proposed in the next section for the addressed scenario sequencing problem (Section III), with the aim of achieving high-quality solutions in an acceptable runtime, and thereby outperforming the solutions that are provided by the two

methods currently utilized for the addressed problem (Section IV.C).

III. BAGESS

To address the previously described scenario sequencing problem, we have developed a novel software module called BAGESS (Battery Aware Green Edge Scenario Sequencer). This module uses a genetic algorithm designed to determine the best sequential order in which a given number of *e* scenario preparation events should be developed. We describe in detail below the input data of this module, and then the general behavior and the main components of the mentioned genetic algorithm that is considered as the core of BAGESS.

A. INPUT DATA

As input data, BAGESS receives a given number of *r* scenarios, which are composed by the same *m* smartphones. Each scenario *i* contains a predefined target battery level $tbl_{s,i}$ for each smartphone *s*, where this level is an integer value on the range [0, 100]%. Moreover, each of the *m* smartphones has associated real timestamped battery charge/discharge profiles. These profiles provide the time (in milliseconds) required to charge/discharge the battery of the smartphone from a given level to another given level.

Besides, BAGESS receives a given number of *g* strategies to be evaluated on each scenario *i*. For each strategy *j*, the estimated battery decrease $d_{s,j}$ that can be suffered by each smartphone *s* once the strategy *j* is evaluated on the set of *m* smartphones is given, where this decrease is an integer value on the range [0, 100]%.

Finally, BAGESS receives a given number of $e = r * g$ scenario preparation events. Each of these *e* events includes one scenario from the given *r* scenarios and one strategy from the given *g* strategies. Besides, these *e* events differ regarding the scenario and/or the strategy included. These *e* events are predefined in this way since each one of the *r* scenarios should be prepared before evaluating each one of the *g* strategies over it.

It is necessary to note that in each scenario *i* received by BAGESS as input data, the current battery level $cbl_{s,i}$ of each smartphone *s* is unknown. This is due to, as detailed in Section II, the current battery level of each smartphone *s* in the scenario inherent to each event is calculated once the sequential order of the events is determined. In this respect, BAGESS only receives the current battery level to be considered for each smartphone *s* in the first event of the sequential order.

B. GENETIC ALGORITHM

After BAGESS receives the input data, it applies the genetic algorithm. This algorithm has been specially designed to explore different feasible sequential orders for developing the given *e* events one at a time, with the aim of finding the sequential order that allows developing these *e* events at the minimal possible time.

1) GENERAL BEHAVIOR

This genetic algorithm (see Algorithm 1) follows an iterative behavior, and starts by generating an initial population with a number of *p* feasible encoded solutions. Each one of these encoded solutions represents a feasible sequential order to develop the given *e* events one at a time. Then, the algorithm evaluates each of these encoded solutions by a fitness evaluation process, regarding the considered optimization objective: minimizing the time required to sequentially develop the *e* events one at a time.

In each one of the iterations, the algorithm applies a parent selection process on the current population, with the aim of defining which solutions will make up the mating pool, and thus, will be used to create new encoded solutions. Specifically, the algorithm utilizes the parent selection process tournament selection [3], under a tournament size *k*, for encouraging the selection of varied high–fitness solutions with respect of the sequential order indicated to develop the *e* events. Then, the algorithm organizes the solutions in the mating pool into pairs, and applies a crossover process on each pair of solutions, under a probability P_c, for creating a pool of new solutions. In this respect, the crossover process applied by the algorithm is feasible for the used solution encoding, and creates new solutions by combining the sequential orders indicated in the parent solutions to develop the *e* events. Then, the algorithm applies a mutation process on each new created solution, under a probability P_m, to introduce diversity in the pool of new created solutions. In this sense, the mutation process applied by the algorithm is feasible for the solution encoding utilized, and modifies the sequential order indicated to develop the *e* events. Subsequently, the algorithm evaluates each new created solution, by the fitness evaluation process. After that, the algorithm applies a survival selection process on the current population and the pool of new created solutions, to determine which solutions will make up the population for the subsequent iteration. Specifically, the algorithm uses the survival selection process steady-state selection [3], under a replacement percentage *c*, with the aim of maintaining the best encoded solutions generated until the current iteration.

The algorithm iterates until a given number of *I* iterations is reached. After this stop condition is achieved, the algorithm provides BAGESS the best solution of the last population, as the solution obtained to sequentially order the *e* events to be developed.

Algorithm 1 Pseudocode of the genetic algorithm

Input: *e*: events to be ordered sequentially
I: the maximum number of iterations
p: number of solutions in the population
k: tournament size (tournament selection)
P_c: crossover probability
P_m: mutation probability
c: replacement percentage (steady-state selection)
Output: *solution* (sequential order of the *e* events)

```
pop = generate_initial_population(p, e);
fitness_evaluation(pop);
i = 1 (record the number of iterations);
while i <= I do
    mating_pool = tournament_selection(pop, k);
    offspring = crossover(mating_pool, P_c);
    mutation(offspring, P_m);
    fitness_evaluation(offspring);
    pop = steady_state_selection(pop, offspring, c);
    i = i + 1;
end while
solution = best_solution(pop);
return solution;
```

2) SOLUTION ENCODING

Each one of the solutions in the population of this algorithm is represented as an *e*-tuple $<o_1, o_2, \ldots, o_e>$, where *e* is the number of events to be sequentially developed. Each of the terms o_h (h=1,…, *e*) represents a different event from the *e* events. Thus, each solution includes the *e* events once (i.e., each solution is a permutation of the *e* events), and represents a feasible sequential order to develop the *e* events one at a time.

To generate each one of the *p* solutions for the initial population of the algorithm, a random-based process is applied. This process begins from an empty *e*-tuple, and then develops a number of *e* iterations. In each iteration *h* (h=1,…,*e*), the process considers the events that have not been included in the *e*-tuple yet, and randomly selects one of these events. The selected event is placed in position *h* of the *e*-tuple, and so is considered as the *h*-th event in the sequential order represented by this *e*-tuple. Then, the current battery level $cbl_{s,sce(h)}$ of each smartphone *s* in the scenario corresponding to this *h*-th event is calculated as detailed in Section II. In this way, the random-based process defines the event to be placed in each position *h* of the *e*-tuple, and consequently, the sequential order indicated by the *e*-tuple to develop the *e* events one at a time.

The solution presented in Fig. 2.a corresponds to a feasible encoded solution for the problem instance shown in Fig. 1. In this case, the encoded solution is represented as a 6-tuple, since the number *e* of events is equal to six. Each position *h* of this 6-tuple contains a different event from the 6 events, and Fig. 2.b presents the current battery level $cbl_{s,sce(h)}$ defined for each smartphone *s* in the scenario of the event placed in each position *h* of this 6-tuple. Then, this 6-tuple indicates a feasible sequential order for developing the six events one at a time.

3) FITNESS EVALUATION PROCESS

This process is used with the aim of evaluating each one of the encoded solutions in the population of the algorithm regarding the considered optimization objective. In this case, the optimization objective is minimizing the time required to sequentially develop the *e* events one at a time, as described in Section II. Thus, for evaluating each one of the encoded solutions according to this objective, the process follows the behavior that is described below.

Considering a given encoded solution $<o_1, o_2, \ldots, o_e>$, where each o_h represents a different event from the *e* events, the process estimates the time required to sequentially develop the *e* events one at a time, following the sequential order indicated by this solution. To do this, the process develops a number of *e* iterations. In each iteration *h*, the process considers the event o_h (including the current battery level $cbl_{s,sce(h)}$ defined for each smartphone *s* in the scenario inherent to this event), and estimates the time required to develop the event o_h. Recall that this time refers to the time required to prepare the scenario inherent to the event o_h. To estimate this time, the process applies the scenario preparation approach proposed in [1] on the scenario inherent to the event o_h. This approach considers the current and target battery levels of the *m* smartphones in the mentioned scenario, and determines the sequences of battery charge/discharge actions that should be applied on the *m* smartphones, to reach the target battery levels of these *m* smartphones from the corresponding current battery levels, at the minimal possible time for the set of *m* smartphones. Then, this approach provides the process: a) the sequences of battery charge/discharge actions that should be applied on the *m* smartphones in the scenario inherent to o_h, and b) the time estimated to apply these sequences of actions, which defines the time estimated to prepare the scenario inherent to o_h.

Once the *e* iterations are complete, the process adds the estimated time for the *e* events, and thus obtains the estimated time to sequentially develop the *e* events one at a time, following the order indicated by the solution. Finally, the process assigns this estimated time as the fitness value to the encoded solution. By applying this process, the lower the time estimated for a solution, the better the solution regarding the considered optimization objective, and so the lower (i.e., better) the fitness value assigned to this solution.

Note that Fig. 2.c considers the event placed in each position *h* (h=1,…,6) of the solution presented in Fig. 2.a, and then details the time required to develop the event placed in each position *h*, as estimated by the scenario preparation approach proposed in [1]. Finally, Fig. 2.c details the time (253.09 minutes) required to sequentially develop the six events one at a time, following the order indicated by the solution. This time has been obtained by adding the time

estimated for the six events, which is the fitness value assigned to the mentioned solution.

4) CROSSOVER PROCESS

The genetic algorithm utilizes a crossover process for creating a pool of new encoded solutions from the encoded solutions that make up the mating pool. Regarding the mating pool, the genetic algorithm applies a known parent selection process named *tournament selection* [3], to determine which encoded solutions from the current population will make up the mating pool. Then, the genetic algorithm organizes the encoded solutions in the mating pool into pairs, and applies the crossover process on each pair of encoded solutions, under a probability *Pc*, to create new encoded solutions. In this sense, we designed a feasible crossover process for the solution encoding detailed in Section III.B.2. Below, the behavior of this crossover process is described in detail.

Considering two given encoded solutions named $p1$ and $p2$, the crossover process creates two new encoded solutions named $o1$ and $o2$. To achieve this, the process follows two stages. In the first stage, the process applies a known crossover operator named LOX (Linear Order Crossover operator) [3] on $p1$ and $p2$. The operator LOX is applied here because of this operator has been satisfactorily applied in the literature to develop the crossover of solutions encoded as permutations of n cities for the TSP [3]. To apply LOX on $p1$ and $p2$, the process develops the next steps. First, the process randomly selects two positions $c1$ and $c2$ from the e positions of the solution $p1$ ($c1, c2 \in \{1,\ldots, e\}$ and $c1 < c2$). After that, to generate the new encoded solution $o1$ ($o2$), the process copies the events placed in positions [$c1$, $c2$] of $p1$ ($p2$) to the positions [$c1$, $c2$] of the solution $o1$ ($o2$), in the same order. Finally, the process considers the events that have not been included in the solution $o1$ ($o2$) yet, and places these events in the empty positions of the solution $o1$ ($o2$), following the order in which these events appear in $p2$ ($p1$). In the second stage, the process goes through the e positions of the solution $o1$ ($o2$). For each position h of $o1$ ($o2$), the process considers the event placed in this position, and then calculates the current battery level $cbl_{s,sce(h)}$ of each smartphone s in the scenario inherent to this event, as detailed in Section II. Once the second stage is complete, the crossover process provides the new encoded solutions $o1$ and $o2$ as result. Each of these solutions represents a feasible sequential order to develop the e events one at a time.

Fig. 4 shows in detail an example of the crossover process. In this example, the crossover process is applied on the encoded solutions $p1$ and $p2$, and then provides the encoded solutions $o1$ and $o2$. In this respect, the solutions $p1$ and $p2$ correspond to the solutions presented in Fig 2 and Fig. 3, respectively, for the problem instance detailed in Fig. 1. Then, the positions $c1$ and $c2$ randomly selected by the process are equal to 3 and 4, respectively. Thus, to create $o1$, the events placed in positions [3, 4] of $p1$ (i.e., the events 5 and 6) are copied to the positions [3, 4] of $o1$, in the same order. After that, the remaining events (i.e., the events 1, 2, 3 and 4) are copied in the empty positions of $o1$ (i.e., the positions 1, 2, 5 and 6), following the relative order in which these events appear in $p2$ (i.e., the relative order is 1, 3, 2 and 4). Similarly, to create $o2$, the events placed in positions [3, 4] of $p2$ (i.e., the events 5 and 2) are copied to the positions [3, 4] of $o2$, in the same order. Subsequently, the remaining events (i.e., the events 1, 3, 4 and 6) are copied in the empty positions of $o2$ (i.e., the positions 1, 2, 5, and 6), following the relative order in which these events appear in $p1$ (i.e., the relative order is 3, 4, 6 and 1). Finally, for each position h ($h = 1,\ldots, 6$) of $o1$ ($o2$), the event placed in this position is considered, and the current battery level $cbl_{s,sce(h)}$ of each of the 3 smartphones ($s = 1,\ldots,3$) in the scenario inherent to this event is calculated, as detailed in Section II, and considering the values $tbl_{s,sce(h-1)}$ and $d_{s,str(h-1)}$ detailed in Fig. 1.

	1st	2nd	3rd	4th	5th	6th
***p*1**	3	4	5	6	1	2
***p*2**	1	3	5	2	4	6
***o*1**	1	3	5	6	2	4

Smartphone	$cbl_{5,sce(1)}$	$cbl_{5,sce(2)}$	$cbl_{5,sce(3)}$	$cbl_{5,sce(4)}$	$cbl_{5,sce(5)}$	$cbl_{5,sce(6)}$
$s = 1$	65%	42%-10%=32%	90%-10%=80%	92%-10%=82%	92%-20%=72%	42%-20%=22%
$s = 2$	95%	28%-10%=18%	96%-10%=86%	51%-10%=41%	51%-18%=33%	28%-18%=10%
$s = 3$	49%	35%-17%=18%	34%-17%=17%	22%-17%=5%	22%-14%=8%	35%-14%=21%

***o*2**	3	4	5	2	6	1

Smartphone	$cbl_{5,sce(1)}$	$cbl_{5,sce(2)}$	$cbl_{5,sce(3)}$	$cbl_{5,sce(4)}$	$cbl_{5,sce(5)}$	$cbl_{5,sce(6)}$
$s = 1$	65%	90%-10%=80%	90%-20%=70%	92%-10%=82%	42%-20%=22%	92%-20%=72%
$s = 2$	95%	96%-10%=86%	96%-18%=78%	51%-10%=41%	28%-18%=10%	51%-18%=33%
$s = 3$	49%	34%-17%=17%	34%-14%=20%	22%-17%=5%	35%-14%=21%	22%-14%=8%

FIGURE 4. Example of the crossover process applied on encoded solutions for the problem instance shown in Fig. 1.

5) MUTATION PROCESS

The genetic algorithm applies a mutation process on each encoded solution provided by the crossover process, under a probability P_m, with the aim of incorporating diversity into the pool of newly created encoded solutions, and consequently, preserving the diversity of the population during the search process developed by the algorithm. In this respect, we designed a feasible mutation process for the solution encoding detailed in Section III.B.2.

Given an encoded solution $p1$, the mutation process provides a new encoded solution $o1$. To do that, the mutation process comprises two stages. In the first stage, the process applies a known mutation operator named IM (Inversion Mutation operator) [3] on $p1$. The operator IM is utilized here since the sequential application of the operators LOX and IM has been successfully evaluated in the literature for exploring solutions encoded as permutations of n cities for TSP [3]. To apply IM on $p1$, the process performs the following steps. First, the process randomly chooses two positions $c1$ and $c2$ from the e positions of the solution $p1$ ($c1$, $c2 \in \{1,\ldots, e\}$ and $c1 < c2$). Then, to generate the new encoded solution $o1$, the process copies the events placed in positions [$c1$, $c2$] of $p1$ to the positions [$c1$, $c2$] of the solution $o1$, in reverse order. To do this, the process develops n iterations ($n = c2 - c1 + 1$). In each iteration t ($t = 0,\ldots, n-1$), the process copies the event placed in position ($c1 + t$) of $p1$ to the position ($c2 - t$) of the solution $o1$. Finally, the process considers the events that have not been included in the solution $o1$ yet, and then copies these events to the solution $o1$, in the same positions that these events have in $p1$. In the second stage, the process goes through the e positions of the solution $o1$. For each position h of $o1$, the process considers the event placed in this position, and then calculates the current battery level $cbl_{s,sce(h)}$ of each smartphone s in the scenario inherent to this event, as detailed in Section II. After the second stage is finished, the mutation process supplies the new encoded solution $o1$ as the result. This new solution represents a feasible sequential order to develop the e events one at a time.

Fig. 5 shows an example of the mutation process. In this example, the process is applied on the encoded solution $p1$, and provides the encoded solution $o1$. Regarding this, the solution $p1$ corresponds to the solution presented in Fig. 2 for the problem instance detailed in Fig. 1. Then, the positions $c1$ and $c2$ randomly selected by the process are equal to 2 and 4, respectively. Therefore, to create the solution $o1$, the events placed in positions [2, 4] of $p1$ (i.e., the events 4, 5 and 6) are copied to the positions [2, 4] of $o1$ in the reverse order (i.e., the reverse order is 6, 5 and 4). After that, the remaining events (i.e., the events 1, 2 and 3) are copied to the solution $o1$, in the same positions that these events have in $p1$. In this sense, the positions of the events 1, 2 and 3 in the solution $p1$ are 5, 6 and 1, respectively. Thus, the events 1, 2 and 3 are copied to the positions 5, 6 and 1 of the solution $o1$. Finally, for each position h (h=1,…,6) of the solution $o1$, the event placed in this position is considered, and then the current battery level $cbl_{s,sce(h)}$ of each one of the 3 smartphones (s = 1,…,3) in the scenario inherent to this event is calculated, as detailed in Section II, and taking into account the values $tbl_{s,sce(h-1)}$ and $d_{s,str(h-1)}$ detailed in Figure 1.

	1st	2nd	3rd	4th	5th	6th
p1	3	4	5	6	1	2
o1	3	6	5	4	1	2

Smartphone	$cbl_{s,sce(1)}$	$cbl_{s,sce(2)}$	$cbl_{s,sce(3)}$	$cbl_{s,sce(4)}$	$cbl_{s,sce(5)}$	$cbl_{s,sce(6)}$
$s = 1$	65%	90%-10%=80%	92%-20%=72%	92%-10%=82%	90%-20%=70%	42%-10%=32%
$s = 2$	95%	96%-10%=86%	51%-18%=33%	51%-10%=41%	96%-18%=78%	28%-10%=18%
$s = 3$	49%	34%-17%=17%	22%-14%=8%	22%-17%=5%	34%-14%=20%	35%-17%=18%

FIGURE 5. **Example of the mutation process applied on an encoded solution for the problem instance shown in Fig. 1.**

C. COMPLEXITY OF THE GENETIC ALGORITHM

We analyze below the computing time complexity, and also the spatial complexity, of the proposed genetic algorithm.

1) COMPUTING TIME COMPLEXITY

As detailed in Algorithm 1, the genetic algorithm starts by applying sequentially the two processes indicated in lines 1-2. Then, the algorithm develops a number of I iterations, as indicated in line 4. In each of these iterations, the algorithm applies sequentially the five processes indicated in lines 5-9. Finally, the algorithm applies the process indicated in line 12. Thus, to determine the computing time complexity of the algorithm, it is necessary first to determine the computing time complexity of each one of the processes applied by this algorithm, which is detailed below.

Generate initial population: This process creates p encoded solutions that are recorded in *pop*. To create each one of these solutions, the process iterates on the e positions of an empty e-tuple. For each position h, the process decides the event to be placed in the position, and after that it calculates the current battery level $cbl_{s,sce(h)}$ of each of the m smartphones in the scenario of this event. Thus, the computing time complexity of this process is $O(p*e*m)$.

Fitness evaluation: This process calculates the fitness value of each one of the p encoded solutions in *pop*. To calculate the fitness value of each solution, the process iterates on the e positions of the solution. For each of these e positions, the process considers the event located in this position, and then estimates the time required to develop this event, by applying the scenario preparation approach proposed in [1] on the scenario inherent to this event. As detailed in Section 3.B.II, this approach considers the m smartphones in the scenario, and the current and target battery levels of each smartphone, to determine the sequence of battery charge/discharge actions that should be applied on each smartphone. It is necessary to mention that the computing time complexity of this approach is as detailed in Eq. (7), where I' refers to the number of iterations developed by the approach (i.e., 2000), and p' is the number of solutions in the population used by the approach (i.e., 100). In this respect, given that each of these p' solutions encodes a possible sequence of actions for each smartphone, the term a refers to the total number of actions encoded in each solution p', and is defined as detailed in Eq. (8), where the term $|cbl(s) - tbl(s)|$ refers to the number of actions in the sequence encoded for the smartphone s. Considering all the previously mentioned, the computing time complexity of the fitness evaluation process is as detailed in Eq. (9).

$$O(I'*p'*a) \tag{7}$$

$$a = \sum_{s=1}^{m} |cbl(s) - tbl(s)| \tag{8}$$

$$O(p*e*I'*p'*a) \tag{9}$$

Tournament selection: This process selects p solutions from *pop* to make up the *mating pool*. To select each one of these p solutions, the process randomly chooses k solutions from *pop*, and after that iterates on these k solutions, in order to choose the solution with the best fitness value. Thus, the computing time complexity of this process is $O(p*k)$.

Crossover: This process is applied on each of the $p/2$ pairs of encoded solutions in the *mating pool*, under a probability Pc. Thus, a number of $(p/2)*P_c$ pairs of encoded solutions are crossed, and 2 new encoded solutions are obtained from each pair. As a result, a number of $(p/2)*P_c*2 = p*P_c$ new solutions are created by applying this process. To create each of these new solutions from a given pair of solutions, the process follows two stages. In the first stage, the process applies the operator LOX on this pair of solutions. This operator iterates on the e positions of the solutions in the pair, to determine the event to be placed in each of the e positions of the new solution. In the second stage, the process iterates on the e positions of the new solution. For each position h,

the process considers the event placed in this position, and then calculates the current battery level $cbl_{s,sce(h)}$ of each of the m smartphones in the scenario of this event. Thus, the computing time complexity of this process is $O(p*P_c*e*m)$.

It is necessary to mention that, as indicated in line 6 of Algorithm 1, the $p*P_c$ new solutions created by applying the crossover process are recorded in *offspring*. In addition, the $p*(1-P_c)$ solutions in the *mating pool* that were not crossed are also recorded in *offspring*. Thus, *offspring* contains $(p*P_c + p*(1-P_c)) = p$ solutions.

Mutation: This process is applied on each of the p encoded solutions in *offspring*, under a probability P_m. Therefore, a number of $p*P_m$ solutions in *offspring* are replaced with new solutions obtained by applying this process. To obtain each of these new solutions from a given solution, the process follows two stages. In the first stage, the process applies the operator IM on the given solution. This operator iterates on the e positions of this solution, to determine the event to be placed in each of the e positions of the new solution. In the second stage, the process iterates on the e positions of the new solution. For each position h, the process considers the event placed in this position, and then calculates the current battery level $cbl_{s,sce(h)}$ of each of the m smartphones in the scenario of this event. Therefore, the computing time complexity of this mutation process is $O(p*P_m*e*m)$.

Steady-state selection: This process selects p solutions from *pop* and *offspring*, in order to make up the population for the subsequent iteration. To achieve this, this process starts by ordering the p solutions in *pop*, according to the fitness values of these solutions, via the MergeSort method [21]. It is necessary to mention that the computing time complexity of this method is $O(p*log(p))$. After that, the process orders the p solutions in *offspring*, according to the fitness values of these solutions, via the mentioned method. Finally, the process considers the replacement percentage c, and replaces the $p*(c/100)$ worst solutions in *pop* (i.e., solutions with the worst fitness values) by the $p*(c/100)$ best solutions in *offspring* (i.e., solutions with the best fitness values). Thus, the computing time complexity of this process is $O(p*log(p))$.

Best solution: This process iterates on the p solutions in *pop*, in order to choose the solution with the best fitness value. Thus, the computing time complexity of this process is $O(p)$.

Besides determining the computing time complexity of the processes applied by the genetic algorithm, it is necessary to determine the number of times that each of these processes is applied by this algorithm. In this respect, as detailed in Algorithm 1, the processes indicated in lines 5-9 are applied in each one of the I iterations developed by this algorithm.

Thus, considering the previously detailed computing time complexity of the processes applied by the genetic algorithm, and also the number of times that each of these processes is applied by this algorithm, the computing time complexity TC of this algorithm is as detailed in Eq. (10). By the summation rule of the *Big-Oh* notation [21], and given that the 6th term on the right-side of Eq. (10) is the higher-complexity term, the computing time complexity TC of this algorithm can be simplified as detailed in Eq. (11).

$$\begin{aligned} TC = {} & O(p*e*m) + \\ & O(p*e*I'*p'*a) + \\ & O(I*p*k) + \\ & O(I*p*P_c*e*m) + \\ & O(I*p*P_m*e*m) + \\ & O(I*p*e*I'*p'*a) + \\ & O(I*p*log(p)) + \\ & O(p) \end{aligned} \tag{10}$$

which translates to:

$$TC = O(I*p*e*I'*p'*a) \tag{11}$$

2) SPATIAL COMPLEXITY

To determine the spatial complexity of the genetic algorithm, it is necessary to determine the spatial complexity of the data structures and variables that are created and then allocated by this algorithm in memory. To do this, because this algorithm was implemented in Java 1.8, it is required to consider the typical memory requirements for primitive types (e.g., integer and double variables), arrays, objects, and references to objects, in Java implementations. These typical memory requirements are considered here as indicated in [22, 31].

Data structures pop, mating_pool, and offspring: Each of these data structures contains p solutions, and each of these p solutions contains e events. For this reason, each of these data structures was implemented as a two-dimensional *p-by-e* array of *Event* objects. Besides, each *Event* object includes: *a*) the identifier of the event, which is an integer variable on the range [1,…, e], and *b*) the current battery levels $cbl_{s,sce(h)}$ defined for the m smartphones in the scenario of the event, which are integer variables on the range [0,…, 100].

In order to define the spatial complexity inherent to each one of these data structures, it is necessary to consider that the two-dimensional *p-by-e* array contains $p*e$ references to *Event* objects, and each one of these references uses 8 bytes. Moreover, each *Event* object uses 20 bytes plus $(1 + m)*4$ bytes: 16 bytes of overhead plus 4 bytes of padding plus 4 bytes for each of its $(1 + m)$ integer variables. Therefore, the spatial complexity of the three data structures (in bytes) is $3*(p*e*(8 + 20 + (1+m)*4)) = 96*p*e + 12*p*e*m$.

In the algorithm, each one of the three data structures has associated an array of p double values, to record the fitness value of each one of the p solutions in the structure. This array uses 8 bytes for each of its p double values. Therefore, the spatial complexity of the fitness value arrays associated to the three data structures (in bytes) is $3*(p*8) = 24*p$.

Data structure solution: This data structure is used to record the best solution obtained by the genetic algorithm. This solution contains e events. For this reason, this data structure was implemented as an array of e *Event* objects. Thus, the spatial complexity of this data structure (in bytes) is equal to $e*(8 + 20 + (1+m)*4) = 32*e + 4*e*m$.

Data structures inherent to the fitness evaluation process: Due to the fitness evaluation process of the genetic algorithm applies the scenario preparation approach proposed in [1], it is necessary to determine the spatial complexity of the data structures created and then used by this approach in memory. In this sense, as detailed in [1], this approach uses three main data structures. Each of these structures contains p' solutions (i.e., 100), and each one of these solutions contains a actions, considering a as detailed in Eq. (8). Then, each one of these data structures was implemented as a two-dimensional p'-by-a array of *Action* objects. In this respect, each *Action* object includes: a) the kind of action (a boolean variable), b) the initial battery level (an integer variable), c) the end battery level (an integer variable), d) CPU load (an integer variable), and e) screen state (a boolean variable).

For defining the spatial complexity of each of these data structures, it is necessary to note that the two-dimensional p'-by-a array contains $p'*a$ references to *Action* objects, and each of these references uses 8 bytes. In addition, each *Action* object utilizes 34 bytes: 16 bytes of overhead plus 4 bytes of padding plus 4 bytes for each of its 3 integer variables plus 1 byte for one each of its 2 boolean variables. Thus, the spatial complexity of the three data structures (in bytes) is equal to $3*(p'*a*(8 + 34)) = 126*p'*a$. Moreover, each of these data structures has associated an array of p' double values, which allows to record the fitness value of each of the p' solutions in the structure. Thus, the spatial complexity of the arrays associated to the three data structures (in bytes) is $3*(p'*8) = 24*p'$.

Therefore, considering the previously mentioned spatial complexity of the data structures created and then used by the genetic algorithm, and also the spatial complexity of the data structures created and then used by the approach proposed in [1], the spatial complexity SC of this algorithm (in bytes) is as detailed in Eq. (12).

$$SC = 96 * p * e + 12 * p * e * m + 24 * p + 32 * e + 4 * e * m + 126 * p' * a + 24 * p' \quad (12)$$

IV. COMPUTATIONAL EXPERIMENTS

As mentioned in Section III, BAGESS uses the previously presented genetic algorithm to determine the sequential order in which a given number of e scenario preparation events should be developed, in such a way that the time required to develop these e events one at a time is minimized. Thus, computational experiments were developed with the aim of evaluating the genetic algorithm's performance on different instances of the addressed scenario preparation event sequencing problem.

In Section IV.A, the instance sets used to carry out these experiments are presented. In Section IV.B, the experimental setting considered for these experiments is detailed. In Section IV.C, the two alternative methods for the problem are described, for comparison purposes. Finally, in Section IV.D, the results obtained by these experiments are both presented and analyzed in detail.

A. INSTANCE SETS

In order to utilize different representative and realistic experimental instances of the addressed problem, 54 sets of instances of this problem were defined. Each one of these instance sets contains 10 different instances, where each instance includes a number of scenario preparation events. The 54 instance sets differ in terms of the category of their instances with respect to the five components that are described below.

Component R: The number r of scenarios considered in the instance, where $r \in \{10, 15, 20\}$. Thus, 3 different categories of instances are considered in respect of the number of scenarios, namely R10, R15 and R20.

Component G: The number g of load-balancing strategies taken into account in the instance, where $g \in \{2, 4, 6\}$. Therefore, 3 different categories of instances are considered regarding the number of strategies, namely G2, G4 and G6.

Component M: The number m of smartphones that compose the r scenarios of the instance. Given that $m \in \{4, 8, 16\}$, 3 different categories of instances are considered with respect to the number of smartphones, namely M4, M8 and M16. It is necessary to mention that each one of the r scenarios includes target battery levels predetermined for the m smartphones. Besides, these r scenarios differ in terms of the target battery levels of the m smartphones.

Component D: The estimated battery percentage decreases that can be suffered by the m smartphones composing the scenarios considered in the instance, if the

strategy considered in the instance is evaluated on the set of *m* smartphones. These decreases are integer values on the range [0, 20]%. This value range has been divided into two disjoint subranges, with the aim of considering two different categories of instances regarding *D*. Specifically, this range has been divided into the next two subranges: [0, 10]% and (10, 20]%, in order to determine the categories Low D (LD) and High D (HD) with the same width, respectively.

Component E: The number $e = r * g$ of events considered in the instance. Given that $r \in \{10, 15, 20\}$, and $g \in \{2, 4, 6\}$, the number $e \in \{20, 30, 40, 60, 80, 90, 120\}$. Therefore, seven different categories of instances are considered in relation to the component *E*.

Tables I, II and III describe in detail the 54 defined instance sets, regarding the five components above-mentioned. In these tables, column 1 presents the name of each instance set, considering that this name is composed by the category of the instances with respect to the components *R*, *G*, *M*, and *D*. Then, columns 2, 3 and 4 indicate the value of the instances of each set in relation to the components *R*, *G* and *M*, respectively. After that, column 5 indicates the value range of the instances of each set in relation to the component *D*. Subsequently, column 6 indicates the value of the instances of each set in relation to the component *E*. Last, column 7 details the number of instances that compose each set.

TABLE I
CHARACTERISTICS OF THE DEFINED INSTANCE SETS WHERE G = 2.

Instance set	*R*	*G*	*M*	*D*	*E*	No. of instances
G2_R10_M4_HD	10	2	4	(10, 20]	20	10
G2_R10_M4_LD	10	2	4	[0, 10]	20	10
G2_R10_M8_HD	10	2	8	(10, 20]	20	10
G2_R10_M8_LD	10	2	8	[0, 10]	20	10
G2_R10_M16_HD	10	2	16	(10, 20]	20	10
G2_R10_M16_LD	10	2	16	[0, 10]	20	10
G2_R15_M4_HD	15	2	4	(10, 20]	30	10
G2_R15_M4_LD	15	2	4	[0, 10]	30	10
G2_R15_M8_HD	15	2	8	(10, 20]	30	10
G2_R15_M8_LD	15	2	8	[0, 10]	30	10
G2_R15_M16_HD	15	2	16	(10, 20]	30	10
G2_R15_M16_LD	15	2	16	[0, 10]	30	10
G2_R20_M4_HD	20	2	4	(10, 20]	40	10
G2_R20_M4_LD	20	2	4	[0, 10]	40	10
G2_R20_M8_HD	20	2	8	(10, 20]	40	10
G2_R20_M8_LD	20	2	8	[0, 10]	40	10
G2_R20_M16_HD	20	2	16	(10, 20]	40	10
G2_R20_M16_LD	20	2	16	[0, 10]	40	10

TABLE II
CHARACTERISTICS OF THE DEFINED INSTANCE SETS WHERE G = 4.

Instance set	*R*	*G*	*M*	*D*	*E*	No. of instances
G4_R10_M4_HD	10	4	4	(10, 20]	40	10
G4_R10_M4_LD	10	4	4	[0, 10]	40	10
G4_R10_M8_HD	10	4	8	(10, 20]	40	10
G4_R10_M8_LD	10	4	8	[0, 10]	40	10
G4_R10_M16_HD	10	4	16	(10, 20]	40	10
G4_R10_M16_LD	10	4	16	[0, 10]	40	10
G4_R15_M4_HD	15	4	4	(10, 20]	60	10
G4_R15_M4_LD	15	4	4	[0, 10]	60	10
G4_R15_M8_HD	15	4	8	(10, 20]	60	10
G4_R15_M8_LD	15	4	8	[0, 10]	60	10
G4_R15_M16_HD	15	4	16	(10, 20]	60	10
G4_R15_M16_LD	15	4	16	[0, 10]	60	10
G4_R20_M4_HD	20	4	4	(10, 20]	80	10
G4_R20_M4_LD	20	4	4	[0, 10]	80	10
G4_R20_M8_HD	20	4	8	(10, 20]	80	10
G4_R20_M8_LD	20	4	8	[0, 10]	80	10
G4_R20_M16_HD	20	4	16	(10, 20]	80	10
G4_R20_M16_LD	20	4	16	[0, 10]	80	10

TABLE III
CHARACTERISTICS OF THE DEFINED INSTANCE SETS WHERE G = 6.

Instance set	*R*	*G*	*M*	*D*	*E*	No. of instances
G6_R10_M4_HD	10	6	4	(10, 20]	60	10
G6_R10_M4_LD	10	6	4	[0, 10]	60	10
G6_R10_M8_HD	10	6	8	(10, 20]	60	10
G6_R10_M8_LD	10	6	8	[0, 10]	60	10
G6_R10_M16_HD	10	6	16	(10, 20]	60	10
G6_R10_M16_LD	10	6	16	[0, 10]	60	10
G6_R15_M4_HD	15	6	4	(10, 20]	90	10
G6_R15_M4_LD	15	6	4	[0, 10]	90	10
G6_R15_M8_HD	15	6	8	(10, 20]	90	10
G6_R15_M8_LD	15	6	8	[0, 10]	90	10
G6_R15_M16_HD	15	6	16	(10, 20]	90	10
G6_R15_M16_LD	15	6	16	[0, 10]	90	10
G6_R20_M4_HD	20	6	4	(10, 20]	120	10
G6_R20_M4_LD	20	6	4	[0, 10]	120	10
G6_R20_M8_HD	20	6	8	(10, 20]	120	10
G6_R20_M8_LD	20	6	8	[0, 10]	120	10
G6_R20_M16_HD	20	6	16	(10, 20]	120	10
G6_R20_M16_LD	20	6	16	[0, 10]	120	10

B. EXPERIMENTAL SETTING

The genetic algorithm was run on each of the 10 instances of each of the 54 instance sets detailed in Section IV.A. Given that the genetic algorithms are not deterministic algorithms [3], this genetic algorithm was run several times on each instance (i.e., 30 runs), with the aim of achieving reliable statistical results. For each one of the runs, the solution reached by the genetic algorithm for the instance was recorded. The fitness value of this solution (i.e., the time (in minutes) required to develop the *e* events considered in the instance one at a time, according to the sequential order indicated by the solution) was also recorded. Besides, the runtime required by the genetic algorithm to obtain this solution for the instance was recorded.

To develop the runs of the genetic algorithm, the parameter setting indicated in Table IV was used. It is necessary to mention that preliminary experiments were developed in order to select this parameter setting. In such experiments, diverse parameter settings typically recommended in the literature on genetic algorithms [3, 9] were considered, which are detailed in Table V. For each one of these parameter settings, the genetic algorithm was run several times (i.e., 30 runs) on each instance, and then the average fitness value of the 30 solutions achieved for each instance was calculated. According to these experiments, the parameter setting indicated in Table IV provided the best average fitness values for the instances used.

TABLE IV
PARAMETER SETTING OF THE GENETIC ALGORITHM.

Parameter	Value
Population size	50
k (*Tournament selection*)	3
P_c (*Crossover*)	1.0
P_m (*Mutation*)	0.1
c (*Steady-state selection*)	50%
Number of generations or iterations	200

TABLE V
PARAMETER SETTINGS CONSIDERED IN THE PRELIMINARY EXPERIMENTS.

Parameter	Considered values
Population size	{50, 100}
k (*Tournament selection*)	{3, 4, 5}
P_c (*Crossover*)	{0.7, 0.8, 0.9, 1.0}
P_m (*Mutation*)	{0.05, 0.1}
c (*Steady-state selection*)	{25%, 50%}
Number of generations or iterations	{200, 400, 800, 1000}

C. CURRENT METHODS FOR SEQUENCING LOAD-BALANCING SCENARIOS

To comparatively evaluate the genetic algorithm's performance, the two methods currently utilized for determining the sequential order to develop a given number e of scenario preparation events one at a time are considered. In this respect, one of these methods is named Sequential Order by Scenario (SO-S), and the other method is named Sequential Order by Load-Balancing Strategy (SO-LBS). These two methods are described below in detail.

1) SEQUENTIAL ORDER BY SCENARIO (SO-S)

Given a number $e = r * g$ of scenario preparation events, this method sequentially orders the e events according to the scenario included in these events. Specifically, the method considers the number r of scenarios, and then develops a number r of iterations. In each iteration i (i = 1,…, r), the method considers the g events that include the scenario i, and differ in terms of the considered strategy. Then, the method adds these g events to the end of the sequential order (i.e., places these events in the positions [(g * (i - 1)) + 1, …, (g * (i - 1)) + g] of the sequential order). Once the e events have been added to the sequential order, the method goes through the e positions of this sequential order. For each position h (h = 1,…, e), the method considers the event placed in this position, and calculates the current battery level $cbl_{s,sce(h)}$ of each smartphone s in the scenario inherent to this event, as detailed in Section II. Thus, SO-S obtains and provides a feasible order to develop the e events one at a time.

Fig. 6.a shows the sequential order provided by SO-S to develop the 6 events detailed in the problem instance shown in Fig. 1, and then Fig. 6.b indicates the scenario $sce(h)$ of the event placed in each position h of this order. In this case, the events that include scenario 1 (i.e, events 1 and 2) have been placed in positions [1, 2] of the sequential order. After that, the events that include scenario 2 (i.e, events 3 and 4) have been placed in positions [3, 4] of the sequential order. Last, the events that include scenario 3 (i.e, events 5 and 6) have been placed in positions [5, 6] of the sequential order. Fig. 6.c presents the current battery level $cbl_{s,sce(h)}$ defined for each one of the 3 smartphones (s = 1,…,3) in the scenario $sce(h)$ of the event placed in each position h of this sequential order.

Regarding the computing time complexity of this method, it is necessary to note that this method follows two stages. In the first stage, this method iterates on the r scenarios. For each scenario, this method iterates on the e events, to identify the events that include the scenario, and add these events to the end of the sequential order. Thus, the computing time complexity of this first stage is $O(r*e)$. In the second stage, this method iterates on the e events in the sequential order. For each event, this method calculates the current battery levels $cbl_{s,sce(h)}$ of the m smartphones in the scenario inherent to the event. Thus, the computing time complexity of this second stage is equal to $O(e*m)$. Considering the computing time complexity of the two mentioned stages, and by the summation rule of the Big-Oh notation [21], the computing time complexity of this method is $max(O(r*e), O(e*m))$.

In relation to the spatial complexity of this method, it is necessary to mention that this method creates and then uses one data structure to contain the e events in the sequential order defined for them. This structure was implemented as an array of e *Event* objects. Therefore, the spatial complexity of this method (in bytes) is equal to $e*(8 + 20 + (1+m)*4) = 32*e + 4*e*m$.

a)

	1st	2nd	3rd	4th	5th	6th
Event	1	2	3	4	5	6

b)

sce(h)	1	1	2	2	3	3

c)

Smartphone	$cbl_{s,sce(1)}$	$cbl_{s,sce(2)}$	$cbl_{s,sce(3)}$	$cbl_{s,sce(4)}$	$cbl_{s,sce(5)}$	$cbl_{s,sce(6)}$
s = 1	65%	42%-10%=32%	42%-20%=22%	90%-10%=80%	90%-20%=70%	92%-10%=82%
s = 2	95%	28%-10%=18%	28%-18%=10%	96%-10%=86%	96%-18%=78%	51%-10%=41%
s = 3	49%	35%-17%=18%	35%-14%=21%	34%-17%=17%	34%-14%=20%	22%-17%=5%

FIGURE 6. Solution provided by the method SO-S for the problem instance shown in Fig. 1.

2) SEQUENTIAL ORDER BY LOAD-BALANCING STRATEGY (SO-LBS)

Considering a given number $e = r * g$ of scenario preparation events, this method sequentially orders the e events according to the strategy considered in these events. In particular, the method considers the number g of strategies, and develops a number g of iterations. In each iteration j (j = 1,…, g), the method considers the r events that include the strategy j, and are different with respect to the considered scenario. Next, these r events are added to the end of the

sequential order (i.e., these events are placed in the positions $[(r * (j - 1)) + 1,\ldots, (r * (j - 1)) + r]$ of the sequential order). After the e events have been incorporated to the sequential order, the method goes through the e positions of this sequential order. For each one of the positions h ($h = 1,\ldots, e$), the method considers the event placed in this position, and then calculates the current battery level $cbl_{s,sce(h)}$ of each smartphone s in the scenario inherent to this event, as detailed in Section II. In this way, this method SO-LBS obtains and provides a feasible order to develop the e events one at a time.

Fig. 7.a shows the sequential order provided by SO-LBS to develop the 6 events detailed in the problem instance shown in Fig. 1, and next Fig. 7.b indicates the strategy $str(h)$ considered by the event placed in each position h of this sequential order. In this sequential order, the events that consider strategy 1 (i.e, events 1, 3 and 5) have been placed in positions [1, 3]. Subsequently, the events that consider strategy 2 (i.e, the events 2, 4 and 6) have been placed in positions [4, 6]. Fig. 7.c presents the current battery level $cbl_{s,sce(h)}$ defined for each one of the 3 smartphones ($s = 1,\ldots,3$) in the scenario $sce(h)$ of the event placed in each position h of this sequential order.

With respect to the computing time complexity of this method, this method follows two stages. In the first of these stages, this method iterates on the g strategies. For each strategy, it iterates on the e events, to identify the events that include the strategy, and add these events to the end of the sequential order. Therefore, the computing time complexity of this first stage is $O(g*e)$. In the second of these stages, this method iterates on the e events in the sequential order. For each event, this method calculates the current battery level $cbl_{s,sce(h)}$ of each of the m smartphones in the scenario inherent to the event. Therefore, the computing time complexity of this second stage is equal to $O(e*m)$. Taking into consideration the computing time complexity of the mentioned stages, and by the summation rule of the Big-Oh notation [21], it is possible to say that the computing time complexity of this method is equal to $max(O(g*e), O(e*m))$.

Regarding the spatial complexity inherent to this method, it is necessary to consider that this method creates and next utilizes one data structure for containing the e events in the sequential order defined for them, and such structure was implemented as an array of e *Event* objects. Thus, the spatial complexity of this method (in bytes) is $e*(8 + 20 + (1+m)*4) = 32*e + 4*e*m$.

3) EXPERIMENTAL EVALUATION OF THE METHODS

The previously described methods SO-S and SO-LBS allow to quickly define a feasible sequential order to develop the e events one at a time. However, these methods do not consider the time required to develop the e events one at a time. Unlike this, the genetic algorithm used by BAGESS has been specially designed to minimize the time required to develop the e events one at a time.

The methods SO-S and SO-LBS were applied on each of the ten instances of each instance set. For each one of the instances, the solution provided by SO-S, and also the solution provided by SO-LBS, were recorded. Then, the fitness values of these solutions were calculated as described in Section III.B.3, with the aim of comparing these fitness values with those of the 30 solutions provided by the genetic algorithm for the same instance.

a)

	1st	2nd	3rd	4th	5th	6th
Event	1	3	5	2	4	6

b)

str(h)	1	1	1	2	2	2

c)

Smartphone	$cbl_{s,sce(1)}$	$cbl_{s,sce(2)}$	$cbl_{s,sce(3)}$	$cbl_{s,sce(4)}$	$cbl_{s,sce(5)}$	$cbl_{s,sce(6)}$
$s = 1$	65%	42%-10%=32%	90%-10%=80%	92%-10%=82%	42%-20%=22%	90%-20%=70%
$s = 2$	95%	28%-10%=18%	96%-10%=86%	51%-10%=41%	28%-18%=10%	96%-18%=78%
$s = 3$	49%	35%-17%=18%	34%-17%=17%	22%-17%=5%	35%-14%=21%	34%-14%=20%

FIGURE 7. Solution provided by the method SO-LBS for the problem instance shown in Fig. 1.

D. EXPERIMENTAL RESULTS

In this section, we analyze the solutions obtained by the genetic algorithm, and the methods SO-S and SO-LBS, for the instance sets presented in Section IV.A. After that, we analyze the runtime required by the genetic algorithm, and also the mentioned methods, to provide the solutions for the instance sets.

1) ANALYSIS OF OBTAINED SOLUTIONS

Tables VI-VIII present the main results obtained by the computational experiments developed. In these tables, Column 1 details the name of each instance set used in these experiments. Then, Columns 2-4 detail the average fitness value of the solutions obtained by the genetic algorithm, the method SO-S and the method SO-LBS for the instances of each set, respectively. Recall that the fitness value of a solution refers to the time required to develop the *e* events considered in the instance one at a time, according to the sequential order indicated by the solution. Next, Columns 5-7 detail the minimum fitness value of the solutions obtained by the genetic algorithm, SO-S and SO-LBS for the instances of each set, respectively. Finally, Columns 8-10 present the maximum fitness value of the solutions obtained by the genetic algorithm, SO-S and SO-LBS for the instances of each set, respectively.

The average fitness value of the solutions obtained by the genetic algorithm for each one of the instance sets is much lower than that of the solutions obtained by SO-S, and also considerably lower than that of the solutions obtained by SO-LBS. Specifically, for all instance sets (i.e., 54 instance sets), the difference between the average fitness values reached by the genetic algorithm and SO-S is in the range of [339, 1334] minutes. Besides, for the instance sets where G=2, the difference between the average fitness values reached by the genetic algorithm and SO-LBS is in the range of [1331, 4545] minutes. Then, for the instances sets where G=4, and the instances sets where G=6, the difference between the average fitness values obtained by the genetic algorithm and SO-LBS is on the range [3241, 11942] minutes, and on the range of [5380, 19474] minutes, respectively.

These results obtained respecting the average fitness value are mainly due to the next reasons. For most instance sets (i.e., 50 of the 54 sets), the maximum fitness value reached by the genetic algorithm is lower than the minimum fitness value reached by SO-S, and for all instance sets (i.e., 54 instance sets), the maximum fitness value reached by the genetic algorithm is lower than the minimum fitness value reached by SO-LBS. Moreover, for each of the 540 instances utilized in these experiments, the fitness values of the 30 solutions reached by the genetic algorithm are significantly lower than the fitness value of the solution provided by SO-S and the fitness value of the solution provided by SO-LBS. The statistical significance of these results was validated by the Mann–Whitney U test [4], considering a confidence level of $\alpha = 0.001$.

Based on the previously mentioned results, the solutions given by the genetic algorithm reduce the time required for sequentially developing the *e* events considered in the instances used. Tables IX-XI detail the average, minimum and maximum RPD (Relative Percentage Difference) value of the solutions provided by the genetic algorithm, regarding the solutions provided by SO-S and SO-LBS, in terms of the time required to sequentially develop the *e* events of the instances. The metric RPD determines the percentage difference of the time required to sequentially develop the *e* events according to o_{EA} (i.e., solution provided by the genetic algorithm), regarding the time required to sequentially develop the *e* events according to $o_{SO\text{-}S}$ (i.e., the solution provided by SO-S) / $o_{SO\text{-}LBS}$ (i.e., solution provided by SO-LBS). This metric is calculated as detailed in Eqs. (13-14), where the terms $T(o_{EA})$, $T(o_{SO\text{-}S})$ and $T(o_{SO\text{-}LBS})$ refer to the fitness value of the solutions o_{EA}, $o_{SO\text{-}S}$ and $o_{SO\text{-}LBS}$, respectively. Note that if the RPD value is a positive value, this means that o_{EA} provides a saving regarding $o_{SO\text{-}S}$ / $o_{SO\text{-}LBS}$, in terms of the time required to sequentially develop the *e* events one at a time.

$$RPD = \frac{T(o_{SO-S}) - T(o_{EA})}{T(o_{SO-S})} \times 100 \tag{13}$$

$$RPD = \frac{T(o_{SO-LBS}) - T(o_{EA})}{T(o_{SO-LBS})} \times 100 \tag{14}$$

From the results detailed in Tables IX-XI, it can be mentioned that the solutions given by the genetic algorithm for the instances of each set provide a significant average saving (average RPD), in terms of the time (in minutes) required to sequentially develop the *e* events one at a time. Specifically, for all instance sets, the average saving of the solutions provided by the genetic algorithm regarding the solutions given by SO-S is on the range [12, 43]%. Besides, for the instance sets where $G = 2$, the average saving of the solutions given by the genetic algorithm in respect of the solutions given by SO-LBS is in the range [51, 69]%. Then, for the instance sets where $G = 4$ or $G = 6$, the average saving of the solutions provided by the genetic algorithm regarding the solutions given by SO-LBS is in the range [66, 85]%.

Moreover, the solutions given by the genetic algorithm for the instances of each set also provide a very good minimal saving (minimal RPD). Specifically, for all instance sets, the minimal saving of the solutions reached by the genetic algorithm regarding the solutions reached by SO-S is in the range [8, 37]%. In addition, for the instance sets where $G = 2$, the minimal saving of the solutions given by the genetic algorithm with respect to the solutions given by SO-LBS is in the range [45, 65]%. Then, for the instance sets where $G = 4$ or $G = 6$, the minimal saving of the solutions provided by the genetic algorithm respecting the solutions given by SO-LBS is in the range [62, 85]%. These results obtained in relation to the minimal saving indicate that, for each of the 540 instances utilized in the computational experiments, the 30 solutions reached by the genetic algorithm give a saving in terms of the time (in minutes) required to sequentially develop the *e* events of the instance.

TABLE VI
AVERAGE, MINIMUM AND MAXIMUM TIME REQUIRED FOR SEQUENTIALLY DEVELOPING THE e EVENTS ACCORDING TO THE SOLUTIONS PROVIDED BY THE GENETIC ALGORITHM (GA), THE METHOD SO-S AND THE METHOD SO-LBS, FOR EACH INSTANCE SET WHERE $G = 2$.

Instance set	Time required to develop the e events (minutes)								
	Average			Minimum			Maximum		
	GA	SO-S	SO-LBS	GA	SO-S	SO-LBS	GA	SO-S	SO-LBS
G2_R10_M4_HD	**930.30**	1329.65	2261.31	796.03	1102.49	1833.84	**1098.90	1534.53	2612.05
G2_R10_M4_LD	**866.46**	1435.22	2700.51	755.68	1206.33	2289.56	**1008.46	1699.68	3163.44
G2_R10_M8_HD	**1325.82**	1717.52	2927.99	1266.61	1627.39	2745.22	**1444.24	1848.01	3139.36
G2_R10_M8_LD	**1263.33**	1809.38	3415.68	1182.40	1411.95	2562.66	**1332.85	1989.64	3830.77
G2_R10_M16_HD	**1702.90**	2089.49	3557.81	1645.36	1964.14	3380.16	**1852.48	2217.80	3766.01
G2_R10_M16_LD	**1695.96**	2167.05	4102.89	1566.73	2042.91	3910.85	**1849.59	2252.59	4266.75
G2_R15_M4_HD	**1268.73**	2072.13	3525.80	1143.22	1888.53	3058.40	**1451.04	2355.23	4118.53
G2_R15_M4_LD	**1195.15**	2076.57	3949.93	1043.81	1809.47	3410.70	**1310.15	2392.38	4600.78
G2_R15_M8_HD	**1765.82**	2551.90	4336.78	1567.10	2075.85	3602.75	**1983.05	2945.04	5047.98
G2_R15_M8_LD	**1870.44**	2631.78	4990.14	1777.24	2362.28	4574.16	**1991.24	2787.48	5300.03
G2_R15_M16_HD	**2454.28**	3016.26	5092.13	2294.29	2834.14	4728.41	**2659.66	3195.48	5421.08
G2_R15_M16_LD	**2497.60**	3241.30	6029.99	2397.73	3157.87	5793.18	**2583.15	3506.28	6405.28
G2_R20_M4_HD	**1709.05**	2696.33	4527.65	1513.28	2499.71	4174.86	**1863.01	2939.42	5064.79
G2_R20_M4_LD	**1569.01**	2775.66	5230.19	1209.62	2427.03	4608.40	**1804.25	3091.53	5754.86
G2_R20_M8_HD	**2577.52**	3368.57	5711.80	2380.10	3000.95	5095.59	**2753.04	3620.62	6081.51
G2_R20_M8_LD	**2529.35**	3699.94	6941.82	2310.23	3481.21	6551.10	**2784.65	4025.86	7652.56
G2_R20_M16_HD	**3312.70**	4063.24	6877.36	3170.70	3921.88	6629.93	**3395.38	4228.44	7223.67
G2_R20_M16_LD	**3390.14**	4249.21	7935.31	3331.07	3946.24	7422.69	**3495.45	4632.10	8581.98

Bold values indicate better average times.
Symbol ** indicates that the maximum time reached by GA is lower than the minimum times reached by SO-S and SO-LBS.

TABLE VII
AVERAGE, MINIMUM AND MAXIMUM TIME REQUIRED FOR SEQUENTIALLY DEVELOPING THE e EVENTS ACCORDING TO THE SOLUTIONS PROVIDED BY THE GENETIC ALGORITHM (GA), THE METHOD SO-S AND THE METHOD SO-LBS, FOR EACH INSTANCE SET WHERE $G = 4$.

Instance set	Time required to develop the e events (minutes)								
	Average			Minimum			Maximum		
	GA	SO-S	SO-LBS	GA	SO-S	SO-LBS	GA	SO-S	SO-LBS
G4_R10_M4_HD	**1335.13**	1802.43	4577.00	1131.78	1544.78	3489.43	**1487.28	2133.59	5792.30
G4_R10_M4_LD	**1002.63**	1521.47	5083.38	761.69	1170.58	3942.30	*1217.25	1879.69	6639.34
G4_R10_M8_HD	**1758.28**	2119.33	5248.99	1532.11	1748.37	4162.49	*1939.83	2445.27	6168.78
G4_R10_M8_LD	**1582.58**	2012.54	6727.38	1469.21	1717.36	5566.47	*1731.83	2334.23	7582.57
G4_R10_M16_HD	**2251.47**	2704.29	7136.86	2143.32	2575.72	6685.55	**2302.95	2894.39	7768.29
G4_R10_M16_LD	**1891.84**	2348.27	7765.47	1651.49	2258.79	7454.27	**2013.20	2468.19	8203.43
G4_R15_M4_HD	**1872.46**	2518.63	6582.86	1652.39	2237.48	5699.96	**2134.76	2855.27	7911.61
G4_R15_M4_LD	**1485.02**	2319.76	7889.95	1257.44	2050.05	7085.94	**1617.95	2629.10	9376.58
G4_R15_M8_HD	**2763.21**	3538.04	9236.26	2572.64	3374.29	8370.11	**2902.65	3752.57	10326.97
G4_R15_M8_LD	**2264.68**	2938.40	9716.87	1990.72	2631.39	8371.27	**2475.78	3305.70	11094.18
G4_R15_M16_HD	**3389.09**	3939.84	10353.63	3304.74	3794.76	9951.33	**3704.15	4139.02	10763.10
G4_R15_M16_LD	**2975.24**	3560.77	11586.58	2807.28	3383.88	11077.08	**3174.08	3723.57	12381.66
G4_R20_M4_HD	**2413.02**	3613.63	9990.10	2113.15	3400.56	8946.00	**2858.97	3987.14	10765.49
G4_R20_M4_LD	**1986.86**	3320.52	11204.89	1848.64	3148.04	10600.76	**2061.31	3470.48	12085.19
G4_R20_M8_HD	**3494.24**	4574.84	12137.34	3283.86	4203.82	10986.94	**3561.04	4845.18	13128.52
G4_R20_M8_LD	**2865.20**	3960.11	13243.04	2566.86	3640.93	12281.18	**3245.16	4180.42	14083.94
G4_R20_M16_HD	**4397.33**	5169.37	13688.41	4209.70	4867.43	12424.00	**4513.92	5377.51	14715.77
G4_R20_M16_LD	**3982.95**	4799.84	15925.89	3912.36	4611.14	15279.66	**4116.48	4944.71	16890.74

Bold values indicate better average times.
Symbol ** indicates that the maximum time reached by GA is lower than the minimum times reached by SO-S and SO-LBS.
Symbol * indicates that the maximum time achieved by GA is lower than the minimum time reached by SO-LBS.

TABLE VIII
AVERAGE, MINIMUM AND MAXIMUM TIME REQUIRED FOR SEQUENTIALLY DEVELOPING THE e EVENTS ACCORDING TO THE SOLUTIONS PROVIDED BY THE GENETIC ALGORITHM (GA), THE METHOD SO-S AND THE METHOD SO-LBS, FOR EACH INSTANCE SET WHERE $G = 6$.

Instance set	Time required to develop the *e* events (minutes)								
	Average			Minimum			Maximum		
	GA	SO-S	SO-LBS	GA	SO-S	SO-LBS	GA	SO-S	SO-LBS
G6_R10_M4_HD	**1632.16**	2054.47	7012.91	1540.69	1854.34	6470.43	**1777.68	2253.19	7762.28
G6_R10_M4_LD	**1175.89**	1700.56	8075.41	1100.02	1511.84	6186.76	**1251.04	1957.49	9527.43
G6_R10_M8_HD	**2287.03**	2723.32	8575.38	1892.85	2350.07	7892.42	*2563.28	2997.96	9355.02
G6_R10_M8_LD	**1750.18**	2290.02	9903.45	1580.99	2078.08	8826.13	**1954.37	2600.26	11933.06
G6_R10_M16_HD	**2846.20**	3247.44	10343.77	2602.81	2980.38	9724.17	*3032.71	3441.78	11510.40
G6_R10_M16_LD	**2239.64**	2578.77	11406.47	2015.31	2284.69	10200.17	*2433.70	2788.39	12516.34
G6_R15_M4_HD	**2469.33**	3377.21	10982.58	2221.06	3049.90	9621.01	**2661.59	3834.65	12770.11
G6_R15_M4_LD	**1737.55**	2549.38	11817.92	1629.32	2239.82	9396.10	**1822.74	2870.16	13485.04
G6_R15_M8_HD	**3348.05**	4121.49	13668.98	3227.14	3986.69	12608.51	**3452.35	4193.34	14622.95
G6_R15_M8_LD	**2469.10**	3204.31	14645.48	2308.23	2900.31	13487.35	**2693.32	3725.55	17160.75
G6_R15_M16_HD	**4235.76**	4829.86	15511.42	3962.95	4318.85	14388.85	*4560.01	5165.77	16380.50
G6_R15_M16_LD	**3320.85**	4036.60	17582.00	3138.14	3785.56	16537.87	**3517.34	4337.99	19139.98
G6_R20_M4_HD	**3215.55**	4305.68	14174.64	2843.97	3936.14	12277.75	**3472.56	4698.32	15975.80
G6_R20_M4_LD	**2198.87**	3334.68	15423.72	1933.96	3013.68	13954.73	**2579.07	3851.99	17818.28
G6_R20_M8_HD	**4518.97**	5454.61	17594.41	4029.12	4965.27	16335.39	**4797.40	5743.91	18328.64
G6_R20_M8_LD	**3349.98**	4452.32	20443.13	3149.82	4130.60	19037.71	**3571.41	4640.35	21364.12
G6_R20_M16_HD	**5673.16**	6589.09	21270.05	5416.38	6394.44	20092.72	**5898.53	6964.77	22342.04
G6_R20_M16_LD	**4212.40**	5217.01	23686.65	4107.39	5012.78	23036.96	**4329.39	5604.92	25566.14

Bold values indicate better average times.
Symbol ** indicates that the maximum time reached by GA is lower than the minimum times reached by SO-S and SO-LBS.
Symbol * indicates that the maximum time achieved by GA is lower than the minimum time reached by SO-LBS.

TABLE IX
AVERAGE, MINIMUM AND MAXIMUM RPD (%) VALUE REACHED BY THE GENETIC ALGORITHM REGARDING SO-S AND SO-LBS, FOR EACH INSTANCE SET WHERE $G = 2$.

Instance Set	RPD (%)					
	Average		Minimum		Maximum	
	SO-S	SO-LBS	SO-S	SO-LBS	SO-S	SO-LBS
G2_R10_M4_HD	28.95	58.17	13.98	48.69	48.13	69.52
G2_R10_M4_LD	39.01	67.64	30.18	63.21	51.89	74.15
G2_R10_M8_HD	22.63	54.63	15.06	51.12	31.46	58.62
G2_R10_M8_LD	29.42	62.46	12.80	51.96	36.58	67.14
G2_R10_M16_HD	18.26	51.99	7.83	45.29	25.81	56.31
G2_R10_M16_LD	21.77	58.69	17.89	56.65	25.06	60.55
G2_R15_M4_HD	38.25	63.52	23.17	52.56	50.13	71.48
G2_R15_M4_LD	41.87	69.42	31.12	63.46	51.26	74.06
G2_R15_M8_HD	30.32	58.91	23.52	52.37	37.64	63.90
G2_R15_M8_LD	28.69	62.41	21.07	59.24	36.04	66.47
G2_R15_M16_HD	18.52	51.69	12.55	47.66	28.20	57.68
G2_R15_M16_LD	22.85	58.54	18.20	56.32	27.68	60.41
G2_R20_M4_HD	36.34	62.01	26.69	55.50	46.11	68.72
G2_R20_M4_LD	43.40	69.94	35.69	65.25	51.26	74.44
G2_R20_M8_HD	23.26	54.72	13.76	48.24	28.37	57.49
G2_R20_M8_LD	31.54	63.53	28.15	61.82	35.97	65.82
G2_R20_M16_HD	18.44	51.79	15.44	48.79	22.49	54.63
G2_R20_M16_LD	19.97	57.17	14.12	54.36	24.89	59.85

TABLE X
AVERAGE, MINIMUM AND MAXIMUM RPD (%) VALUE REACHED BY THE GENETIC ALGORITHM REGARDING SO-S AND SO-LBS, FOR EACH INSTANCE SET WHERE $G = 4$.

Instance Set	RPD (%)					
	Average		Minimum		Maximum	
	SO-S	SO-LBS	SO-S	SO-LBS	SO-S	SO-LBS
G4_R10_M4_HD	25.73	70.40	19.13	65.81	34.69	75.77
G4_R10_M4_LD	33.99	80.17	29.26	78.61	42.12	81.69
G4_R10_M8_HD	16.65	66.20	12.37	62.72	23.86	69.82
G4_R10_M8_LD	20.91	76.31	14.35	73.61	28.57	79.40
G4_R10_M16_HD	16.61	68.35	10.59	65.55	20.94	70.96
G4_R10_M16_LD	19.42	75.63	12.06	73.26	26.89	77.98
G4_R15_M4_HD	25.58	71.37	18.99	66.57	32.90	75.78
G4_R15_M4_LD	35.96	81.13	33.50	78.96	39.39	83.00
G4_R15_M8_HD	21.76	69.89	16.64	66.39	29.34	73.49
G4_R15_M8_LD	22.57	76.46	13.04	72.67	31.11	80.32
G4_R15_M16_HD	13.99	67.27	10.51	65.58	16.74	68.50
G4_R15_M16_LD	16.41	74.29	12.10	71.99	21.29	75.41
G4_R20_M4_HD	33.34	75.81	28.30	73.44	40.72	80.03
G4_R20_M4_LD	40.13	82.24	37.21	80.89	44.01	83.65
G4_R20_M8_HD	23.52	71.13	20.54	69.09	26.94	73.22
G4_R20_M8_LD	27.53	78.30	19.16	75.92	36.51	81.77
G4_R20_M16_HD	14.89	67.81	13.10	66.12	16.89	69.63
G4_R20_M16_LD	16.98	74.96	14.01	74.05	20.38	76.69

TABLE XI
AVERAGE, MINIMUM AND MAXIMUM RPD (%) VALUE REACHED BY THE GENETIC ALGORITHM REGARDING SO-S AND SO-LBS, FOR EACH INSTANCE SET WHERE G = 6.

Instance Set	RPD (%)					
	Average		Minimum		Maximum	
	SO-S	SO-LBS	SO-S	SO-LBS	SO-S	SO-LBS
G6_R10_M4_HD	20.37	76.66	16.32	74.65	27.45	79.40
G6_R10_M4_LD	30.37	85.15	17.25	79.78	39.15	87.50
G6_R10_M8_HD	16.14	73.30	13.16	69.57	19.46	76.02
G6_R10_M8_LD	23.25	82.17	17.28	80.52	32.54	85.30
G6_R10_M16_HD	12.33	72.41	8.54	69.99	15.20	74.91
G6_R10_M16_LD	13.06	80.32	9.82	78.68	17.30	81.58
G6_R15_M4_HD	26.75	77.43	22.36	76.05	32.61	79.77
G6_R15_M4_LD	31.41	85.07	20.62	81.08	37.59	86.75
G6_R15_M8_HD	18.75	75.45	15.29	73.21	21.94	77.74
G6_R15_M8_LD	22.68	83.09	17.95	81.77	28.01	84.41
G6_R15_M16_HD	12.20	72.70	8.24	71.11	19.10	74.26
G6_R15_M16_LD	17.72	81.11	16.05	80.48	19.18	81.68
G6_R20_M4_HD	25.17	77.14	16.40	73.20	31.62	80.45
G6_R20_M4_LD	34.12	85.75	29.77	85.00	39.28	86.80
G6_R20_M8_HD	17.13	74.32	14.28	73.15	20.15	75.34
G6_R20_M8_LD	24.71	83.58	20.91	82.40	28.08	84.84
G6_R20_M16_HD	13.89	73.31	12.31	72.26	15.42	73.98
G6_R20_M16_LD	19.15	82.20	15.20	81.40	24.19	83.38

2) RUNTIME ANALYSIS

Tables XII-XIV present the average runtime (in minutes) required by the genetic algorithm (GA), the method SO-S and the method SO-LBS, for each one of the 54 instance sets utilized in the computational experiments. The computational experiments were developed on a PC equipped with an AMD Ryzen 5 with six 2022 MHz cores, 16 GB of RAM, and SDD, running Manjaro. Moreover, the genetic algorithm, and the methods SO-S and SO-LBS, were implemented in Java 1.8.

From Tables XII-XIV, it is possible to mention that the runtime required by the genetic algorithm increases as both the number e of events considered in the instances, and the

number m of smartphones that compose the scenarios of these events, increase (see values of E and M in Tables I-III). This is consistent with the fact that, as detailed in Section III.C.1, the computing time complexity of this algorithm, $O(I*p*e*I'*p'*a)$, increases as both e and m increases (recall that a depends on m, as detailed in Eq. (8)). Regarding this, as detailed in Sections III.B and III.C.1, the genetic algorithm applies different processes (i.e., fitness evaluation process, crossover process, and mutation process) on solutions encoded as an e-tuple $<o_1, o_2, \ldots, o_e>$. Thus, the number e of events impacts the runtime of these processes, and as a result the runtime of the algorithm. Besides, given an encoded solution, the fitness evaluation process estimates the time required to develop each of the e events, by applying the scenario preparation approach proposed in [1] on the scenario of each event. Thus, the runtime of the fitness evaluation process is also impacted by the runtime required by the approach proposed in [1]. Respecting this, the runtime required by this approach depends on the number of smartphones that compose the scenario on which the approach is applied, as detailed in [1]. Therefore, the number m of smartphones that compose the scenarios of the e events impacts on the runtime of the mentioned approach, and as a result the runtime of the fitness evaluation process and the runtime of the algorithm.

Tables XII-XIV also indicate that the runtimes required by the methods SO-S and SO-LBS are significantly lower than that required by the genetic algorithm, for each of the 54 instance sets. These results coincide with the fact that the computing time complexities of SO-S and SO-LBS are considerably lower than the computing time complexity of the genetic algorithm. Recall that, the computing time complexity of SO-S is $max(O(r*e), O(e*m))$ (as detailed in Section IV.C.1), and the computing time complexity of SO-LBS is $max(O(g*e), O(e*m))$ (as detailed in Section IV.C.2), whereas the computing time complexity of the genetic algorithm is $O(I*p*e*I'*p'*a)$. Regarding this, as detailed in Section IV.C, the methods SO-S and SO-LBS determine a sequential order to develop the e events one at a time, based on a predefined ordering criterion. Specifically, SO-S sequentially orders the e events according to the scenario included in these events (i.e., order by scenario), whereas SO-LBS sequentially orders the e events according to the strategy included in these events (i.e., order by strategy). In addition, these ordering criteria do not consider or estimate the time required to develop the e events one at a time. Unlike these methods, as detailed in Section III.B, the genetic algorithm is aimed to determine the sequential order to develop the e events, in such a way that the time required for developing these e events one at a time is minimized. To achieve this aim, the algorithm works on a population of solutions encoded as the previously mentioned e-tuple, and iteratively applies the previously mentioned processes, as well as selection processes, on the encoded solutions that compose this population. Thus, the runtime of the genetic algorithm is impacted by the size of the population, the number of iterations, and also the runtimes of the mentioned processes.

However, it is necessary mentioning that, for each one of the 54 instance sets, the runtime required by the genetic algorithm to obtain the solutions for the instances is a really small percentage (i.e., < 0.40%) of the time needed for sequentially developing the e events considered in these instances according to the order indicated in these solutions. For example, for the instance set named G6_R20_M16_HD, the runtime required by the genetic algorithm to obtain the solutions for the instances (i.e., 16.75 minutes) is 0.30% of the time needed for sequentially developing the events considered in the instances according to the order indicated in these solutions (i.e., 5673.16 minutes).

Besides, it is necessary to note that, for each one of the 54 instance sets, the runtime required by the genetic algorithm to obtain the solutions plus the time needed for developing the events considered in the instances according to these solutions is lower (i.e., [12, 43]% lower) than the time needed for developing these events according to the solutions provided by SO-S, and besides is lower (i.e., [51, 85]% lower) than the time needed for developing these events according to the solutions provided by SO-LBS. For example, for the instance set G6_R20_M16_HD mentioned, the runtime required by the genetic algorithm to obtain the solutions (i.e., 16.75 minutes) plus the time needed to develop the events considered in the instances according to these solutions (i.e., 5673.16 minutes) is 5689.91 minutes. This value is 13.76% lower than the time needed to develop such events according to the solutions provided by SO-S (i.e., 6589.09 minutes), and is 73.25% lower than the time needed to develop such events according to the solutions provided by SO-LBS (i.e., 21270.05 minutes).

All in all, we can confirm that the runtime needed by the genetic algorithm to obtain the solutions for the instances used is appropriate in the context of the addressed problem.

TABLE XII
AVERAGE RUNTIME (IN MINUTES) REQUIRED BY THE GENETIC ALGORITHM (GA), THE METHOD SO-S AND THE METHOD SO-LBS, FOR EACH INSTANCE SET WHERE $G = 2$.

Instance set	Average runtime (minutes)		
	GA	SO-S	SO-LBS
G2_R10_M4_HD	0.51	1.0E-06	1.0E-06
G2_R10_M4_LD	0.52	1.0E-06	1.0E-06
G2_R10_M8_HD	0.77	1.0E-06	1.0E-06
G2_R10_M8_LD	0.78	1.0E-06	1.0E-06
G2_R10_M16_HD	1.34	1.0E-06	1.0E-06
G2_R10_M16_LD	1.36	1.0E-06	1.0E-06
G2_R15_M4_HD	1.15	1.0E-06	1.0E-06
G2_R15_M4_LD	1.16	1.0E-06	1.0E-06
G2_R15_M8_HD	1.72	1.0E-06	1.0E-06
G2_R15_M8_LD	1.80	1.0E-06	1.0E-06
G2_R15_M16_HD	2.97	1.0E-06	1.0E-06
G2_R15_M16_LD	3.20	1.0E-06	1.0E-06
G2_R20_M4_HD	2.07	1.0E-05	1.0E-05
G2_R20_M4_LD	2.07	1.0E-05	1.0E-05
G2_R20_M8_HD	3.13	1.0E-05	1.0E-05
G2_R20_M8_LD	3.20	1.0E-05	1.0E-05
G2_R20_M16_HD	5.45	1.0E-05	1.0E-05
G2_R20_M16_LD	5.76	1.0E-05	1.0E-05

TABLE XIII
AVERAGE RUNTIME (IN MINUTES) REQUIRED BY THE GENETIC ALGORITHM (GA), THE METHOD SO-S AND THE METHOD SO-LBS, FOR EACH INSTANCE SET WHERE $G = 4$.

Instance set	Average runtime (minutes)		
	GA	SO-S	SO-LBS
G4_R10_M4_HD	2.53	1.0E-05	1.0E-05
G4_R10_M4_LD	2.55	1.0E-05	1.0E-05
G4_R10_M8_HD	3.26	1.0E-05	1.0E-05
G4_R10_M8_LD	3.36	1.0E-05	1.0E-05
G4_R10_M16_HD	5.22	1.0E-05	1.0E-05
G4_R10_M16_LD	5.07	1.0E-05	1.0E-05
G4_R15_M4_HD	3.52	1.0E-04	1.0E-04
G4_R15_M4_LD	3.54	1.0E-04	1.0E-04
G4_R15_M8_HD	4.53	1.0E-04	1.0E-04
G4_R15_M8_LD	4.67	1.0E-04	1.0E-04
G4_R15_M16_HD	7.26	1.0E-04	1.0E-04
G4_R15_M16_LD	7.05	1.0E-04	1.0E-04
G4_R20_M4_HD	4.89	1.0E-03	1.0E-03
G4_R20_M4_LD	4.93	1.0E-03	1.0E-03
G4_R20_M8_HD	6.30	1.0E-03	1.0E-03
G4_R20_M8_LD	6.49	1.0E-03	1.0E-03
G4_R20_M16_HD	10.09	1.0E-03	1.0E-03
G4_R20_M16_LD	9.80	1.0E-03	1.0E-03

TABLE XIV
AVERAGE RUNTIME (IN MINUTES) REQUIRED BY THE GENETIC ALGORITHM (GA), THE METHOD SO-S AND THE METHOD SO-LBS, FOR EACH INSTANCE SET WHERE $G = 6$.

Instance set	Average runtime (minutes)		
	GA	SO-S	SO-LBS
G6_R10_M4_HD	3.49	1.0E-04	1.0E-04
G6_R10_M4_LD	3.54	1.0E-04	1.0E-04
G6_R10_M8_HD	4.28	1.0E-04	1.0E-04
G6_R10_M8_LD	4.34	1.0E-04	1.0E-04
G6_R10_M16_HD	7.47	1.0E-04	1.0E-04
G6_R10_M16_LD	7.11	1.0E-04	1.0E-04
G6_R15_M4_HD	5.84	1.0E-03	1.0E-03
G6_R15_M4_LD	5.89	1.0E-03	1.0E-03
G6_R15_M8_HD	7.53	1.0E-03	1.0E-03
G6_R15_M8_LD	7.76	1.0E-03	1.0E-03
G6_R15_M16_HD	12.05	1.0E-03	1.0E-03
G6_R15_M16_LD	11.71	1.0E-03	1.0E-03
G6_R20_M4_HD	8.12	1.0E-02	1.0E-02
G6_R20_M4_LD	8.18	1.0E-02	1.0E-02
G6_R20_M8_HD	10.46	1.0E-02	1.0E-02
G6_R20_M8_LD	10.78	1.0E-02	1.0E-02
G6_R20_M16_HD	16.75	1.0E-02	1.0E-02
G6_R20_M16_LD	16.27	1.0E-02	1.0E-02

V. DISCUSSION

In this section, the main contributions of BAGESS are discussed. After that, the validity of the results obtained by BAGESS is explained.

1) CONTRIBUTIONS

In lack of previous support to automate experiments involving battery-driven mobile devices, in this work, a new software component called BAGESS is proposed and then evaluated, in order to address the problem of sequentially ordering several load-balancing experimental scenarios, so that the time required to prepare these scenarios one at a time is minimized. It must be taken into account that, for preparing a single scenario composed of several battery-driven devices, charging and discharging actions must be applied on each device to reach its target battery level pre-configured in the scenario from its current battery level. The preparation of a single scenario was addressed by applying the approach proposed in [1]. However, in the duty of comparatively evaluating diverse load-balancing strategies, many scenarios that are composed of the same battery-driven devices, and vary regarding the target battery levels pre-configured for these devices, need to be prepared.

Commonly, after evaluating a load-balancing strategy on a single prepared scenario, the target battery level reached by each device via the scenario preparation suffers a decrease due to the work-load assigned by the strategy to the devices, which defines the current battery level of each device in the next scenario to be prepared. The problem addressed by BAGESS is determining the next scenario to be prepared, considering the closeness between the current and pre-configured target battery levels of the devices in the scenario. More specifically, given a set of load-balancing scenarios, each one provided with target battery levels pre-configured for the devices, and given the load-balancing strategy to be considered in each scenario, detailing the estimated battery decreases that the devices can suffer once the strategy is evaluated on them, BAGESS provides a feasible sequential order to prepare the scenarios one at a time, minimizing the time required to prepare these scenarios.

BAGESS effectiveness was evaluated on 540 different problem instances that vary in the amount of scenarios, amount of devices composing these scenarios, target battery levels pre-configured for these devices, and load-balancing strategies. Sequencing scenarios with BAGESS provides researchers an average saving of [12, 43]% and [51, 85]% regarding the methods SO-S and SO-LBS, respectively, in terms of the time required to sequentially prepare the scenarios one at a time.

To use BAGESS for devices different from those provided in the public repository, the user must collect battery traces, a task that can be time consuming but is facilitated and can be achieved with the profiling platform explained in [15] and Motrol [8]. Collecting traces is not only required for applying BAGESS, but also for simulating the execution of a given load-balancing strategy in the context of DewSim [18], which means that the benefit is twofold. A limitation of BAGESS is that the resulting sequential ordering of the scenarios does not consider devices battery's thermal stress, which might affect them when exposing batteries to several charging/discharging cycles without resting time between scenarios preparation. Finally, with BAGESS, the capabilities of a generic testing platform are expanded; it aims at being an integral tool to facilitate experimental

reproducibility, and can be applied for evaluating and validating load balancing strategies for Edge/Dew Computing environments.

2) RESULTS VALIDITY

The accuracy of our results is explained by the accuracy of our experimental methodology, specifically the data source used for simulating discharging/charging events and the trace-based simulation approach that timely relates battery level drops/increments with mobile device components usage. All the data employed by this approach is profiled using a controlled procedure performed over real battery-powered devices. Details of such a procedure along with validation tests, can be found in [18]. Thus, we refer as device profile to the data set composed by traces with information of battery level behavior for various discrete components usage. In the current work, we used four device profiles corresponding to the following smartphone models: Xiaomi Mi A2 lite, Xiaomi Redmi Note 7, Motorola Moto G6 and Samsung A30. Each device profile is composed of twenty full baseline battery behavior traces – from 2% to 100% – that allow researchers to model a wide variety of charging/discharging scenarios considering steady CPU usages of 0, 30, 50, 75 and 100 percentage and on/off screen states. Despite being a time consuming task, once obtained, a device profile can be used as many times as needed to feed the simulation methodology [18] to faithfully simulate battery level drops and increments, i.e., -1% and +1% (dis)charging scenarios in a timeline. Moreover, this approach has been used to represent battery behavior in other works [28, 29] where different load balancing strategies for mobile distributed computing were studied. In this work, we use (dis)charging battery profiles as input for simulating the preparation time of mobile devices clusters running a sequence of load balancing tests scenarios.

Our experiment design considers problem instances that are representative of load-balancing evaluation scenarios for mobile distributed computing, and sufficient because the particular configuration values used are realistic and practicable in in-vivo tests with the testing platform with which this work contributes. The variables considered in the problem instances include different cluster sizes (i.e., number of smartphones) – with random combinations of the smartphone models previously referred whose initial battery level is also randomly set –, number of load balancing strategies commonly used when performing in-lab comparisons, number of scenarios usually used to evaluate each load balancing strategy, estimated battery drops that the evaluation of each strategy produces on devices, and finally, the number of scenario preparation events which results from multiplying the number of evaluation scenarios times the number of load balancing strategies. In total 540 problem instances were run. Variables and values were explained in Section IV.A.

Providing that constituent parts of genetic algorithms are subject to parameters fine tuning, which is done via exploratory tests to find the appropriate population size, number of solutions competing in tournament selection, crossover probability, mutation probability, among other parameters, we run preliminary tests on the problem instances. Moreover, given that randomness is present in solutions obtained with genetic algorithms, with the aim of reporting reliable results we have foreseen that solution quality comparisons of our proposal w.r.t other heuristics were done considering the average of 30 runs of the genetic algorithm for every problem instance. Besides, we run Man-Witnney tests, in order to test statistical significance of such results.

VI. RELATED WORKS

Measuring and comparing the outcome of different load balancing techniques when scavenging computing resources using a set of real battery-powered nodes, such as smartphone clusters, requires a systematic procedure to restore experimental conditions between one measurement and another. When measurements are based on metrics derived directly or indirectly from nodes battery level, e.g., fairness, restoration implies to perform dis/charging events over several physical batteries until they reach pre-defined levels of charge configured in the scenario under evaluation. Indeed, it is a time-consuming task that takes even longer when it is performed without taking advantage of proximity between current and target battery levels, causing in turn unnecessary smartphone batteries over utilization.

Studies evaluating load balancing techniques and their impact on battery usage in distributed mobile computing environments comprising real battery-powered nodes clusters are scarce. A common practice to measure such impact is through virtually modeled batteries [27]. For instance, Mattia & Beraldi [12] propose a load balancing technique for maximizing the lifespan of homogeneous SBCs (Single Board Computer) battery-driven clusters. Experiments were done with real SBC clusters, but battery charging and discharging behaviors are simulated using SBCs energy consumption profiles and real-time measurements of CPU usage in different states (idle, running). Aslanpour et al. [13] propose an energy-aware scheduling algorithm for operating a serverless Edge computing cluster powered with battery-driven nodes. The algorithm prioritizes the execution and migration of functions replicas differentiating between well-powered, low-powered, vulnerable and powerless nodes. Scheduling comparisons are performed with a real cluster of Raspberry Pis and using real traces of renewable energy to simulate different charging rates of a virtually modeled battery. In relation to model batteries virtually, it is important to characterize node performance and energy consumption. Aslanpour et al. [14] propose WattEdge, a fine-grained profiling framework to measure nine energy consumption factors in edge nodes including connectivity, memory,

storage, CPU, among others, through different stress tests. The framework is structured in separate software and hardware modules that run tests, monitor hardware usage and temperature, and log current, volts and wattage parameters. The framework also provides some stress tests for profiling battery charging and discharging behavior under variable conditions via PiJuice, a portable power platform specially designed for Raspberry Pi. Other work in this field [15] proposes a profiling and benchmarking tool targeting Android mobile devices to automate battery trace capturing under configurable resources usage. Tools like these help in automating data gathering that is used as input for modeling complex entities such as battery behavior. The methodology [18] in which we base the validation of BAGESS also relies on a custom Android application to data gathering (profile device building), though the purpose of our toolset is to realize full in-vivo experimentation with smartphone clusters.

In [10, 11, 23], like in our work, performed experiments include smartphones as battery-powered nodes, however the impact of the load balancing proposals on batteries were not assessed. By contrast, the measurements were centered on metrics such as throughput, speedup, data transferring volume varying nodes quantity, nodes topology and tasks division schemes. Energy-related metric included in these works relates power consumption with resource utilization, which is obtained via a profiling procedure and a power monitor device. Works using residual battery capacity indicators over real batteries are [24, 25]. The indicator can be obtained through the battery management API available in mobile operating systems. The experiments show how the proposed load balancing approach impacts on the battery level of smartphones which have heterogeneous computing capabilities and battery capacities. It was not within the objectives of the experimental methodology employed in these works to provide a systematic approach for replicating experiments and/or testing under varying initial battery levels, features that would help to increase results robustness and statistical confidence. Our work aims at filling this gap.

The necessity of performing tests on real batteries is specifically present in research attained to the improvement of battery management systems. In [26] a HIL (hardware in loop) set up and procedure was proposed to test and validate battery-fuel gauge algorithms. These algorithms track the battery state of charge that is informed to users as remaining battery percentage. The HIL includes customized aluminum casts where batteries are kept while being tested under varying operation conditions. The temperature inside the cast can be set and controlled within a range from -25 to +45 Celsius. Another hardware plugged into this case is used to inject programmable loads and measure battery parameters like voltage, current and temperature. Like the module we have proposed, the described HIL and procedure aim at automating the preparation of tests to be run over real batteries. A fundamental difference between [26] and our testing platform is that in [26] several batteries are treated as different study subjects, while in our work, several batteries - smartphones- are treated as a single study object (i.e. smartphone cluster). To the best of our knowledge, our work is the first proposing a software-based systematic approach for preparing a set of tests which, in conjunction with hardware of our own previously published, represent another step towards facilitating the empirical assessment of load balancing techniques over real battery-powered settings, and particularly smartphone clusters.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, the problem of defining the sequential order to develop a given number of load-balancing scenario preparation events has been addressed, considering that the total time required to develop them should be minimized. This problem has been modeled as the well-known ATSP, and, therefore, is a NP-Hard problem.

To solve the addressed problem, a novel software module named BAGESS has been proposed, which uses a genetic algorithm to determine the sequential order in which the given events should be developed, in such a way that the total time required to develop them is minimized. This genetic algorithm has been specially designed to: *a*) explore many different feasible sequential orders for developing the given events one at a time, and *b*) identify the sequential order that allows developing these events at the minimal possible time.

Computational experiments have been developed with the aim of evaluating the performance of the genetic algorithm utilized by BAGESS on different representative and realistic experimental instances of the addressed problem. In this regard, the performance of this algorithm has been evaluated on 54 instance sets. Each of these instance sets contains 10 different instances, where each instance includes a number of scenario preparation events. The 54 instance sets differ in terms of the category of their instances with respect to five well-defined components (i.e., number of events, number of scenarios, number of smartphones that compose the scenarios, number of load-balancing strategies, and estimated battery variation that the smartphones suffer once each strategy is evaluated). After that, the performance of this algorithm on each of the 54 instance sets has been compared with those of the two methods currently used for determining the sequential order to develop a given number of events one at a time, namely SO-S and SO-LBS.

Based on the performance comparison developed, it is possible to conclude that the solutions given by the genetic algorithm for the instances of each set provide a significant average saving, in terms of the time (in minutes) required to sequentially develop the events one at a time. Specifically, for all instance sets, the average saving of the solutions given by the genetic algorithm regarding the solutions given by SO-S is on the range [12, 43]%, and regarding the solutions given by SO-LBS is on the range [51, 85]%. Therefore, the solutions given by the genetic algorithm significantly reduce

the time required for sequentially developing the events considered in the instances used.

In future work, the incorporation of other optimization objectives in the addressed problem will be analyzed. The currently considered objective is minimizing the time required to sequentially develop the given events one at a time. As detailed in Section II, the development of each one of these events consists of applying sequences of battery charge/discharge actions on the smartphones considered in the scenario inherent to the event, in order to reach the target battery levels predefined for these smartphones, from the corresponding current battery levels. It is necessary to mention that the application of these sequences of battery charge/discharge actions on these smartphones can wear the smartphones' batteries, when there is not adequate resting time for these batteries between the developments of the different events, which can reduce the lifespan of these batteries. Due to this, it is convenient to minimize the wear of the smartphones' batteries during the development of the given events. A possible alternative to do that is including resting times for the smartphones' batteries between the developments of the different given events. However, this alternative increases the total time required to sequentially develop the given events. Taking all this into account, an bi-objective extension of the problem addressed in this work emerges, which implies defining the sequential order for developing a given number of events, in such a way that: *a*) the total time required to develop these events one at a time is minimized, and *b*) the wear suffered by the batteries of the smartphones considered in these events is minimized. This extension considers two objectives that are relevant and also conflictive in the context of sequentially developing the given scenario preparation events.

This bi-objective extension of the addressed problem will be studied in detail, which includes analyzing alternatives to estimate the wear that can be suffered by the smartphones' batteries, throughout the development of the given events. Besides, given that this bi-objective extension is an NP-Hard problem, the application of multi-objective meta-heuristic algorithms will be analyzed to solve this extension. Initially, the application of multi-objective genetic algorithms will be analyzed (e.g., the well-known NSGA-III [30]), considering the solution encoding, and also the crossover and mutation processes, utilized in the genetic algorithm proposed in this work. In addition, different alternatives for minimizing the runtime required by these multi-objective genetic algorithms will be studied, with the aim of these algorithms being able to provide solutions in an acceptable runtime to BAGESS. One of these alternatives involves parallelizing the search and optimization process carried out by these multi-objective algorithms [17].

Author Contributions: Conceptualization, V.Y., M.H., J.T., C.M.; Methodology, V.Y.; Software, V.Y., M.H., J.T., C.M; Validation, V.Y.; Formal Analysis, V.Y.; Investigation, V.Y, M.H., T.M., T.G., A.Z., C.M.; Resources, A.Z., C.M.; Data Curation, V.Y.; Writing – Original Draft Preparation, V.Y., M.H., T.M., T.G.; Visualization, V.Y., M.H.; Supervision, M.H., C.M.; Project Administration, C.M. All authors have checked the manuscript and have agreed to the submission in the specified author order.

Data Availability Statement: The software and data (i.e. problem instances) used to evaluate the genetic algorithm can be found at https://github.com/matieber/BAGESS

REFERENCES

[1] Yannibelli, V., Hirsch, M., Toloza, J., Majchrzak, T.A., Zunino, A., and Mateos, C., "Speeding up Smartphone-Based Dew Computing: In Vivo Experiments Setup Via an Evolutionary Algorithm", *Sensors*, 23(3), pp. 1388, 2023.

[2] Pop, P.C., Cosma, O., Sabo, C., and Sitar, C.P., "A comprehensive survey on the generalized traveling salesman problem", *European Journal of Operational Research*, vol. 314(3), pages 819-835, 2024.

[3] Eiben, A. E., and Smith, J. E., *Introduction to Evolutionary Computing*, *2nd. Edition*. Ed. Berlin, Germany: Springer, 2015.

[4] Mann, H. B., and Whitney, D. R., "On a test of whether one of two random variables is stochastically larger than the other", *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[5] Ahammad, I., "Fog Computing Complete Review: Concepts, Trends, Architectures, Technologies, Simulators, Security Issues, Applications, and Open Research Fields", *SN Computer Science*, 4(6), pp.765, 2023.

[6] Hirsch, M., Mateos, C. and Zunino, A., "Augmenting computing capabilities at the edge by jointly exploiting mobile devices: A survey", *Future Generation Computer Systems*, 88, pp.644-662. 2018.

[7] Mateos, C., Hirsch, M., Toloza, J.M. and Zunino, A. "LiveDewStream: A stream processing platform for running in-lab distributed deep learning inferences on smartphone clusters at the edge", *SoftwareX*, 20, pp.101268, 2022.

[8] Toloza, J.M., Hirsch, M., Mateos, C. and Zunino, A., "Motrol: A hardware-software device for batch benchmarking and profiling of in-lab mobile device clusters", *HardwareX*, 12, p.e00340, 2022.

[9] Rajwar, K., Deep, K., and Das, S., "An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open

challenges", *Artificial Intelligence Review*, vol. 56, pp. 13187–13257, 2023.

[10] Huang, Yakun, et al. "Enabling DNN acceleration with data and model parallelization over ubiquitous end devices." IEEE Internet of Things Journal 9.16 (2021): 15053-15065.

[11] Mao, Jiachen, et al. "Modnn: Local distributed mobile computing system for deep neural network." Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, 2017.

[12] G. P. Mattia and R. Beraldi, "Lifespan and energy-oriented load balancing algorithms across sets of nodes in Green Edge Computing", in *IEEE Cloud Summit*, Baltimore, MD, USA, 2023, pp. 41-48.

[13] Aslanpour, M.S., Toosi, A.N., Cheema, M.A. and Gaire, R., "Energy-aware resource scheduling for serverless edge computing", in *22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 190-199.

[14] Aslanpour, M.S., Toosi, A.N., Gaire, R. and Cheema, M.A., "WattEdge: a holistic approach for empirical energy measurements in edge computing", in *19th International Conference on Service-Oriented Computing (ICSOC 2021)*, Virtual Event, November 22–25, 2021, pp. 531-547.

[15] Hirsch, M., Mateos, C., Zunino, A. and Toloza, J., "A platform for automating battery-driven batch benchmarking and profiling of Android-based mobile devices", *Simulation Modelling Practice and Theory*, 109, pp.102266, 2021.

[16] Luo, Q., Hu, S., Li, C., Li, G. and Shi, W., "Resource scheduling in edge computing: A survey", *IEEE Communications Surveys & Tutorials*, 23(4), pp.2131-2165, 2021.

[17] Falcón-Cardona, J.G., Hernández-Goméz, R., Coello, C.A., Catillo Tapia, M.G., "Parallel Multi-Objective Evolutionary Algorithms: A Comprehensive Survey", *Swarm and Evolutionary Computation*, 67, pp. 100960, 2021.

[18] Hirsch, M., Mateos, C., Rodriguez, J.M. and Zunino, A., "DewSim: A trace-driven toolkit for simulating mobile device clusters in Dew computing environments", *Software: Practice and Experience*, 50(5), pp.688-718, 2020.

[19] Dantzig, G.; Fulkerson, J.; Johnson, S., "Solution of a Large-Scale Traveling-Salesman Problem", *Operations Research*, Vol. 2, pp.393–410, 1954.

[20] Wu, C., Fu, X., Pei, J., and Dong, Z., "A Novel Sparrow Search Algorithm for the Traveling Salesman Problem", *IEEE Access*, Vol. 9, pp. 153456-153471, 2021.

[21] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C., *Introduction to Algorithms, Fourth Edition*. Ed. Cambridge, Massachusetts: The MIT Press, 2022.

[22] Sedgewick, R., and Wayne, K., *Algorithms, Fourth Edition*. Ed. United States: Addison-Wesley, 2011.

[23] Loke, Seng W., et al. "Mobile computations with surrounding devices: Proximity sensing and multilayered work stealing", ACM Transactions on Embedded Computing Systems (TECS) 14.2 (2015): 1-25.

[24] Viswanathan, Hariharasudhan, et al. "Uncertainty-aware autonomic resource provisioning for mobile cloud computing", IEEE transactions on parallel and distributed systems 26.8 (2014): 2363-2372.

[25] Ghasemi-Falavarjani, Simin, Mohammadali Nematbakhsh, and Behrouz Shahgholi Ghahfarokhi. "Context-aware multi-objective resource allocation in mobile cloud", Computers & Electrical Engineering 44 (2015): 218-240.

[26] Avvari, G. V., et al. "Experimental set-up and procedures to test and validate battery fuel gauge algorithms", Applied Energy 160 (2015): 404-418.

[27] Mesdaghi, A., and Mollajafari, M. "Improve performance and energy efficiency of plug-in fuel cell vehicles using connected cars with V2V communication", Energy Conversion and Management 306 (2024): 118296.

[28] Hirsch, Matías, Juan Manuel Rodriguez, Cristian Mateos, and Alejandro Zunino. "A two-phase energy-aware scheduling approach for cpu-intensive jobs in mobile grids", Journal of Grid Computing 15 (2017): 55-80.

[29] Hirsch, Matías, Cristian Mateos, Alejandro Zunino, Tim A. Majchrzak, Tor-Morten Grønli, and Hermann Kaindl. "A task execution scheme for dew computing with state-of-the-art smartphones", Electronics 10, no. 16 (2021): 2006.

[30] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints," IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577–601, 2014.

[31] https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-2.html#jvms-2.2. Accessed August 6, 2024.

VIRGINIA YANNIBELLI received a PhD. Degree in Computer Science in 2009, and a BSc. Degree in Systems Engineering in 2005, at the Faculty of Exact Sciences, UNICEN University, Argentina. She is Professor at the Department of Computing, Faculty of Exact Sciences, UNICEN University, and besides Scientific Researcher at the National Scientific and Technological Research Council of Argentina (CONICET).

She is moreover a member of the ISISTAN Institute (Tandil Software Engineering Research Institute), UNICEN-CONICET, Argentina. She has served as guest editor in special issues from reputed international journals (e.g., Information Processing and Management Journal published by Elsevier), and also as organizer/chair of workshops/tracks from reputed international conferences (e.g., IJCAI and MICAI). Her research areas include Artificial Intelligence, Evolutionary Computing, Metaheuristics, Optimization, Scheduling, and Cloud/Dew Computing.

MATÍAS HIRSCH graduated as Systems Engineering from Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA), Buenos Aires, Argentina in 2010. Since then, at the beginning of his professional career, he gain experience working for several software factory companies. In 2013, he gained a scholarship to start his doctoral studies. In 2018 he received his Ph.D in Computer Science from UNCPBA. Currently, he is an assistant professor at UNCPBA, a member of the Instituto Superior de Ingenieria de Software Tandil (ISISTAN) and, since 2020, he has held a position as a researcher at CONICET. Under the supervision of Prof. Cristian Mateos and Prof. Alejandro Zunino, he has published around 16 Sci-index Journal papers and 9 conference papers. His research interests are in the fields of Parallel and Distributed Computing, Mobile Computing, Edge Computing, Dew Computing, IoT, Big Data.

JUAN TOLOZA graduated as Systems Engineer from the National University of the Center of the Province of Buenos Aires (UNICEN), Buenos Aires, Argentina in 2009. In 2009, he obtained a scholarship to begin his doctoral studies. In 2013, he received his PhD in Computer Science from UNLP. Currently, he is an adjunct ordinary professor at UNICEN, a member of the Instituto Superior de Ingenieria de Software Tandil (ISISTAN) and, since 2020, he has held the position of assistant professional of CONICET under the supervision of Prof. Alejandro Zunino. He has published 23 journal papers and 22 conference papers. His research interests are embedded systems, control systems, Distributed Computing, Mobile and IoT.

TIM A MAJCHRZAK (M'08–SM'18) received Bachelor and Master of Science in Information Systems degrees from the University of Münster, Germany, in 2006 and 2007, respectively. He received his PhD from the University of Münster in 2011. He currently is professor in Information Systems at the University of Agder (UiA) in Kristiansand, Norway. He also is a member of the Centre for Integrated Emergency Management (CIEM) at UiA.

His research comprises technical, socio-technical, and organizational aspects of Software Engineering, often in the context of Mobile Computing. He also engages in diverse interdisciplinary Information Systems topics, most notably targeting Crisis Prevention and Management. His research projects typically have an interface to industry and society.

Prof. Majchrzak is a senior member of the IEEE and the IEEE Computer Society, a member of the Gesellschaft für Informatik e.V., and a member of the Association for Information Systems (AIS).

TOR-MORTEN GRØNLI (M'03) received Bachelor in Information Technology from Norwegian School of IT in 2004, and Master of Technology in Computer Science degree from the Brunel University, London, UK in 2007. He received his PhD from Brunel University, London, UK in 2011. He currently is professor in applied computer science at Kristiania University College (KUC) in Oslo, Norway. He is a founding member of the Mobile Technology Lab at KUC.

His research comprises technical, sociotechnical, and applied perspectives of Software Engineering, often in the context of Mobile Computing. He also engages in diverse interdisciplinary research, bridging from applied computer science to information systems. His research projects typically have an interface to industry and society.

Prof. Grønli is a member of the IEEE, IEEE Computer Society, ACM and of the Association for Information Systems (AIS).

ALEJANDRO ZUNINO received a PhD. Degree in Computer Science in 2003. He is the Director of the Tandil Scientific and Technological Center (CCT CONICET Tandil) and Director of the Tandil Institute of Software Engineering (ISISTAN) of the National University of the Center of Buenos Aires Province (UNICEN) and the National Scientific and Technical Research Council (CONICET). He is an Associate professor at the Department of Computing, UNICEN and a Principal Researcher of CONICET. He has published more than 100 journals papers and more than 80 conference papers. His main research interests are in the area of distributed computing, including edge, dew, mobile and service-oriented computing.

CRISTIAN MATEOS received a PhD. Degree in Computer Science in 2008, a MSc. Degree in Systems Engineering in 2005, and a BSc. Degree in Systems Engineering in 2002 at the Faculty of Exact Sciences, UNICEN University, Argentina. He is a Full Associate Professor at the UNICEN, and Independent Researcher at the National Scientific Council of Argentina (CONICET).

He has published 99 papers in journals (77 ISI-indexed), 68 papers in national and international conferences, and 11 book chapters (+2915 citations according to GS). He has served as (lead) guest editor in special issues from reputed venues (e.g. Future Generation Computer Systems - Elsevier, and Information Processing and Management - Elsevier). His main research interests are parallel and distributed programming, with a special emphasis on methods for gridifying/parallelizing applications, application-level parallelism, (mobile) Grid/Cloud middlewares and platforms, Service-Oriented Computing and Web Services.