Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID-19. The COVID-19 resource centre is hosted on Elsevier Connect, the company's public news and information website.

Computer Communications 161 (2020) 225–237

Contents lists available at ScienceDirect

Computer Communications

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

Performance evaluation of hybrid crowdsensing systems with stateful CrowdSenSim 2.0 simulator☆

Federico Montori [a,*], Luca Bedogni [b], Claudio Fiandrino [c], Andrea Capponi [d], Luciano Bononi [a]

[a] University of Bologna, Italy
[b] University of Modena and Reggio Emilia, Italy
[c] IMDEA Networks Institute, Madrid, Spain
[d] University of Luxembourg, Luxembourg

ARTICLE INFO

Keywords:
Mobile crowdsensing
Simulation
Modeling
Distributed algorithms

ABSTRACT

Mobile crowdsensing (MCS) has become a popular paradigm for data collection in urban environments. In MCS systems, a crowd supplies sensing information for monitoring phenomena through mobile devices. Depending on the degree of involvement of users, MCS systems can be participatory, opportunistic or hybrid, which combines strengths of above approaches. Typically, a large number of participants is required to make a sensing campaign successful which makes impractical to build and deploy large testbeds to assess the performance of MCS phases like data collection, user recruitment, and evaluating the quality of information. Simulations offer a valid alternative. In this paper, we focus on hybrid MCS and extend CrowdSenSim 2.0 in order to support such systems. Specifically, we propose an algorithm for efficient re-route users that would offer opportunistic contribution towards the location of sensitive MCS tasks that require participatory-type of sensing contribution. We implement such design in CrowdSenSim 2.0, which by itself extends the original CrowdSenSim by featuring a stateful approach to support algorithms where the chronological order of events matters, extensions of the architectural modules, including an additional system to model urban environments, code refactoring, and parallel execution of algorithms.

1. Introduction

Mobile CrowdSensing (MCS) gained exponential interest in the last years and has become one of the most promising paradigms for data collection in urban environments within the scope of smart cities [1]. MCS systems gather data from sensors typically embedded in citizens' mobile devices, such as smartphones, tablets, and wearables. The number of worldwide smartphones sales is still increasing according to Gartner statistics, reaching 1.55 billion units in 2018 [2]. The crowd analytics market is projected to reach USD 1 142.5 million by 2021, raising from USD 385.1 million of 2016 at a compound annual growth rate of 24.3% [3].

Data acquisition can be either participatory or opportunistic, depending on the degree of involvement of the users in sensing processes. Participatory sensing systems directly tasks users by specifying a set of requests, e.g., to record a sound using the microphone [4]. To be effective and mitigate the risk of obtaining little contribution, participatory approaches require specific incentive mechanisms and recruitment policies [5]. Opportunistic sensing systems do not task users directly user, but rather the applications themselves are responsible for taking sensing decisions that are typically context-aware. Hybrid approaches combine the strengths of both paradigms [6]. A number of applications in the context of public health monitoring, safety and emergency can benefit from hybrid schemes [7]. Public health monitoring is particularly relevant nowadays because of the recent outbreak of the COVID-19 virus and crowdsensing techniques can help mitigate and control the rate of diffusion. The concept of hybrid in MCS is also employed to refer to those systems that mix static sensor networks and mobility provided by MCS. HySense [8] is one of such frameworks. Its objective is to leverage the fixed sensing architecture when mobile nodes contribute too little so to balance sensing opportunities in different regions make the resulting data contribution as uniform as possible. Following a similar principle, the authors of [9] build a multi-sensor platform to create high-resolution pollution maps.

The success of a MCS campaign typically relies on large participation of users [10]. Unfortunately, often it is not feasible to develop testbeds and platforms that involve a multitude of citizens.[1] On the one hand, the cost of recruitment scales with the number of users involved

☆ This paper is an extension of Montori et al. (2019).

* Corresponding author.
E-mail addresses: federico.montori2@unibo.it (F. Montori), luca.bedogni@unimore.it (L. Bedogni), claudio.fiandrino@imdea.org (C. Fiandrino), andrea.capponi@uni.lu (A. Capponi), luciano.bononi@unibo.it (L. Bononi).

[1] In the rest of the paper, we will use the terms users, citizens, and participants interchangeably.

https://doi.org/10.1016/j.comcom.2020.07.021
Received 6 April 2020; Received in revised form 1 July 2020; Accepted 10 July 2020
Available online 22 July 2020

and the amount of data collected. On the other hand, the required time for setting up a large-scale sensing campaign is prohibitively long. To this end, simulators offer a valid alternative to assess the performance of MCS systems in city-wide scenarios with large user participation in a reasonable time. Specifically, simulators are well-suited to assess and compare the performance of specific aspects of MCS systems (e.g., the decision process to sense and report data).

In this work, we focus on hybrid MCS and provide the researchers in the area with a fundamental tool to assess the performance of such systems. To this end, we extend CrowdSenSim 2.0 [11] by proposing an algorithm to efficiently re-route users that would offer opportunistic contribution towards the location of sensitive MCS tasks that require participatory-type of sensing contribution. This is achieved by including a new module in the simulator called *Path Changer*. CrowdSenSim 2.0 itself extends the architecture of CrowdSenSim [12]. From the original 1.0 version, we kept its core architecture and re-implemented almost integrally the simulator engine, besides several other improvements. As a matter of fact, the legacy version of CrowdSenSim can simulate with a high level of detail MCS systems in urban scenarios and assess the energy consumption of mobile devices. However, it lacks adaptation to several MCS sensing paradigms like Hybrid and other applications that require features such as statefulness and flexible event triggering. Indeed, as it will be explained in Section 3, the original CrowdSenSim featured only stateless use cases and was oriented to model network and energy consumption characteristics rather than algorithmic ones.

With respect to the original CrowdSenSim [12], CrowdSenSim 2.0 makes the following contributions:

1. It significantly improves the original platform by implementing a set of crucial features tailored to embrace a larger class of MCS algorithms and frameworks. In a nutshell, CrowdSenSim 2.0 supports stateful simulations (i.e., where the simulation events are chronologically dependent and the algorithm operation relies on such dependence), MGRS spatial encoding, a flexible time interval for event generation, and the integration of a new algorithm to determine user trajectories.
2. It optimizes the computational performance by means of a full code refactoring and the introduction of algorithm-level parallelism, which enables researchers to run several MCS algorithms simultaneously or several runs of the same algorithm at the same time.

Furthermore, besides the above contributions directly inherent to the simulation platform, the paper makes these additional contributions:

- We include a new module in CrowdSenSim 2.0 to support hybrid data collection. Such module enhances the original flow of the simulator by operating a significant change in terms of the applications supported.
- We validate the benefits CrowdSenSim 2.0 brings in terms of runtime execution and memory utilization. We also show the impact of simulation parameters, such as the city size, the number of simulated users and the number of path changes, in terms of time performance.
- We present two use cases: (*i*) an analysis of a stateful distributed data collection algorithm implemented in CrowdSenSim 2.0 and (*ii*) a new algorithm for hybrid data collection and its validation evaluated in three different cities.

The rest of the paper is structured as follows. Section 2 outlines the main research efforts in the area, Section 3 describes the simulator with particular focus on the improvements, Section 4 validates its computational performance in comparison to CrowdSenSim, Section 5 describes the distributed data collection algorithm that we integrated, implementation details of such algorithm, and its results. Section 6 describes a new algorithm for hybrid data collection and shows how our simulator provides support for such type of data collection. The results are validated for three different cities, i.e., Luxembourg, Bologna and Melbourne. Finally, Section 7 concludes the work.

2. Related works

After having scanned the state-of-the-art thoroughly, it has become evident that it does not exist a simulation tool that covers all the components of MCS. This is because MCS is a general paradigm which groups highly heterogeneous requirements, settings and objectives. For example, data collection can be opportunistic or participatory according to the degree of user involvement. Users might contribute freely to a sensing campaign or can be recruited through specific policies. The objectives of MCS research span over a number of areas, including quality and coverage of information over an area of interest, user recruitment, and incentive mechanisms [1,13]. Thus, such research areas typically propose optimization frameworks or algorithms that are evaluated standalone and often leave apart important components that impact on the correct modeling. These components include realistic user mobility [14] and modeling of urban environments [15] as well as modeling of the network that transfers sensing readings from end-users to the cloud where it is typically processed.

For the above reasons, a number of proposed simulation platforms are not suitable to properly evaluate MCS systems because they typically focus on one component at a time [16]. For example, in [17] the authors propose to leverage the capabilities of Network Simulator 3 (NS-3) to simulate ad-hoc scenarios for reporting incidents. NS-3 is a highly detailed simulation tool for networking purposes and models network protocols down to the granularity of the single packet across all the layers of the network stack. This strongly limits scalability, as the level of detail in such simulations is too high and modeling typical MCS sensing campaigns with thousands of users overall contributing during hours/days timescale becomes prohibitive. The same applies to similar simulation tools such as OMNeT++, used in [18]. CupCarbon, proposed in [19], is a WSN-based simulator in which the researcher can individually deploy both sensors and base stations on realistic urban environments obtained from OpenStreetMap (OSM).[2] Sensors can be mobile and can have dedicated paths along the roads, which makes it suitable for MCS scenarios. However, CupCarbon limits the size of the scenario, which precludes scalability to thousands of nodes. The most notable effort in the last years is given by CrowdSenSim [12], a simulator for MCS scenario capable of supporting a high number of users (order of hundreds of thousands) and their motion along the roads of cities imported by OSM without modeling in full the network stack, yet providing a sufficient level of detail on battery consumption statistics and number of tasks executed. The focus of the simulator is heavily energy-driven, implementing a number of algorithms, both in participatory and opportunistic scenarios, aiming to reduce the energy consumption per device. Being primarily implemented for energy consumption oriented scenarios, CrowdSenSim lacks adaptability to many MCS use cases as it does not support a number of features, such as statefulness, that are required by the majority of MCS systems.

3. The CrowdSenSim 2.0 architecture

This section presents the architecture of CrowdSenSim 2.0 by highlighting its novelties over the original CrowdSenSim. In particular, we detail the architecture of the simulator outlining the role of each module and we expose the new features and improvements. We specifically highlight the novelties that allow CrowdSenSim 2.0 to support hybrid CrowdSensing data collection, since such novelties bring a significant conceptual change to the whole architecture compared to the one presented in [11].

[2] https://www.openstreetmap.org/.

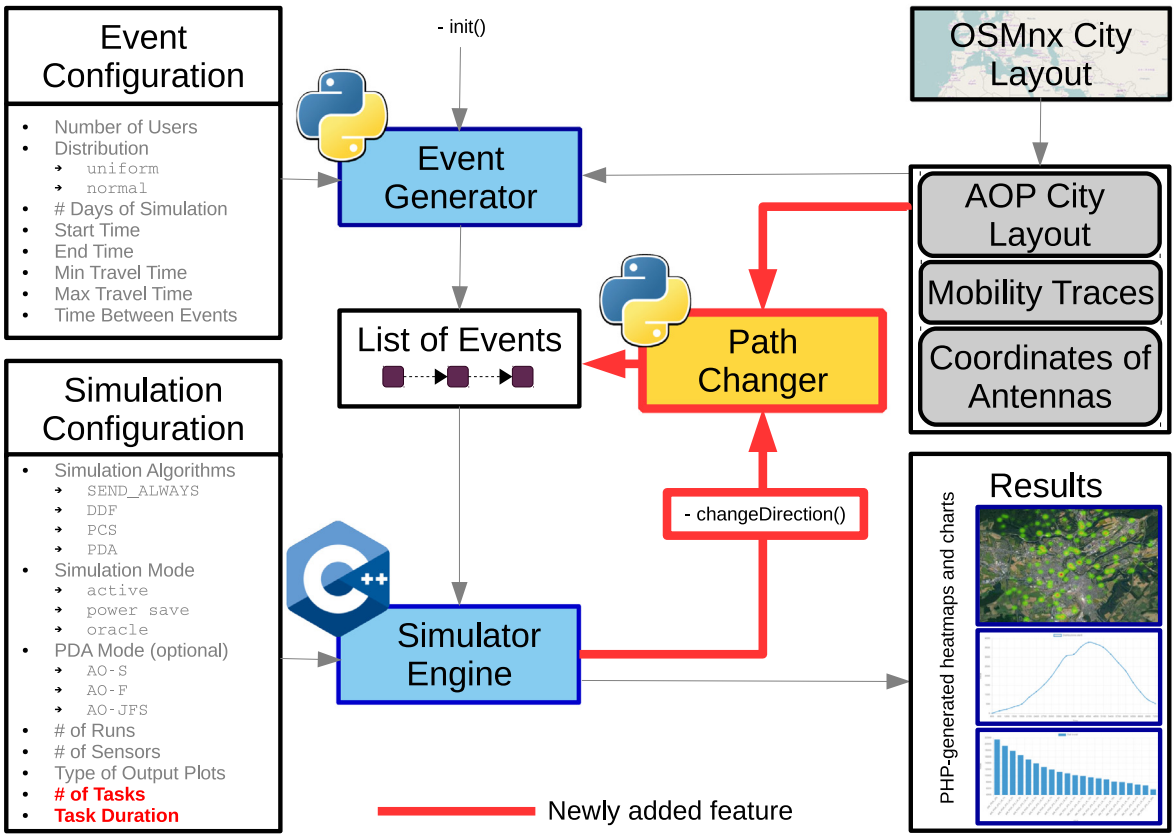


Fig. 1. Simulator modular architecture, including original features of CrowdSenSim and novelties of CrowdSenSim 2.0 in its dynamic version.

3.1. General architecture

Fig. 1 shows the architecture of CrowdSenSim 2.0, which includes major modifications and added features to the previous version of CrowdSenSim. In particular, the novel contributions include a stateful approach that is fundamental for specific classes of MCS applications, the support for a more flexible generation of events in terms of temporal granularity and other configurable parameters, the MGRS spatial encoding, the generation of highly-precise user trajectories and dynamic path change to support hybrid MCS scenarios. These features will be explained in detail throughout the section. Additionally, the code underwent a significant code refactoring and cleaning processes.

The simulator generates a set of participants moving within a street network, contributing data through the sensors of their mobile devices, and reporting it through the closest cellular base station or WiFi access point, according to the design of the MCS campaign. The *Event Generator* module consists of creating events, defined as "the arrival of a participant in a given location at a defined time". To this end, it takes in input the *City Layout*, the *User Mobility*, the *Coordinates of the Antennas* (that can include either/both deployed WiFi access points and LTE/4G base stations), and a set of parameters from the *Event Configuration file*. After such a macro step has been performed, the list of events is passed to the Simulator Engine, which defines the behavior of each participant upon each event.

The *Path Changer* module was developed from scratch and integrated with the system by implementing procedures that the other modules can call. Throughout the simulation, the *Path Changer module* allows users change their path at runtime, upon occurrence of certain conditions. This change alters the list of events, and once is finalized the *Simulator Engine* resumes its execution using the updates list. Both the *Simulator Engine* and the *Event Generator* have been integrally rewritten to make possible the integration of a set of necessary functionalities, which are explained thereafter.

3.2. City layout

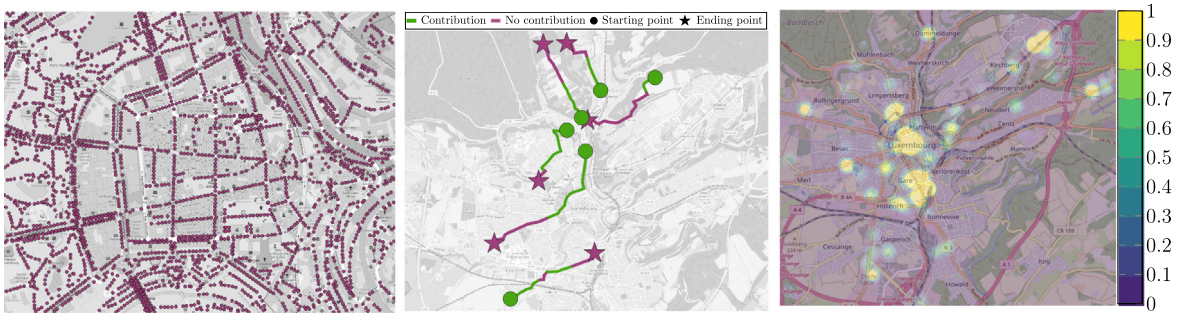
The City Layout module allows researchers to define the urban street network of a city-wide scenario over which the participants move. The street network is defined as a set of coordinates where pedestrians can be located, including *latitude*, *longitude*, and *altitude*.

3.2.1. High-precision street network design

While CrowdSenSim received as input a *.txt* file with a list of all coordinates to generate the city layout, CrowdSenSim 2.0 automatically gets the coordinates by exploiting OSMnx, an open-source Python package to download and simplify street networks from OSM [20]. The map thereby obtained is in a form of a graph describing the street network, in which each node corresponds roughly to a change of direction in a street or to a cross. We refer to such graph as the "OSM City Layout". Furthermore, CrowdSenSim 2.0 implements the AOP algorithm [21] to augment the precision of the OSMnx City Layout, with a granularity chosen according to the needs and the objectives of the MCS campaign under study. Fig. 2(a) shows the map of the city layout and the street network where users can walk. Pedestrian movement is generated over the points, which correspond to the set of downloaded coordinates. This second graph is defined as the "AOP City Layout"; it is much denser than the OSMnx one and much more suitable for generating fluidly mobility traces.

3.2.2. MGRS support

CrowdSenSim 2.0, in addition, supports Military Grid Reference System (MGRS) spatial encoding [22], which allows the developer to design data collection algorithms on top of such hierarchical spatial encoding. As a matter of fact, the usage of MGRS is crucial in data collection algorithms for MCS and several applications are built on top of it [23–25]. In particular, each event is generated along with its MGRS coordinates with the finest possible granularity (a 1 m-sided square



(a) AOP City Layout (b) User trajectories (c) Heatmap

Fig. 2. City layout, user trajectories and distribution of users connected to BSs in Luxembourg City. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

area) and such data is then passed to the Simulator Engine module for processing.

3.3. User mobility and event generation

The User Mobility module defines the initial user placement, at what time they "spawn" in the urban environment and how they move. Users are generated using a spatial distribution function and move according to different possible models. For instance, mobility can be uniformly or randomly distributed, can be based on real traces, or built upon different weights according to the point of interests and time of the day (e.g., following the distribution of Google Popular Times[3]). Each participant has a certain travel time (e.g., 20 min walk) and its trajectory is generated consequentially as a sequence of events, defined in Section 3.1, equally spaced in time. After an event is generated, the participant jumps to a location over the urban network topology reachable in a certain time given its walking speed, which is generated uniformly within the interval 1–1.5 m/s.

3.3.1. User trajectories

User mobility is generated as pedestrian trajectories with a random start and end point according to the walking period of each citizen over the street network of the chosen city. This feature allows highlighting the periods of active contribution of users along their paths according to the data collection framework (DCF) under analysis. For instance, Fig. 2(b) illustrates the trajectories of 5 participants walking in Luxembourg City and contributing with a Deterministic Distributed Framework (DDF) [26] in which users stop the sensing process after a certain amount of collected data. The circle represents the starting walking point and the star the ending point. The green path indicates when users contribute data, the purple one when they do not sense data.

Each user sends data to the closest cellular base station (BS) or WiFi access point (AP) according to the chosen communication technology. For instance, Fig. 2(c) shows the concentration of users connected to each BS in Luxembourg City at a given instance of time of the simulation runtime. The same concept is reused when a participant wants to change its path at runtime. In such case, the starting point of the newly generated path is the GPS coordinate where the participant is located, whereas the new destination is chosen by the participant itself (further details in Section 3.5).

3.3.2. Event configuration

In CrowdSenSim 2.0 many options in the generation of events have been made configurable (see the block "Event Configuration" in Fig. 1). The distribution function for generating users can be selected when configuring the simulation, e.g., a normal or uniform distribution, whereas in the old version users were generated only uniformly. By selecting among various generation functions, it is now possible to simulate different density of the users throughout the simulation runtime. The amount of time each user moves in the urban environment can be configured like in the original CrowdSenSim. The time interval Δt between two events has been made configurable as well, while in CrowdSenSim was fixed to 60 s. This enables to simulate possibly more complex scenarios, in which the time between updates is not decided at design time, but can change through configuration. This makes the number of supported applications significantly higher.

3.4. Simulator engine

The simulator engine is written in C++ and, as shown in Fig. 1, it takes in input a list of events, corresponding to the time in which users perform an action. In turn, each event triggers the sampling of each sensor as well as the communication module of each device because the Simulator Engine implements an apposite callback function. In practice, the Simulator Engine defines the behavior of each participant by implementing the action performed upon each event.

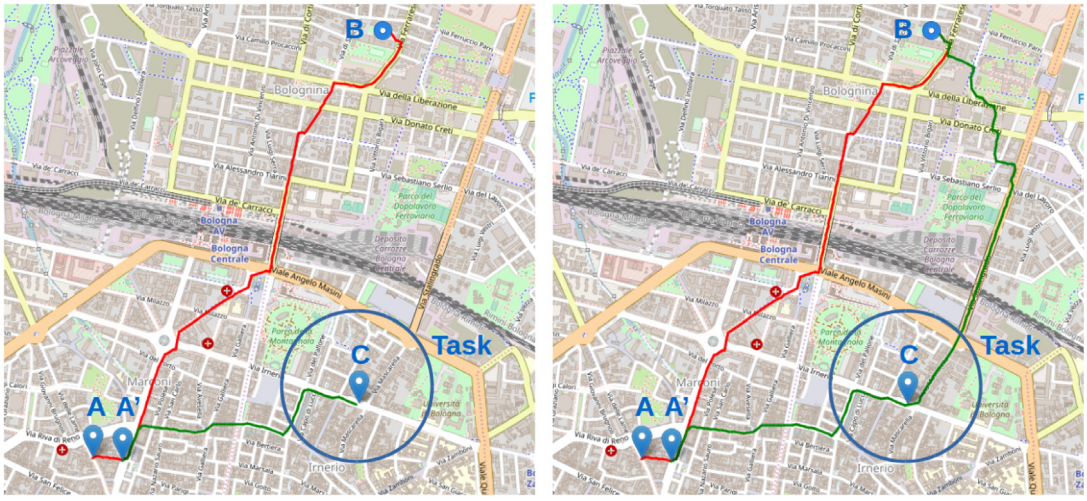
3.4.1. Global statefulness

CrowdSenSim 2.0 executes the events in absolute chronological order. To make the present CrowdSenSim 2.0 simulator more oriented to the algorithmic side rather than energy consumption of MCS systems, we implemented the Simulator Engine in a way in which events are ordered chronologically, whereas, in the previous version, events were executed per-participant – i.e., all the events of the first participant were executed before all the events of the second, and so on – making the implementations of certain algorithms unfeasible. In fact, the original CrowdSenSim could implement any stateless algorithms as well as any algorithm requiring only local statefulness, i.e., a state maintained only internally by each participant and does not interact with other participants in any case. With the novel version of the simulator presented in this paper, CrowdSenSim 2.0, we can implement any algorithm with global statefulness, i.e., the state is maintained by each participant as well as a central entity, and each event can be influenced by the past ones. Note that this increases the expressiveness of the simulator without preventing data collection algorithms that were implementable in the past version to be also implemented in CrowdSenSim 2.0.

3.4.2. Algorithm-level data collection parallelism

Comparing the performance of multiple algorithms (e.g., for data collection) at a time is often a desirable feature in simulation platforms. CrowdSenSim 2.0 makes it possible to run different algorithms within the same run, simply by defining more than one callback function relative to each event. More in detail, any time the Simulator Engine

[3] https://support.google.com/business/answer/2721884.



(a) Type 1: Change of Direction (b) Type 2: Deviation

Fig. 3. Examples of the two different types of path change performed by a participant originally going from A to B and, when in A′, decides to head to a new direction C in order to perform a task. The example takes place in the Italian city of Bologna.

is triggered upon the occurrence of an event, it is possible to specify more than one function to be called separately. In this way, it is possible to define a number of algorithms to run at the same time, that will output their results separately just as if they were separate runs. Obviously, one can also run the same algorithm several times in parallel with a different random seed. Such advance boosts the performance of the simulator significantly, as in a single run several algorithms can be tested at the same time. We show performance evaluation of this aspect in Section 4. As shown in Fig. 1, parameters such as the algorithms used as well as the number of runs can be specified in the Simulation Configuration file. The simulator implements DDF, PCS, and PDA algorithms as in [27], in particular, PDA has been re-designed in its global stateful version [28] (Section 5). Note that, in order for the algorithmic-level parallelism to take place, all the different simulation configurations must observe the same list of events, which means that the mobility of each user must be unaltered.

3.4.3. Participant awareness

To enable applications and algorithms that require privacy or fine-grained energy savings mechanisms, CrowdSenSim 2.0 allows the simulated users to be aware of certain information detained by the central entity organizing the sensing campaign. For example, instructions about the amount of yet to be delivered information in a certain area. Therefore, users can be in:

- **Power-save mode**, thus eligible to receive such information when it is piggybacked on another communication (e.g., when they are pushing data).
- **Active mode**, thus eligible to receive such information upon each of their events occur.
- **Oracle mode**, thus aware of such information at any time.

When looking at such division from a privacy perspective, users in power-save mode are those with limited access to global information because they are not trustworthy, whereas users in oracle mode have higher privileges. Clearly, the additional state may be implemented in case the simulated scenario requires it.

3.5. Path changer

The newly added feature for CrowdSenSim 2.0 is the Path Changer. This additional module allows the participants to alter their trajectory at runtime upon the occurrence of certain events. The new module significantly amplifies the potential of CrowdSenSim 2.0 to support reactive application and participatory data collection. The main novelty lies in the fact that the simulated mobile users are enforced with the capability of actively performing decisions that contribute in the MCS campaign, such as moving towards a localized task upon occurrence of a specific event like an accident. This comes at the price of breaking the usual "waterfall-like" flow of CrowdSenSim 2.0, in which the list of events is pre-determined. Instead, as it can be observed in Fig. 1, the execution can revert to the modification of the list of events by the Path Changer and the new list has to be passed again to the Simulator Engine.

In detail, the Simulator Engine implements a `changeDestination()` function, which can be invoked by a participant by setting a new GPS coordinate as a destination. This is translated into a Python object through embedding [29], which then invokes the dedicated function in the Path Changer module. The latter elaborates the path by using the single source Dijkstra algorithm over the AOP City Layout graph, by setting the source and the destination to their closest counterparts in the graph. Alternatively, the OSMnx graph can be used, which allows for faster execution (the graph is much coarser) at the price of a higher approximation. The newly generated list of events is passed on to the Simulator Engine, which then deletes all the events belonging to the calling participant that are yet to be processed and replaces them with the new list, respecting the chronological order of the total list of events. Such operation is performed in linear time, as all the newly generated events as well as the original list remain in chronological order.

Fig. 3(b) shows the two types of path change that are implemented:

- Type 1, or simply "change of direction" allows for each participant to pick a new destination.
- Type 2, or simply "deviation" allows for the user to pick a point of interest and immediately change its path to first get there and then, once such point has been reached, head to the original destination. This results in performing the single-source Dijkstra algorithm twice, however only one call to the Path Change module is performed.

As the use of this module changes the original list of events, the boost given by the algorithm level parallelism cannot be used for such type of simulations.

4. Performance evaluation

The newly developed CrowdSenSim 2.0 has undergone a complete code refactoring procedure as detailed in Section 3. Such operation has been extremely delicate as we needed to assure that every feature of the original CrowdSenSim was left intact. In other words, the novel simulator is retro compatible with the previous implementation, to allow reproduce results regardless of the simulator version used. In this section, we show that both CrowdSenSim and CrowdSenSim 2.0 exhibit the same behavior on common use cases and we highlight the benefits in terms of RAM utilization that the new version brings. Moreover we quantify the performance of the newly introduced *Path Changer* module along with scalability tests that show how the simulator performs by changing its parameters.

4.1. Validation of CrowdSenSim 2.0 over CrowdSenSim

Both the instances of CrowdSenSim and CrowdSenSim 2.0 that we used throughout the paper have been launched on a virtual machine using 1 core of the host machine with 4 GB of dedicated RAM and running Ubuntu 16.04.6 LTS. The host machine is an AMD Ryzen 5 1600 at 3.2 GHz (6 core, 12 thread) with 16 GB RAM and running Windows 10 Pro 1809.

In order to efficiently validate CrowdSenSim 2.0, we referred to an energy consumption analysis of the DDF data collection algorithm originally proposed in [26] that was implemented and practically evaluated in [27]. DDF is a locally stateful data collection algorithm in which participants keep on generating data up to a certain threshold of energy consumption depending on their battery capacity. For the sake of energy-related analysis, we left the energy calculation of the original CrowdSenSim untouched. In detail, we equipped each participant with a mobile device carrying an accelerometer, a pressure sensor, and a temperature sensor. As in [12], the sensors generate readings with the same sampling frequency. For all the simulations, we resort to 10 000 participants in the center of Luxembourg City, which covers an area of 51.73 km^2 with a perimeter of 52.5 km and a population of 119 214 inhabitants as of the end of 2018. The simulation has a duration of 12-h (starting at 12:00 PM and ending at 11:59 PM) and paths are generated with a duration uniformly distributed between 20 min and 40 min.

We fed both CrowdSenSim and CrowdSenSim 2.0 with the same set of events, which are sampled per-participant with a frequency of 60 s.

We ran extensive simulations and measured the current drain of the device of each participant in relation to the sensing and reporting activity, and we plot the results in Fig. 4. Devices use the WiFi technology for communication as in [12]: a number of WiFi hotspots are deployed in the area of interest, and every time a participant needs to transmit it sends data through the closest AP in the map. We observed that both CrowdSenSim and CrowdSenSim 2.0 generate the exact same values, which proves their equivalence in terms of results output.

In order to further strengthen our claim, we also compared the transmission consumption between the two simulators. As we did for the sensors, we did not modify the way in which participants transmit data. Again the values produced by the two simulators match perfectly, validating their equivalence.

4.2. Performance analysis of runtime execution and memory utilization

The code refactoring and the parallel processing feature brought a significant boost in the simulation performance in terms of the time of execution and memory consumption.

Fig. 5 shows the simulator runtime in seconds on top of the number of algorithms run. Even in one single run, CrowdSenSim 2.0 achieves a lower run time than CrowdSenSim (7 s in average against 8 s). Furthermore, as CrowdSenSim 2.0 allows for multiple algorithms to run in parallel — or multiple runs of the same algorithm, the time execution remains almost constant whereas in the original CrowdSenSim it scales linearly. Indeed, to perform multiple runs, the original CrowdSenSim needs to be launched several times sequentially.

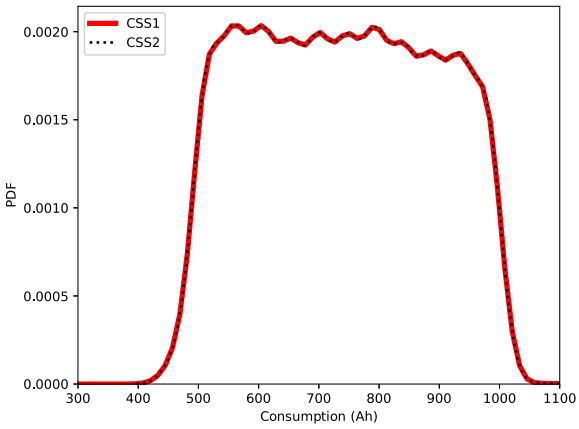


Fig. 4. Density of the energy consumed by participants with CrowdSenSim and CrowdSenSim 2.0.

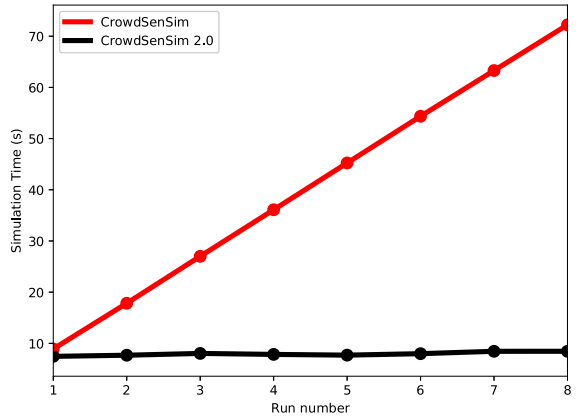


Fig. 5. Runtime execution of DDF with multiple runs.

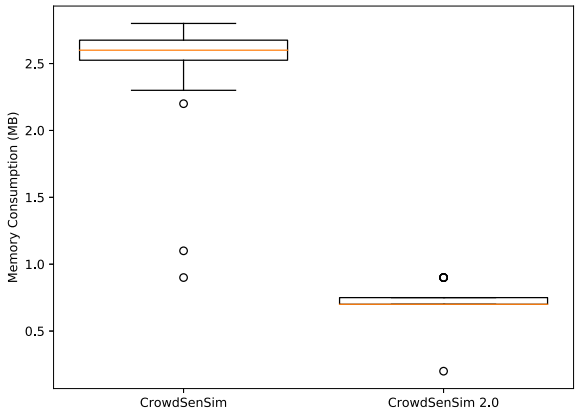


Fig. 6. Performance of CrowdSenSim and CrowdSenSim 2.0 in terms of RAM consumption.

Fig. 6 shows in boxplot form the performance of CrowdSenSim and CrowdSenSim 2.0 in terms of RAM consumption. For a fair comparison, we used only one algorithm at a time. CrowdSenSim 2.0 outperforms CrowdSenSim significantly. This is due to several code optimizations, such as the use of integers as identifiers instead of strings.

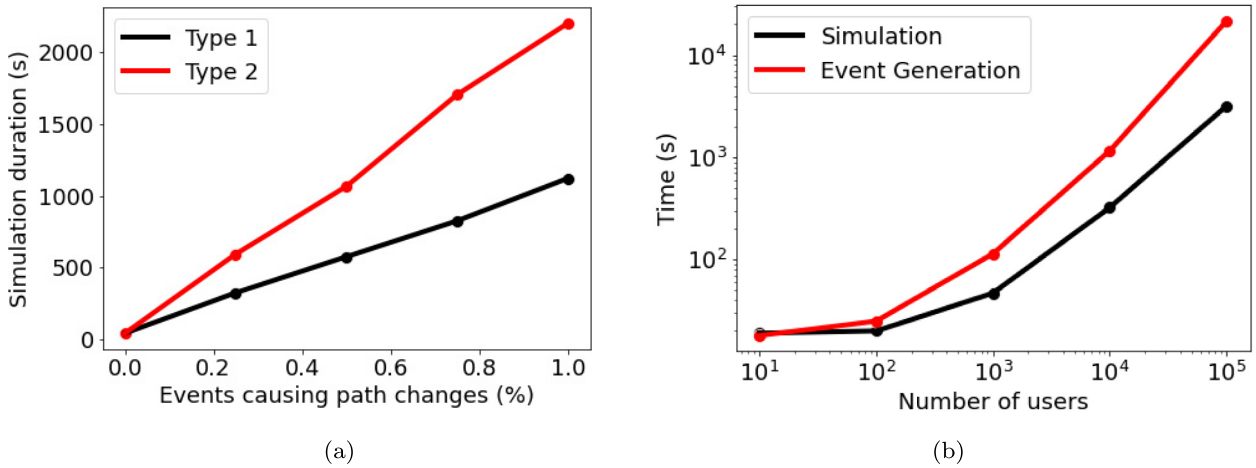


Fig. 7. Performance of the Path Changer. In (a), plot of simulation time by changing the probability of event changes. In (b), plot of the time needed by the event generation phase and simulation time for the same city (Luxembourg city) while varying the total number of users in the simulation.

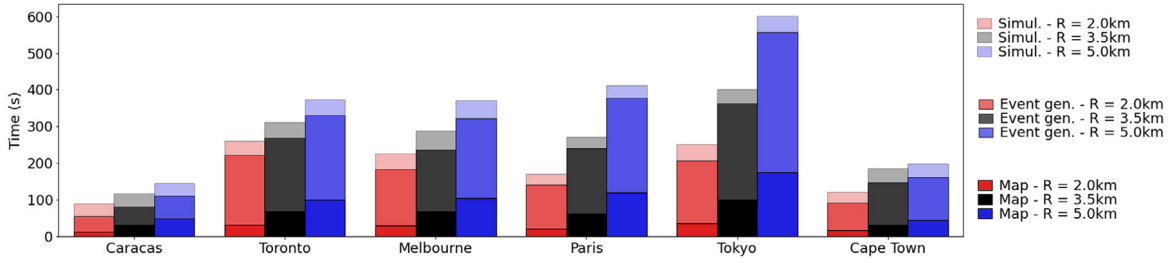


Fig. 8. Map building, Event generation and Simulation times for 6 cities in different continents.

Table 1
General information about the cities taken into account for the scalability test.

	Caracas	Toronto	Melbourne	Paris	Tokyo	Cape town
Surface (km^2)	4715	5906	9943	17,174	14,034	400
Population (in millions)	~2.25	~6.42	~5.08	~12.53	~38.14	~4.01

4.3. Performance analysis of path changer and scalability tests

The benefit of supporting hybrid CrowdSensing scenarios comes at the cost of slowing down the whole simulation process. Indeed, the *Path Changer* module breaks the usual simulator flow and requires a communication pipe between a C++ module and a Python module as well as individual re-instantiation of shortest path calculations. We now quantify the execution delay in the presence of a massive amount of path changes.

The test is performed in the city of Luxembourg, the number of participants is lowered to 1000 and the duration of the simulation is reduced to 2 h. The sampling frequency is tuned to 10 s, therefore bringing the total amount of events above 150 000. We ran several simulations in which participants change direction with a given probability for each event. The destination of the path change is kept the same as the original, as defining a new one would not be meaningful to a performance evaluation test. The shortest path calculation is executed through the use of the OSMnx City Layout, which has 35,548 nodes.

Fig. 7(a) shows the obtained time execution performance. On the x-axis the probability for each event of generating a path change spans from 0% (no change) to a maximum of 1%, which translates in roughly 1500 changes. The two curves represent the time in seconds taken for executing a full simulation either with a Type 1 or a Type 2 path change. As the Type 2 path change almost doubles the execution time, we can state that a large part of the execution is devoted to the shortest path invocation (as no real change happens in the mobility). We also observe that a high number of Type 2 path changes pushes the simulation time to less than 40 min, which is well below the simulated time, also bearing in mind that such a high number of changes is unlikely to happen in a realistic scenario.

In order to complement the performance test shown in Fig. 7(a), we ran further experiments to show the scalability of the simulator, specifically quantifying its behavior by changing (i) the number of users and (ii) the size of the map. Fig. 7(b) shows the performance in time of the simulator by varying the number of users. We used the same configuration as above: the simulation takes place in the city of Luxembourg, we fixed the number of Type 1 path changes to 25, the duration of the simulation to 2 h and increase the number of users by a factor 10 starting from a total of 10 to 100,000. The axes in the figure are represented in logarithmic scale and show the time taken by the simulator to generate the events as well as the duration of the actual simulation. Both of them display a linear increase, which is easily deducible by the architecture of the simulator. Fig. 8 shows instead the performance of the simulator by changing the size and the structure of the city taken into account. With respect to the previous tests, we still kept the number of path changes fixed to 25 and the number of users to 1000, while we changed the city and the radius of the map extracted from its center. In particular, we used one representative city for each continent and ran simulations over areas with a radius of 2.0 km, 3.5 km and 5.0 km. Table 1 reports the actual size of the metropolitan area of each city and its population as of the latest reports available. For a fair comparison, we extracted the following features: the time needed for downloading the map and running the AOP algorithm (the "Map" time), the time for computing the generation of the events (the "Events generation" time) and the actual simulation time (the "Simulation" time). These are stacked in bars in the figure, quantifying also the total time taken. In general, we can observe how enlarging the area increases mostly the map time with a linear trend, the event generation time is minimally affected and the simulation time is almost the same. Some of

the cities, like Toronto and Melbourne, tend to present a small increase in duration when extending the radius from 2.0 km to 3.5 km compared to its absolute duration when the radius is 2.0 km. This happens even though the area considered is more than doubled, reason being most of the streets (and, consequently, nodes) are grouped in the very center, which is far denser than the outskirts. This difference is less noticeable in cities like Caracas and Tokyo.

5. Case study: a stateful distributed opportunistic algorithm

In this section, we outline our Asymptotic Opportunistic algorithm for Joint Fairness and Satisfaction index (AO-JFS) that was initially proposed in [28] along with its simplified versions AO-F and AO-S. Both variants are implemented in CrowdSenSim 2.0 as members of a family of algorithms called Probabilistic Distributed Algorithms (PDA) [27]. Originally, performance evaluation was conducted using an ad-hoc simulator. The algorithm has been designed for data collection control, that is, preventing the whole scenario from generating too much or excessively less data. Too less data would result in a poor mapping of a phenomenon on an urban (or rural) environment, whereas too much data may result in too much noise to get rid of as well as an unbearable amount of users to reward for data that is much more than required and, consequently, an unnecessary energy consumption. We model such scenario in a push-based and totally anonymous distributed algorithm in which the central entity has no direct control over the single users – i.e., it cannot specifically query users about certain resources – neither it has knowledge of where the users are and how many of them are contributing. Participants can contribute pushing their data at a defined frequency to the central entity, which will only reply with a Satisfaction Index (SI), a number that resembles how much the server is "satisfied" about the number of observations received in a defined time window about a resource. Consequentially, participants decide with a probabilistic distribution whether to contribute or not at the following time slot on top of the received SI: if the satisfaction is low, they are pushed to contribute more, if it is high, their contribution is discouraged.

A similar version of such algorithm was implemented in the original CrowdSenSim [27] and called PDA, although CrowdSenSim 2.0 for its stateful approach would have been required to assess AO-JFS properly. Indeed, AO-JFS requires the central entity to be aware of all data delivered by the participants at each instant of time in order to correctly calculate the SI. With the original CrowdSenSim, each participant executes all its events completely before the events of another participant can take place. Hence, the chronological order of the events is not enforced. Therefore, CrowdSenSim 2.0 with its stateful approach is necessary for any MCS algorithm that heavily depends on the chronological sequence of the events. In the rest of the section, we will outline in detail the behavior of AO-JFS.

5.1. AO-JFS core idea

We can model the problem as N different stations (we use the terms stations, users, and participants interchangeably) that adhere to the MCS campaign and perform observations against a given phenomenon. Such number N can vary over time due to mobility, in particular, participants may leave the interested area, whereas new ones may join it. We assume to split our timeline in time slices Δt_i, that represent the atomic units during which a station cannot transmit more than once due to internal clocks. We also assume that the stations will send observations relative to ν certain resources $\Psi_0, \dots, \Psi_\nu$ periodically. The central entity's goal is to obtain exactly M_j observations about Ψ_j for $j \in [0, \nu]$ within every time window T_i, the length of which is given by $|T| = w$. We follow the approach of the sliding window, thus $T_i = \{\Delta t_{i-w}, \dots, \Delta t_i\}$, this means that T_i and T_{i-1} are overlapping by $w-1$ time slots. The central entity displays the performances of the data collection process through the above-cited SI. Such value is calculated upon each time window T_i and it is defined as $SI_{i,j} = \frac{m_{i,j}}{M_j}$, where $m_{i,j}$ is the actual number of observations received by the central entity within T_i for the resource Ψ_j. The aim of the central entity is to obtain a SI equal to 1 for each resource.

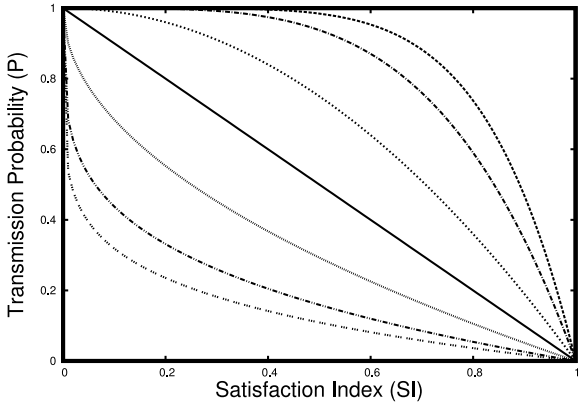


Fig. 9. Base probability plus different booster values.

We assume that each participant knows the SI values at all times (i.e., the central entity broadcasts the SI constantly) and, for each atomic time slot, performs a decision of whether to send or not the local measurement of the sensor (for each resource). In detail, at the time Δt_{i+1}, each participant calculates a probability of sending the measurement relative to the sensor Ψ_j that is basically the inverse of the received $SI_{i,j}$, therefore $P_{i,j} = 1 - CSI_{i-1,j}$ with CSI_i, j being the Constrained Satisfaction Index:

$$CSI_{i,j} = \begin{cases} \epsilon & \text{if } SI_{i,j} < \epsilon, \\ 1-\epsilon & \text{if } SI_{i,j} > 1-\epsilon, \\ SI_{i,j} & \text{otherwise,} \end{cases} \tag{1}$$

with ϵ being a very small number (in our case 0.001). This forces the SI to range from a very small number close to 0 to a number close to 1 for the purpose of probability calculation.

5.2. Boosting runtime execution

To prevent contributions to stabilize at a too low or too high SI, we introduced boosters for the probability calculation. We define a booster as an exponent E to which we elevate the CSI in the probability calculation: $P_{i,j} = 1 - CSI^E_{i-1,j}$. Fig. 9 shows the probability curve for different values of E (the central straight line is obviously for $E = 1$). Specifically, we introduce an overall booster value b_j and an individual value k_j per sensor.

Suppose the aim is to maintain SI in the range $SI \in [\sqcap_{SI}; \sqcup_{SI}]$, where the bounds are, for example, set as 0.95 and 1.15. Then $\forall i, j$, if $SI_{i,j} > \sqcap_{SI}$ then $b_j = dec(b_j)$, whereas if $SI_{i,j} < \sqcup_{SI}$ then $b_j = inc(b_j)$ with:

$$inc(b) = \begin{cases} \frac{1}{(1/b)-1} & \text{if } b < 1, \\ b+1 & \text{otherwise.} \end{cases} \tag{2}$$

$$dec(b) = \begin{cases} b-1 & \text{if } b > 1, \\ \frac{1}{(1/b)+1} & \text{otherwise.} \end{cases} \tag{3}$$

k is the attempting factor, calculated individually by each station on top of the received b as

$$k_j = \begin{cases} \lfloor \log_2(\eta) \rfloor & \text{if } b_j < 1, \\ \eta \cdot b_j & \text{if } b_j \geq 1, \end{cases} \tag{4}$$

where $\eta > 0$ is the number of Δt_i slots elapsed since the last transmission. In the end, the probability is calculated as

$$P_{i,j} = 1 - CSI^{inc^k(b)}_{i-1,j}. \tag{5}$$

Note that the term $f^n(x)$ indicates the iterative composition as $f^n(x) = f \circ f^{n-1}(x)$.

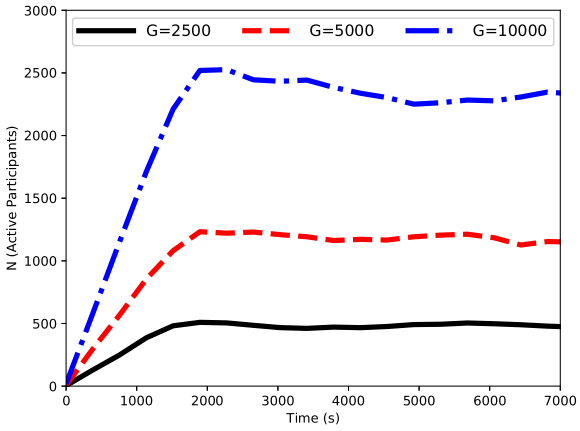


Fig. 10. Number of active users N over time for 3 different values of generated users G ($G = 2500$, $G = 5000$, $G = 10\,000$).

5.3. Implementation and testing of AO-JFS

We outline how to integrate the AO-JFS algorithm (Section 5) with CrowdSenSim 2.0. AO-JFS is a distributed algorithm where the active part is given by the participants and the central entity is only "reactive". In more details, each event generated by the event generator on the map triggers an action by a participant. Since events are processed in chronological order, each of them depends on the sequence of events previously occurred. Thus CrowdSenSim 2.0 can be employed for its analysis. We implemented AO-JFS only in Active Mode, which means that each participant receives information about the status of the SI once every time slot Δt. Upon the occurrence of an event, each participant at time slot t_i (for each sensor j, with $j \in [0, \nu]$):

1. Retrieves the known value of the $SI_{i,j}$ and transform it to a $CSI_{i,j}$ for the purpose of the probability calculation using Eq. (1).
2. Retrieves the known value of the global booster b.
3. Sets the local attempting factor k using Eq. (4).
4. Calculates the actual probability $P_{i,j}$ to send the observation for the resource Ψ_j using Eq. (5).

Therefore, events for an AO-JFS simulation are merely the instants in which a participant decides whether to send observations or not. When all the events for t_i occurred, then the values of the SI are updated before taking into account the next time slot. The Path Changer module is not used in this algorithm, as users are contributing opportunistically and do not actively perform decisions. On the contrary, we make use of the algorithm-level parallelism in this use case.

Our implementation of AO-JFS is evaluated for 2 h long simulations in the center of Luxembourg City. Parallelism is used to test 50 runs of AO-JFS at the same time, each of them using a different random seed. Δt is set to 10 s and $w = 30$, therefore the time window T is 5 min. We set $\nu = 3$, in particular, we used the three sensors mentioned in Section 4 as our resources, and fixed the desired amount of observations as $M_1 = 7500, M_2 = 5000, M_3 = 2500$. We generated users using a uniform distribution and set the total number G as 2500, 5000 and 10 000. Fig. 10 shows the number of active users N over time. As the distribution is uniform, N tends to reach a steady state after an initial transient.

Fig. 11 shows the results of the simulations. In particular, the values of the SI for the three sensors Ψ_1, Ψ_2, and Ψ_3 are shown for each configuration in the form of a Probability Distribution Function (PDF) in which the data points are the value of the SI sampled each Δt. For each sensor, the SI value stabilizes around 1, which is the goal of AO-JFS. In more details, Fig. 11(b) shows the behavior of the SI at $G = 5000$, with the number of active users over time N floating around 1 200. This number is in a good balance with the number of required observations, in fact the SI value for all the sensors tends to cluster almost regularly around 1. Fig. 11(c) shows the behavior of the SI with $G = 10\,000$, with N settling around 2300. This number is very high in comparison to the number of required measurements. Therefore the effort of AO-JFS is in limiting the number of contributions by the participants. This is even more evident for Ψ_3, the resource for which the fewer contributions are needed, as the respective SI values tend to cluster at a slightly higher value (i.e., around 1.1). On the other hand, Fig. 11(a) shows the SI for $G = 2500$ and, consequently, N floating around 500, which is a low number in comparison to the contribution required. Although the behavior of the SI is quite similar to the other plots, we can appreciate a slight difference for Ψ_3, as its values are more spread. This is due to b being quite high: as the participants are pushed to contribute more, the probability curve gets very high for most of the CSI values in input and causes more likely peaks and troughs in the contribution over time.

6. Case study: a stateful task-based hybrid algorithm

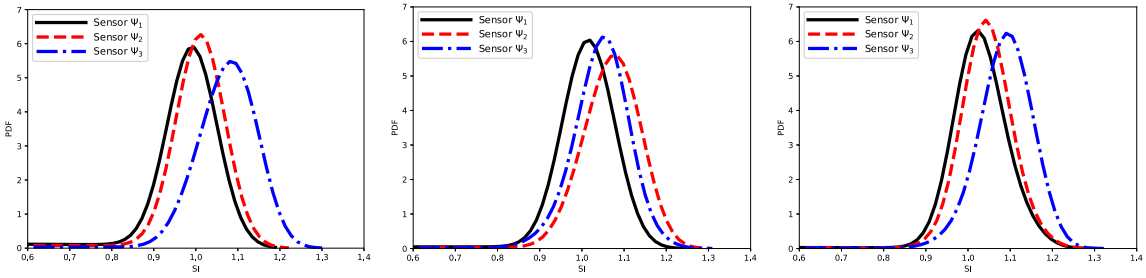
This section shows the potential of CrowdSenSim 2.0 in supporting hybrid data collection campaigns by implementing a policy that allows to both capture a phenomenon through opportunistic data collection as well as sporadic events through participatory data collection. Without loss of generality, in this section we term tasks the latter and assume that opportunistic data collection occurs in the background through AO-JFS. We implement hybrid collection by allowing the users to be notified by some important events occurred in the vicinity and allow them to change their mind at runtime by deviating from the initial trajectory to move close to the location of the event. Please note that acceptance to deviate from the original trajectory is expressed probabilistically to capture the fact that only part of the population could be willing to do so. We call such hybrid scheme HTA (Hybrid Task Accomplishment).

6.1. HTA core idea

The scenario works as follows: we assume an urban scenario in which users walk along their pre-defined routes. We also assume that each of such users is provided with a mobile application able to notify them whenever a task occurs – for simplicity, tasks are assumed to be sudden events that cannot be inferred beforehand – together with their location and their duration in time. In order to perform a task, it is sufficient for a participant to be at some point in the coverage area of the task, without the constraint of lingering there for a minimum amount of time. We make this assumption as it does not weakens our evaluation and realistically it is enough for the user to perform the action needed, e.g., to shoot a picture.

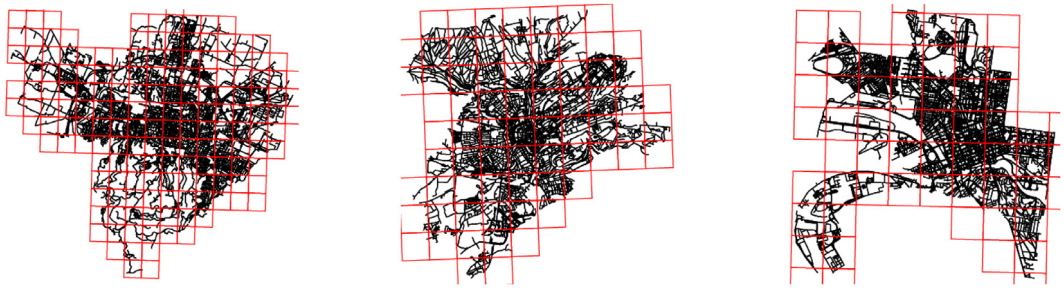
When a task appears all the users in the city are notified of its presence and can freely decide whether to go and perform it, which translates in deviating from their path in order to cross the area of the task. Mathematically, a task is a tuple $T_i = \langle lat_i, lon_i, r_i, start_i, end_i \rangle$, where lat is its latitude, lon is its longitude, r is the radius of the area pertaining the task, $start$ is the starting time and end is its ending time. A participant can be represented in each moment as $P_j = \langle lat_j, lon_j, lat_f, lon_f \rangle$, where lat_j, lon_j are its position and lat_f, lon_f are its destination. When the task is generated the user is notified and, assuming that lat'_i, lon'_i are the coordinates of the point closest to the participant within the area of the task, then we model the probability of the participant to go there (and then go to its original destination) as inversely proportional to the amount of time spent to perform the task in addition to its normal path. Formally, we define the function $E(lat_1, lon_1, lat_2, lon_2)$ to output the estimated time to go from the location (lat_1, lon_1) to (lat_2, lon_2). Then we model the probability P for the participant P_j to perform task T_i as:

$$P = 1 - \frac{E(lat_j, lon_j, lat'_i, lon'_i) + E(lat'_i, lon'_i, lat_f, lon_f) - E(lat_j, lon_j, lat_f, lon_f)}{T_{MAX}}, \quad (6)$$



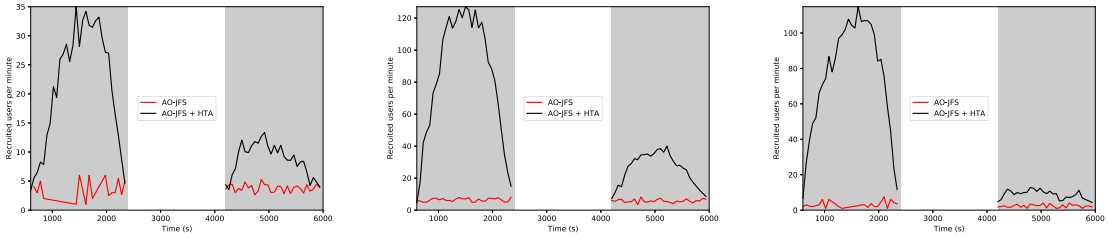
(a) $G = 2\,500$ (b) $G = 5\,000$ (c) $G = 10\,000$

Fig. 11. PDF of the SI for the three sensors for different values of G.



(a) Bologna (b) Luxembourg City (c) Melbourne CBD

Fig. 12. Street networks used for the evaluation of HTA, together with the shapes of the 1 km-sided MGRS squares.



(a) Bologna (b) Luxembourg City (c) Melbourne CBD

Fig. 13. Number of users in the task area per minute over the simulation. The task on the left is close to the city center, while the task on the right is in the outskirts.

where T_{MAX} is the maximum amount of time that a person would realistically spend to perform a task. The calculation of such probability is obviously subject to

$$E(lat_j, lon_j, lat'_i, lon'_i) \leq t - end_i, \tag{7}$$

where t is the current time. We intentionally formulate P is fairly simple way. Indeed, modeling as realistically as possible the user behavior is beyond the scope of this paper as it would require sociologically studies. Our point is rather different: we aim to prove that CrowdSenSim 2.0 can support the simulation of such scenarios and we let the adopters of our simulator to specifically define a correct mathematical formulation in line with the problem definition.

6.2. Implementation and testing of HTA

We implemented HTA in CrowdSenSim 2.0 by modifying the definition of a task to be MGRS-based (which provides simultaneously the notion of location and area), therefore $T_i = \langle MGRS_i, start_i, end_i \rangle$ and, for simplicity, participants calculate their distance to the center of the MGRS square. The predicted time calculated by the function $E()$ has been implemented by interfacing the simulator with a local deployment of OSRM [30], which is a REST-based service that takes input a pair of coordinates through an HTTP GET request and outputs the time and

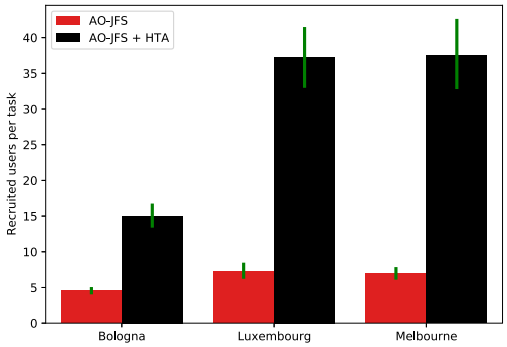
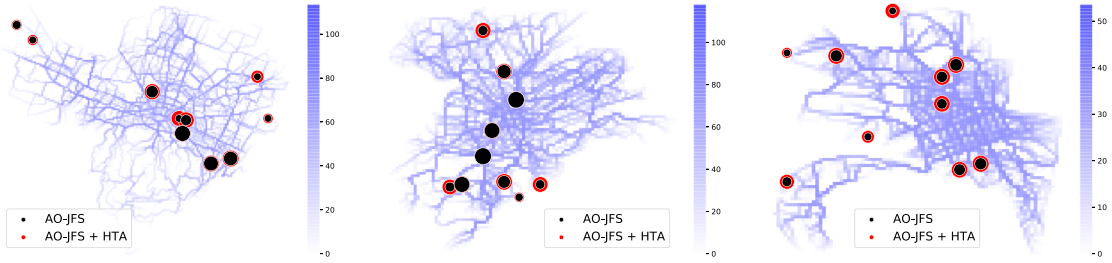


Fig. 14. Bar plots on the number of users in an MGRS square containing a task while the task is active, averaged over the number of tasks.



(a) Bologna (b) Luxembourg City (c) Melbourne CBD

Fig. 15. Heatmaps showing the density of the users, sampled by 100 m-sided MGRS squares, and the accomplishment of the tasks through the sizes of red (with HTA) and black (without HTA) circles. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

the distance to get from the first point to the second (optionally gives all the streets crossed through the path). In order to evaluate the HTA algorithm, in line with the evaluation of AO-JFS (Section 5.3), we ran 2-h long simulations, this time in different cities, as the street network of differently shaped cities may affect significantly features related to mobility such as rerouting [31]. In particular, we chose, alongside with the already cited city center of Luxembourg 12(b), the city centers of Bologna 12(a) and Melbourne 12(c).

Besides presenting the street networks employed, Fig. 12 also shows the shapes of all the 1 km-sided MGRS squares in the area, which gives also a flavor of its size. All the simulations feature 10 000 participants running by default AO-JFS with the same parameters specified in Section 5.3. In addition, each user can run HTA on top of AO-JFS and, therefore, make use of the Path Changer module described in Section 3.5. For HTA, we set T_{MAX} to 30 min and deployed randomly a set of tasks throughout the simulation. Each task is assigned to the center of a 100 m-sided MGRS square, has a duration of 30 min and is generated randomly once for each city.

For each set of tasks we performed 20 simulations (without algorithm-level parallelism) with and without HTA to evaluate the level of task accomplishment, i.e., how many users have contributed data for it. We performed two sets of simulations for each city, one with 2 tasks and the other with 10 tasks.

Fig. 13 shows for the first set the results for each isolated task where each plot depicts the behavior of the two tasks. The first one is generated at the beginning of the simulation close to the city center, the other towards the end of the simulation and in the outskirts. The tasks are enough distant in time to let the simulation revert back to a steady state between the end of the first and the beginning of the second. We compare the number of users performing the task within the MGRS area coverage at a minute-level granularity. As expected, HTA allows for a higher number of participants per task with respect to the solely opportunistic approach AO-JFS. We note that three cities share a similar behavior: tasks in the city center have higher chances of augmenting the participation through HTA than tasks in the periphery. Besides the actual scale, the trend is interesting because the three cities have a different urban tissue which suggests that the re-direction principle could be re-used effectively across different cities.

We notice similar results in Fig. 14. The plot is the result of a simulation with 10 tasks randomly over the three cities and the individual result is the average amount of participants within MGRS task coverage for the participatory tasks with and without HTA. The bars show the mean and the standard deviation values over 20 simulations. The trend for Luxembourg City and Bologna is quite similar compared to their size, whereas Melbourne shows lower numbers. This happens because the whole city center of Melbourne is in a grid-shape and streets are spread quite evenly, causing users to be more distributed and less likely to be close to a task.

Fig. 15 shows further results about this set of simulations with geographical overlays over the three different cities. Specifically, we colored in a scale from white to blue all the 100 m-sided MGRS squares according to the number of users within such square that contribute to the opportunistic sensing campaign. The results are averaged by the number of simulations and, since at this level of representation differences are almost unnoticeable, we also averaged the results between the configurations with and without HTA to show how users are distributed at a macro-level in the street networks. The second overlay shows the tasks depicted as circles, i.e., the center of the 100 m-sided MGRS square that provides the coverage area of the task. The size of the circle corresponds to the number of users that contributed to the task when it was active (red and black for HTA-enabled and HTA-disabled respectively). Obviously, red circles are always equal or bigger than the black circles (where circles are equal, the red one is not visible). The figure shows how the street network influences the accomplishment of the tasks: in Luxembourg, users are dense in the center and get sparser in the outskirts, in fact, tasks in the city center, unlike the ones in the outskirts, show little gain for HTA; on the other hand, Melbourne, being in a grid-shape and hosting a much higher number of streets in a smaller area, shows an evener distribution of users, in fact, tasks get a similar gain of accomplishment due to HTA. Bologna behaves in a fuzzier way: this could be associated with its size, much bigger and less dense than the other two, and its irregularity.

7. Conclusions

This paper presents CrowdSenSim 2.0[4] and its support for hybrid mobile crowdsensing data collection mechanisms. CrowdSenSim 2.0 extends with major advances the existing CrowdSenSim platform for simulations of MCS activities in realistic urban environments. CrowdSenSim 2.0 exhibits three main novel aspects. First, the simulator features a stateful approach that enforces all events to be executed in chronological order with a higher fine-grained temporal resolution. Second, it features two models to generate the city layouts over realistic street networks where users move, based on the popular OSM and the Military Grid Reference System. Third, the simulator supports changes in user trajectories during runtime, which is a fundamental prerequisite for hybrid data collection. In a nutshell, other advances include extensions of the architectural modules, code refactoring, and algorithms' parallel execution that boost performance by making significantly lower the runtime execution and memory utilization. This clearly enables the simulation of larger scale scenarios, which is of paramount importance for research in MCS.

We demonstrate that when feeding CrowdSenSim 2.0 and the original CrowdSenSim with the same list of events, they perform identically in terms of mobile device energy consumed for sensing and reporting, thus making CrowdSenSim 2.0 compatible with previous studies. We validate CrowdSenSim 2.0 with two use cases. First, we showcase the

[4] We make publicly available all the source files and scripts of CrowdSenSim 2.0 under https://crowdsensim.gforge.uni.lu/download.html.

performance evaluation of a distributed opportunistic algorithm for data collection that shows how the stateful approach is fundamental for specific applications. Second, we show the support to hybrid crowdsensing by developing and validating HTA, an algorithm that re-routes users offering opportunistic contribution towards the location of sensitive MCS tasks that require participatory-type of sensing contribution.

CRediT authorship contribution statement

Federico Montori: Conceptualization, Methodology, Software, Investigation. **Luca Bedogni:** Validation, Formal analysis, Writing - review & editing. **Claudio Fiandrino:** Conceptualization, Writing - original draft, Funding acquisition. **Andrea Capponi:** Visualization. **Luciano Bononi:** Supervision.

Declaration of competing interest

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

Acknowledgment

Dr. Fiandrino's work is supported by the Juan de la Cierva grant from the Spanish Ministry of Economy and Competitiveness (FJCI-2017-32309).

The authors would like to thank the students Emanuele Cortesi and Andrea Zapparoli for their valuable implementation work.

References

[1] A. Capponi, C. Fiandrino, B. Kantarci, L. Foschini, D. Kliazovich, P. Bouvry, A survey on mobile crowdsensing systems: Challenges, solutions and opportunities, IEEE Commun. Surv. Tutor. 21 (3) (2019) 2419–2465.

[2] Gartner, Gartner says global smartphone sales stalled in the fourth quarter of 2018, 2019, https://www.gartner.com/en/newsroom/press-releases/2019-02-21-gartner-says-global-smartphone-sales-stalled-in-the-fourth-quart.

[3] MarketsandMarkets, Crowd analytics market worth $1,142.5 million by 2021, 2018, https://www.marketsandmarkets.com/PressReleases/crowd-analytics.asp.

[4] J.A. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, M.B. Srivastava, Participatory sensing, Center Embedd. Netw. Sens. (2006).

[5] C. Fiandrino, B. Kantarci, F. Anjomshoa, D. Kliazovich, P. Bouvry, J. Matthews, Sociability-driven user recruitment in mobile crowdsensing internet of things platforms, in: IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6, http://dx.doi.org/10.1109/GLOCOM.2016.7842272.

[6] M. Avvenuti, S. Bellomo, S. Cresci, M.N. La Polla, M. Tesconi, Hybrid crowdsensing: A novel paradigm to combine the strengths of opportunistic and participatory crowdsensing, in: Proceedings of the 26th International Conference on World Wide Web Companion, in: WWW '17 Companion, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2017, pp. 1413–1421, http://dx.doi.org/10.1145/3041021.3051155.

[7] L.A. Kalogiros, K. Lagouvardos, S. Nikoletseas, N. Papadopoulos, P. Tzamalis, Allergymap: A hybrid mhealth mobile crowdsensing system for allergic diseases epidemiology : a multidisciplinary case study, in: 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2018, pp. 597–602, http://dx.doi.org/10.1109/PERCOMW.2018.8480280.

[8] G. Han, L. Liu, S. Chan, R. Yu, Y. Yang, Hysense: A hybrid mobile crowdsensing framework for sensing opportunities compensation under dynamic coverage constraint, IEEE Commun. Mag. 55 (3) (2017) 93–99, http://dx.doi.org/10.1109/MCOM.2017.1600658CM.

[9] Q. Zhu, M.Y. Sarwar Uddin, N. Venkatasubramanian, C. Hsu, Spatiotemporal scheduling for crowd augmented urban sensing, in: Proc. IEEE Conference on Computer Communications (INFOCOM), 2018, pp. 1997–2005, http://dx.doi.org/10.1109/INFOCOM.2018.8485869.

[10] F. Montori, L. Bedogni, L. Bononi, A collaborative internet of things architecture for smart cities and environmental monitoring, IEEE Internet Things J. 5 (2) (2018) 592–605, http://dx.doi.org/10.1109/JIOT.2017.2720855.

[11] F. Montori, E. Cortesi, L. Bedogni, A. Capponi, C. Fiandrino, L. Bononi, Crowdsensim 2.0: A stateful simulation platform for mobile crowdsensing in smart cities, in: Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, in: MSWIM '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 289–296, http://dx.doi.org/10.1145/3345768.3355929, https://doi-org.proxy.bnl.lu/10.1145/3345768.3355929.

[12] C. Fiandrino, A. Capponi, G. Cacciatore, D. Kliazovich, U. Sorger, P. Bouvry, B. Kantarci, F. Granelli, S. Giordano, Crowdsensim: a simulation platform for mobile crowdsensing in realistic urban environments, IEEE Access 5 (2017) 3490–3503, http://dx.doi.org/10.1109/ACCESS.2017.2671678.

[13] F. Montori, P.P. Jayaraman, A. Yavari, A. Hassani, D. Georgakopoulos, The curse of sensing: Survey of techniques and challenges to cope with sparse and dense data in mobile crowd sensing for internet of things, Pervasive Mob. Comput. 49 (2018) 111–125, http://dx.doi.org/10.1016/j.pmcj.2018.06.009.

[14] F. Montori, M. Gramaglia, L. Bedogni, M. Fiore, F. Sheikh, L. Bononi, A. Vesco, Automotive communications in LTE: A simulation-based performance study, in: Proc. of IEEE 86th Vehicular Technology Conference (VTC-Fall), 2017, pp. 1–6, http://dx.doi.org/10.1109/VTCFall.2017.8288297.

[15] L. Bedogni, M. Fiore, C. Glacet, Temporal reachability in vehicular networks, in: Proc. of IEEE Conference on Computer Communications (INFOCOM), 2018, pp. 81–89, http://dx.doi.org/10.1109/INFOCOM.2018.8486393.

[16] I. Lendák, K. Farkas, Evaluation of simulation engines for crowdsensing activities, 2015.

[17] C. Tanas, J. Herrera-Joancomartí, Crowdsensing simulation using ns-3, in: International Workshop on Citizen in Sensor Networks, Springer, 2013, pp. 47–58, http://dx.doi.org/10.1007/978-3-319-04178-0_5.

[18] R. Pinciroli, S. Distefano, Characterization and evaluation of mobile crowdsensing performance and energy indicators, ACM SIGMETRICS Perform. Eval. Rev. 44 (4) (2017) 80–90, http://dx.doi.org/10.1145/3092819.3092829.

[19] K. Mehdi, M. Lounis, A. Bounceur, T. Kechadi, Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool, in: Proc. of 7th International ICST Conference on Simulation Tools and Techniques, 2014, pp. 126–131, http://dx.doi.org/10.4108/icst.simutools.2014.254811.

[20] G. Boeing, Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks, Comput. Environ. Urban Syst. 65 (2017) 126–139, http://dx.doi.org/10.1016/j.compenvurbsys.2017.05.004.

[21] P. Vitello, A. Capponi, C. Fiandrino, P. Giaccone, D. Kliazovich, P. Bouvry, High-precision design of pedestrian mobility for smart city simulators, in: Proc. of IEEE International Conference on Communications (ICC), 2018, pp. 1–6, http://dx.doi.org/10.1109/ICC.2018.8422599.

[22] R. Lampinen, Universal transverse mercator (UTM) and military grid reference system (MGRS), 2001.

[23] M. Marjanović, L. Skorin-Kapov, K. Pripužić, A. Antonić, I.P. Žarko, Energy-aware and quality-driven sensor management for green mobile crowd sensing, J. Netw. Comput. Appl. 59 (2016) 95–108, http://dx.doi.org/10.1016/j.jnca.2015.06.023.

[24] Y. Gao, X. Li, J. Li, Y. Gao, A dynamic-trust-based recruitment framework for mobile crowd sensing, in: Proc. of IEEE International Conference on Communications (ICC), 2017, pp. 1–6, http://dx.doi.org/10.1109/ICC.2017.7997420.

[25] F. Montori, L. Bedogni, A. Di Chiappari, L. Bononi, Sensquare: A mobile crowdsensing architecture for smart cities, in: Proc. of IEEE 3rd World Forum on Internet of Things (WF-IoT), 2016, pp. 536–541, http://dx.doi.org/10.1109/WF-IoT.2016.7845471.

[26] A. Capponi, C. Fiandrino, D. Kliazovich, P. Bouvry, S. Giordano, A cost-effective distributed framework for data collection in cloud-based mobile crowd sensing architectures, IEEE Trans. Sustain. Comput. 2 (1) (2017) 3–16, http://dx.doi.org/10.1109/TSUSC.2017.2666043.

[27] M. Tomasoni, A. Capponi, C. Fiandrino, D. Kliazovich, F. Granelli, P. Bouvry, Why energy matters? Profiling energy consumption of mobile crowdsensing data collection frameworks, Pervasive Mob. Comput. 51 (2018) 193–208, http://dx.doi.org/10.1016/j.pmcj.2018.10.002.

[28] F. Montori, L. Bedogni, L. Bononi, Distributed data collection control in opportunistic mobile crowdsensing, in: Proc. of ACM SMARTOBJECTS, 2017, pp. 19–24, http://dx.doi.org/10.1145/3127502.3127509.

[29] G. van Rossum, F.L. Drake, Extending and Embedding the Python Interpreter, Centrum voor Wiskunde en Informatica, 1995.

[30] D. Luxen, C. Vetter, Real-time routing with openstreetmap data, in: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, in: GIS '11, ACM, New York, NY, USA, 2011, pp. 513–516, http://dx.doi.org/10.1145/2093973.2094062, http://doi.acm.org/10.1145/2093973.2094062.

[31] G. Boeing, Urban spatial order: Street network orientation, configuration, and entropy, Appl. Netw. Sci. 4 (1) (2019) 67.

Federico Montori is a postdoctoral researcher since May 2019 with the University of Bologna, Italy, where he also obtained his Ph.D. in Computer Science and Engineering in the same year. He received the B.S. and the M.S. in Computer Science, both Summa Cum Laude, from the University of Bologna in 2012 and 2015 respectively. He has

been a visiting researcher at Swinburne University of Technology, Australia, and Luleå Tekniska Universitet, Sweden. He participated in several EU projects and he is now WP leader for the H2020 Project Arrowhead Tools. His primary research interests include mobile crowdsensing, service-oriented computing, IoT automation and data analysis for IoT scenarios.

Luca Bedogni received the masters degree in computer science and Ph.D. degree from the University of Bologna, Italy, in 2013 and 2015. He was a visiting researcher at RWTH Aachen University (2013), Queen Mary University London (2015), and University of California, Irvine (2017, 2019). He was a post doctoral researcher at the University of Bologna until 2018, when he became assistant professor at the same University. He is currently an Assistant Professor at the University of Modena and Reggio Emilia, Italy, since 2019. He participated in more than 5 international and national projects. He is Member of IEEE, and served as TPC member in several international conferences. His current research interests include the study of privacy aware context aware system, in the domain of IoT, Crowdsensing and Mobile Applications. He won 3 best paper awards for his work at international conferences.

Claudio Fiandrino joined as a postdoctoral researcher the IMDEA Networks Institute in December 2016 right after having obtained his Ph.D. degree at the University of Luxembourg. He received the Bachelor Degree in Ingegneria Telematica in 2010 and the Master Degree in Computer and Communication Networks Engineering in 2012 both from Politecnico di Torino. Claudio has been awarded with the Spanish Juan de la Cierva grant and the Best Paper Awards in IEEE Cloudnet 2016, in ACM WiNTECH 2018 and in IEEE Globecom 2019. He is member of IEEE and ACM, served as Publication and Web Chair in IEEE CloudNet 2014, Publicity Chair in ACM/IEEE ANCS 2018, Workshop Co-Chair in MoCS 2019 and TPC Co-Chair in IEEE CAMAD 2019. His primary research interests include mobile crowdsensing, multi-access edge computing, and machine learning driven network optimization.

Andrea Capponi is a Ph.D. candidate at the University of Luxembourg. He received the Bachelor Degree and the Master Degree in Telecommunication Engineering both from the University of Pisa, Italy. He was a visiting research student at the McMaster University (Hamilton, ON, Canada) in 2018. Andrea's work on the impact of human mobility on edge data center deployment in urban environment received the Best Paper Award at IEEE GLOBECOM 2019. He is member of IEEE and ACM, and served as Workshop Co-Chair of MoCS 2019. His primary research interests are in the field of mobile crowdsensing, multi-access edge computing, and urban computing.

Luciano Bononi (M), (MSC, Summa cum laude, 1997, Ph.D, 2001), is Associate Professor of Computer Networks, Internet of Things, Wireless and Mobile Systems, and Mobile Applications at the Department of Computer Science and Engineering of the University of Bologna. He has co-authored more than 140 peer reviewed conference and journal publications and 8 book chapters, receiving four best paper awards, and his research areas include wireless systems and networks, protocol architectures, Internet of Things, Internet of Energy, Smart Mobility, modeling, simulation, performance evaluation, mobile services and mobile applications. He has been involved in more than 10 international research projects, and he is Associate Editor of three international Journals and guest edited more than 10 special issues. He was chair in more than 15 IEEE/ACM conferences and TPC member in more than 150 IEEE/ACM conferences on the above research topics. He is the founder and director of the Laboratory of Wireless Systems and Mobile Applications at CSE.