Speeding up Routing Schedules on Aisle-Graphs with Single Access

Francesco Betti Sorbelli, Stefano Carpin, *Senior Member, IEEE,* Federico Corò, Sajal K. Das, *Fellow, IEEE,* Alfredo Navarra, and Cristina M. Pinotti, *Senior Member, IEEE*

Abstract—In this paper, we study the Orienteering Aisle-graphs Single-access Problem (OASP), a variant of the orienteering problem for a robot moving in a so-called *single-access aisle-graph*, i.e., a graph consisting of a set of rows that can be accessed from one side only. Aisle-graphs model, among others, vineyards or warehouses. Each aisle-graph vertex is associated with a reward that a robot obtains when visits the vertex itself. As the robot's energy is limited, only a subset of vertices can be visited with a fully charged battery. The objective is to maximize the total reward collected by the robot with a battery charge. We first propose an optimal algorithm that solves OASP in $\mathcal{O}(m^2 n^2)$ time for aisle-graphs with a single access consisting of m rows, each with n vertices. With the goal of designing faster solutions, we propose four greedy sub-optimal algorithms that run in at most $\mathcal{O}(mn\ (m + n))$ time. For two of them, we guarantee an approximation ratio of $\frac{1}{2}(1 - \frac{1}{e})$, where e is the base of the natural logarithm, on the total reward by exploiting the well-known submodularity property. Experimentally, we show that these algorithms collect more than 80% of the optimal reward.

Index Terms—Orienteering problem, routing, submodularity.

I. INTRODUCTION

ROUTE planning is a key problem in robotics and sensor networks. A classic instance requires that the robot visits multiple Points of Interest (POIs) to perform some tasks, such as to collect data, deliver a package, or perform some repairs. Depending on the environment in which a robot moves, and based on its capabilities, different trajectories could be calculated. The decision-making process of selecting a trajectory is inherently tied to the energy (fuel) consumption, because a robot must be able to reach its destinations, perform the assigned tasks, and then return to a recharge/refueling station before it runs out of power and remains stranded in the environment.

In this paper, we focus on a robot moving on a specific environment modeled as a graph that belongs to the so-called

aisle-graphs family recently introduced in [1]. Specifically, we assume the environment is modeled by a set of vertices in rows (*aisles*) accessible through one column (*junction line*) at one endpoint of the rows (see Figures 1 and 3). This is indeed the topology that can be abstracted from Figure 1a and Figure 1b when the robots perform complex tasks. In a vineyard, for example, when the robot moves along a row it has grapevines on its left and right. For complex tasks, like picking grapes in vineyards [2] (or pruning trees in orchards [3]), the robot must act on both the left and right side of the aisles. Since the growth of grapes on vines is not homogeneous inside a vineyard, often an indiscriminate full visit is not the most appropriate choice, especially for autonomous energy-constrained vehicles. The aisle can then be seen as a two-line aisle: going back and forth, the robot uses the two different lines and serves always the grapevines on its left (see Figure 2). Consequently, in this scenario, the robot always enters and exits from the same side of the vineyard. The same concept can be applied inside a warehouse, like that in Figure 1b. In fact, in a warehouse aisle, there are shelves on the left and on the right from which the robot can pick items. Interestingly, a commercial solution which implements a warehouse system with single access for pallets (flat wooden structure) is proposed in [4].

(a) Vineyard.

(b) Warehouse.

Fig. 1. Real examples of aisle-structures with single access.

In the applications we sketched above, the budget is determined by the maximal distance a robot can travel on a single battery charge, the cost of an edge is its length (or the amount of energy spent to traverse it), and the reward of a vertex is the utility generated by the robot by performing an assigned task at a certain location associated with the vertex. Given a single-access aisle-graph with a reward associated to each vertex, and assuming an unitary cost for traversing any edge, our objective is to determine a cycle in the graph that can be traversed without exceeding the preassigned constant budget and maximizes the total sum of the rewards of the visited vertices. This is a new variant of the well known *Orienteering* Problem (OP) [5], called *Orienteering Aisle-graphs Single-access Problem* (OASP). As in the original orienteering problem, the reward of visiting a vertex is collected only the first time the vertex is visited by the robot, whereas the cost for traversing an edge is charged as many times as the robot traverses such an edge.
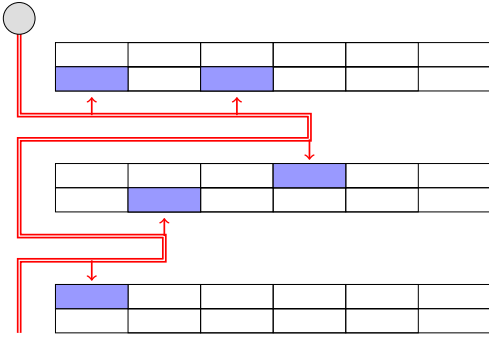

Fig. 2. Abstraction of aisle-graph.

As will be discussed below, for general graphs, the orienteering problem is known to be NP-hard [5], thus motivating the design of heuristics, or approximation algorithms. However, for the special class of aisle-graphs with a single access, the OASP variant of the orienteering problem is polynomially solvable as we prove in Section IV. The optimal algorithm OPTSA, whose time complexity is a polynomial of grade 4 in the maximum between the number of aisles and the number of vertices composing a single aisle, may be not scalable in real-world applications with tens of thousand aisles, each with hundreds of vertices. For example, commercial vineyards often feature units (called blocks) with more than $50,000$ vines arranged in hundreds of rows with 200 or more trees. The potentially large size of aisle-graphs and the fact that the robot's trajectory has to be frequently recomputed due to the reward changes, motivate us to search also for simpler and faster suboptimal solutions with guaranteed reward. Accordingly, we then propose four greedy algorithms, and for two of them we prove a constant factor approximation ratio between the reward of their solutions and that of the optimal solution.

Contributions. Our results are summarized as follows:

- We provide a polynomial time optimal algorithm, called Optimum Single Access (OPTSA), for the Orienteering Aisle-graphs Single-access Problem (OASP).

- We design two greedy heuristic algorithms, called Greedy Max Element (GDYME) and Greedy Max Cumulative (GDYMC), which are more efficient in terms of time and space complexity with respect to the optimal solution.

- We provide two additional greedy approximation algorithms faster than the optimal one, but slower than the greedy ones, called Approximate Max Ratio Element (APXMRE) and Approximate Max Ratio Cumulative (APXMRC), which provide a $\frac{1}{2}(1 - \frac{1}{e})$-approximation guarantee in terms of the collected reward by exploiting submodularity properties – with e being the base of the natural logarithm.

- We evaluate the performance of our algorithms on synthetic and real data, showing that the approximation algorithms collect more than 80% of the optimum reward.

This paper extends preliminary results presented in [6] and [7]. In particular, the algorithm OPTSA discussed herein was first introduced in [6] but is here adapted to solve a different problem, i.e., OASP. Its optimality proof is also new and has never appeared before. Moreover, we provide a much thorough experimental comparison for all the algorithms on a set of real-world data obtained from a large scale commercial vineyard.

Organization. The rest of the paper is organized as follows. Section II reviews the related work on the orienteering problem. Section III formally defines the problem and submodularity properties. Section IV introduces the OPTSA algorithm, a polynomial-time optimal algorithm for OASP, and prove its optimality. Section V proposes four greedy algorithms in details, while Section VI proves a constant factor approximation for the APXMRE and APXMRC algorithms by exploiting the submodularity property. Section VII evaluates the effectiveness of our algorithms on large scale instances through simulation experiments. Section VIII offers conclusions.

II. RELATED WORK

In this section, we provide selected references to previous works on the orienteering problem and submodularity.

Orienteering. Orienteering is a classic combinatorial optimization problem defined over graphs with costs associated with edges and rewards associated with vertices. Introduced in [5], the orienteering problem is also known in the literature as the *bank robber* problem [8] and a few other names [9]. Orienteering was shown to be NP-hard in [5] and APX-hard in [10]. Numerous variants of the orienteering problem have been proposed in the literature. For example, the start vertex may be fixed (rooted orienteering problem) or not (unrooted orienteering problem), edges can be directed or undirected, vertices in the graph may be placed on a plane, or there may be more than one robot collecting rewards. The reader is referred to [11] for a survey of different formulations. Due to its intrinsic computational complexity, numerous heuristic solutions have been proposed in literature [11]. Exact solutions using branch and bound techniques have been proposed as well [12], but do not scale to problem instances with tens of thousands of vertices like those motivating this work. A different line of

research has aimed at developing approximation algorithms for orienteering, often providing specialized results applicable only to restricted versions of the problem. For the rooted version of the problem, the first constant-factor approximation was given in [13], where the authors give a 4-approximation. This result was later improved in [14], where for the case of directed graphs the authors give a $\mathcal{O}(2 + \varepsilon)$-approximation with a time complexity of $n^{\mathcal{O}(\frac{1}{\varepsilon^2})}$ (where n is the number of vertices in the graph), and for the case of undirected graphs, the authors provide an $\mathcal{O}(\log^2 \text{OPT})$-approximation, where OPT is the number of vertices in an optimal solution. For the case where the graph is planar and fully connected, a $(1+\varepsilon)$-approximation algorithm was proposed in [15].

Orienteering Applications in Robotics. Orienteering has found various applications in robotics. These include precision irrigation [16] and robotic harvesting [17]. The orienteering framework has also been used to study surveillance problems [18], monitoring [19], and information gathering [20]. Stochastic extensions of the orienteering problem with random edge costs have also been used to study *survival* problems where multiple robots are navigating a dangerous environment and the objective is to visit locations while ensuring a minimal probability of success [21].

Orienteering Based on Submodularity. There exist other works where the orienteering problem is approached observing that the collected rewards may follow the well known *submodularity* property. A function is *submodular* if its marginal increment (when a single element is added to the input set) decreases as the size of the input set increases [22].

The aforementioned paper [21] as well as [23] studied a multi-robot coordination problem modeled as an orienteering problem for general graphs leveraging on the submodularity property. The authors consider a team of robots tasked with visiting sites in a risky environment and subject to team-based operational constraints. Such a problem is modeled using a graph and a matroid for capturing the team-based operational constraints. The formulated Matroid Team Surviving Orienteers (MTSO) problem has a submodular structure, and hence polynomial-time algorithms with a guaranteed solution are developed. In [24], [25], approximation algorithms are proposed for the Team Orienteering Problem in the context of mobile charging of rechargeable sensors. This problem has many potential applications related to the Internet of Things (IoT) and smart cities, such as dispatching energy-constrained mobile chargers to charge as many energy-critical sensors as possible to prolong the network lifetime. In [26], the submodularity property is used to design a trajectory planning algorithm for an aerial 3D scanning employing a drone. The authors leveraged submodularity to develop a computationally efficient method for generating scanning trajectories, that reasons jointly about coverage rewards and travel costs. However, none of the above works addresses the constrained aisle-graph structure.

In [27], an approximation algorithm is proposed that runs in quasi-polynomial time for the submodular tree orienteering problem. Differently from us, they work on directed graphs. Moreover, their approximation ratio is $\frac{\log k}{\log \log k}$, where $k \leq$ $|V|$ is the number of vertices in an optimal solution, while in our work we guarantee a constant approximation ratio of $\frac{1}{2}(1 - \frac{1}{e})$. In [28], an interesting new problem called Robust Multiple-Path Orienteering Problem, in which the main goal is to construct a set of paths for robots guaranteeing robustness in case of malicious attacks, is proposed. The authors provided an approximation algorithm based on submodularity.

Orienteering on Aisle-graph Family. The orienteering problem for aisle-graphs was first investigated in [1] which considered variants on the graph topology and robot capabilities along with experimental results. The authors in [1], working on aisle-graphs with two accesses (two junction lines, one at each endpoint of the aisles), proposed two greedy heuristics, Greedy Full-Row (GFR) and Greedy Partial-Row (GPR), which respectively select a subset of full or partial rows to be traversed. At each selection, the robot computes the budget required to collect rewards from its current position and prefers full/partial rows with maximum reward per unit of budget. The time complexities of GFR and GPR are $\mathcal{O}(m^2)$ and $\mathcal{O}(m^2n)$, respectively, where m is the number of aisles and n is the number of vertices composing each aisle. The authors in [6] developed polynomial-time algorithms to improve some of the problems faced in [1]. In particular, they designed an optimal algorithm, called Optimal Full-Row Improved (OFR-I), that improves on GFR by determining the optimal solution for the full-row policy, whose time complexity is $\mathcal{O}(m \cdot \max\{n, \log m\})$. Moreover, they proposed Heuristic General-case (HGC), which slightly improves on GPR at a higher time complexity. In [29], authors studied the problem of routing multiple robots within a vineyard, where movement is limited when a row is entered, for the application of precision irrigation, proposing three algorithms combining GFR and GPR either in series or in parallel.

In a seminal paper in [30], aisle-graphs have been investigated for solving the original version of the well known Traveling Salesman Problem (TSP). The OP is a particular variant of TSP where profits/rewards are introduced. In other words, the goal of TSP is only to find the minimum cost cycle in the area such that all the points are visited exactly once. Despite TSP is NP-hard [31] on general graphs, the authors in [30] have found an optimal polynomial algorithm (linear in the number of aisles) for aisle-graphs with two accesses based on an efficient dynamic programming approach. Further, this result has been generalized by the authors in [32] providing an optimal pseudo-polynomial algorithm with time complexity $\mathcal{O}(nh7^h)$ where $h \leq n$ denotes the number of horizontal accesses in the area.

Finally, in [33], aisle-graphs are used to model a warehouse where an automated picking system is implemented. The distance traveled for collecting items belonging to a customer order by a robot, forced to follow the aisles, is compared with the distance traversed by a drone that accomplishes the same goal flying freely inside the warehouse.

III. PROBLEM DEFINITION

Consider an undirected *aisle-graph* $A(m, n) = (V, E)$, where m denotes the number of rows (aisles), n denotes

the number of columns (number of vertices composing each aisle), while rows are all connected only via the first column. Formally, the set of vertices is defined as $V = \{v_{i,j} | 1 \leq i \leq m, 1 \leq j \leq n\}$. The set of edges E is defined as follows:

- Each vertex $v_{i,j}$ with $1 \leq i \leq m$ and $1 < j < n$ has two edges, one toward $v_{i,j-1}$ and the other toward $v_{i,j+1}$;
- each vertex $v_{i,1}$ with $1 < i < m$ has three edges: one toward $v_{i-1,1}$, one toward $v_{i+1,1}$, and one toward $v_{i,2}$. Accordingly, edges connected to the corner vertices $v_{1,1}$ are $v_{m,1}$ are well defined.

Problem 1 (Orienteering Aisle-graphs Single-access Problem (OASP)). *Let $A(m,n)$ be an aisle-graph; $v_{1,1} \in V$ be the home vertex; $\mathcal{R} : V \to \mathbb{R}_{\geq 0}$ be a reward function where $\mathcal{R}(v_{i,j}) = 0$ if $j = 1$; let α be a positive constant cost (in terms of the budget required) to traverse any edge; let $B > 0$ be the budget, i.e., the maximum distance that can be traveled along edges. The OASP aims to find a cycle of maximum reward starting at $v_{1,1}$ of cost no greater than B.*

Without loss of generality, we assume $\alpha = 1$. Figure 3 shows an aisle-graph $A(4,5)$. The vertices in the first column (for $j = 1$) do not provide any reward, and they are used simply to connect the m rows. In what follows, by r_i and c_j we denote the i-th row and the j-th column of A, respectively, and by $R = \{r_1, \ldots, r_m\}$ and $C = \{c_1, \ldots, c_n\}$ the set of rows and columns of A, respectively.
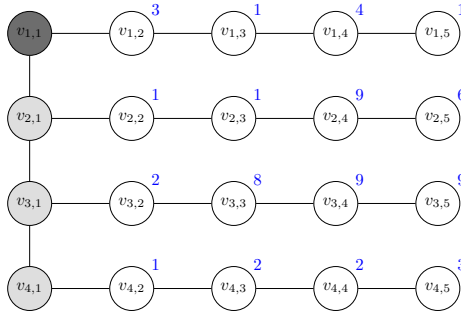

Fig. 3. Aisle-graph $A(4,5)$; the interconnecting vertices are in dark, the home vertex is the grayest, the edges costs are unitary, and the rewards $\mathcal{R}(v_{i,j})$ are in blue (note that interconnecting vertices have no reward).

A. Submodularity

In this section, we revise the fundamental definition of submodularity that will be used to derive some of our results.

Definition 1 (Submodularity [22]). *Given a finite set $V = \{v_1, \ldots, v_n\}$, a set function $f : 2^V \to \mathbb{R}$ is submodular if for any $X \subseteq Y \subseteq V$ and $v \in V \setminus Y$,*

$$f(X \cup v) - f(X) \geq f(Y \cup v) - f(Y) \qquad (1)$$

or equivalently, for each $X, Y \subseteq V$

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \qquad (2)$$

A well-studied class of problems aims to select a k-element subset that maximizes a monotone submodular objective f classified with respect to the cost function. The simplest subset

selection problem with cardinality constraint $|X| \leq k$ is NP-hard [34]. The greedy algorithm, which iteratively selects one element with the largest marginal gain, can achieve $(1 - \frac{1}{e})$-approximation guarantee [22]. The best known guarantees are $\frac{1}{\kappa}(1 - e^{-\kappa})$ due to [35], and $(1 - \frac{\kappa}{e})$ due to [36] for a cardinality constraint, where $0 \leq \kappa \leq 1$ is called *curvature* [35] that substantially characterizes how close a monotone submodular function f is to the modularity. A generic function f is *modular* if and only if $\kappa = 0$. A function $f : 2^N \to \mathbb{R}$ is monotone *modular* (also called "additive" or "linear") if $f(X) = \sum_{j \in X} w(j)$ for some $w : N \to \mathbb{R}_{\geq 0}$.

For the class of problems aiming to select a subset that maximizes a monotone submodular objective f under the cost constraint $\mathcal{C}(X) \leq B$, where $\mathcal{C}$ is a linear function, the greedy rule selecting the element with the largest marginal gain on f leads to an unbounded approximation ratio [37]. Instead, the generalized greedy rule that iteratively selects the element with the largest ratio of the marginal gain on f and $\mathcal{C}$, and outputs the better of the found subset and the best single element achieves $\frac{1}{2}(1 - \frac{1}{e})$-approximation guarantee [38].

B. Reward and Cost Modularity

OASP aims to select a subset S of vertices V in A that maximizes the reward under a constraint on the traveling cost to reach X. To formalize OASP, recall that the orienteering problem with a budget constraint can be stated as:

$$S = \arg\max_{X \subseteq V} \{\mathcal{R}(X) \mid \mathcal{C}(X) \leq B\} \qquad (3)$$

where $\mathcal{R}$ is the modular monotone reward function to maximize, $\mathcal{C}$ is the modular monotone cost function subject to the constraint to be limited by B, the budget given as input.

Due to the specific structure of the aisle-graph A, given a subset of vertices X, the cycle of minimum cost that visits all the vertices in X traverses the subgraph T_X of A containing $v_s = v_{1,1}$ that consists of the vertices in X along with the minimum set of vertices that make it a connected subgraph of A (see, e.g., Figure 4). Such a subgraph is uniquely defined because there is only one simple path from any two vertices of A, i.e., T_X is a tree.
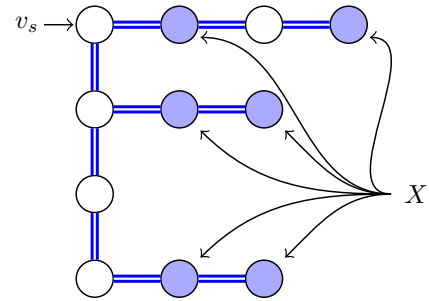
Fig. 4. The selected subset of vertices $X \subseteq V$ (in blue) and the subgraph T_X that connects X (whole subgraph rooted in v_s with double edges in blue).

Given X, the cycle of minimum cost to visit X starts from $v_{1,1}$, visits the vertices in T_X scanning the rows of A in order from the top to the bottom. Each row is visited back-and-forth up to its furthest vertex in X. After visiting the last row $i_{\max}$

that has vertices in X, the cycle is completed by vertically traversing the first column from $v_{i_{\max},1}$ to $v_{1,1}$. The cost of such a cycle is given by the sum of the costs of each traversed edge multiplied by the number of times it is traversed, that is, $\mathcal{C}(X) = 2(|T_X| - 1)$.

(a) $\mathcal{C}(X \cup v_{5,4})$.

(b) $\mathcal{C}(X \cup v_{3,4})$.

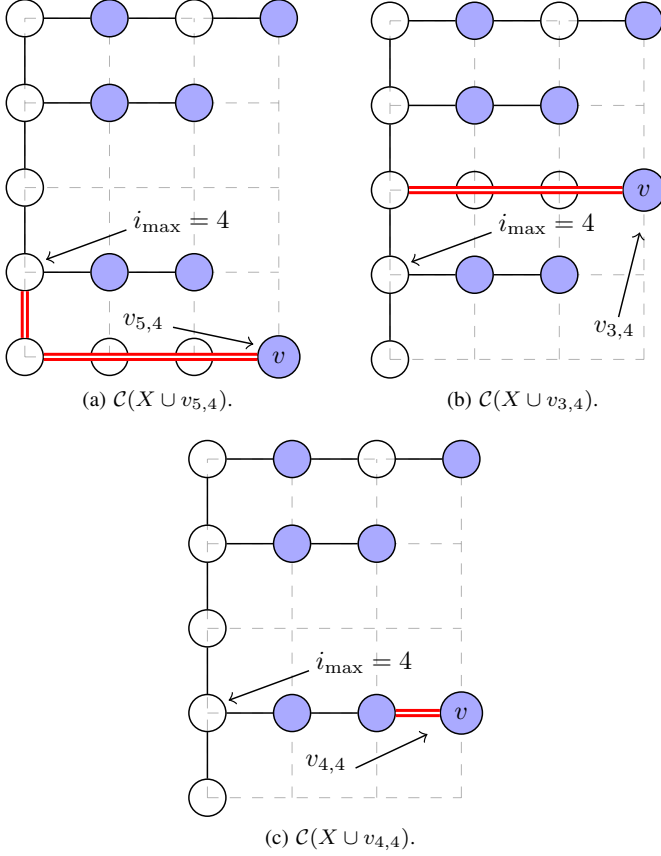(c) $\mathcal{C}(X \cup v_{4,4})$.

Fig. 5. Marginal cost $\mathcal{C}(X \cup v_{i,j}) - \mathcal{C}(X)$ (in red) when the vertex $v = v_{i,j}$ is added and $i_{\max} = 4$ with unitary costs: (a) $i > i_{\max}$ with marginal cost 8; (b) $i < i_{\max}$ with marginal cost 6, no vertex of row i belongs to S; (c) $i \leq i_{\max}$ with marginal cost 2.

Consider for instance Figure 4. The cycle $\mathcal{C}(X)$ traverses, in the order, row 1 back-and-forth up to vertex $v_{1,4}$, row 2 back-and-forth up to vertex $v_{2,3}$, vertex $v_{3,1}$ in row 3, and row 4 back-and-forth up to vertex $v_{4,3}$. Finally, it returns to $v_{1,1}$ traversing back column 1. The cost of X in Figure 4 is $2(|T_X| - 1) = 20$ (precisely, $3 \cdot 2 + 1 + 2 \cdot 2 + 2 + 2 \cdot 2 + 3$). Since T_X is uniquely defined given X, for convenience, we define, in Eq. (3), $\mathcal{C}(X)$ as the cost of visiting the tree T_X, and the function $\mathcal{R}(X)$ as the sum of the rewards of all the vertices $v \in X$. Note that $\mathcal{R}(T_X) \geq \mathcal{R}(X)$ because $X \subseteq T_X$.

It is very important to note that the marginal cost of $\mathcal{R}$ and $\mathcal{C}$, i.e., the cost for updating both $\mathcal{R}$ and $\mathcal{C}$ after the set X is increased by one vertex, can be computed in constant time. This is trivial for $\mathcal{R}$, indeed $\mathcal{R}(X \cup v) = \mathcal{R}(X) + \mathcal{R}(v)$. The increment of the cost of the $\mathcal{C}(X)$ when a new vertex is added depends on the position of v with respect to X. The possible scenarios are illustrated, for example, in Figure 5 and the increment is depicted in red. Figure 5 illustrates how the cost changes depending on the relative position of $v = v_{i,j}$ with respect to the last line in the solution set S. It is easy

to compute the increment in constant time. Specifically, in Figure 5a (case $i > i_{\max}$), the increment of cost considers the vertical movements for reaching the new i-th row and the horizontal movements for visiting vertices up to v; in Figure 5b (case $i < i_{\max}$ with no vertex of row i belongs to S), the increment of cost only considers the full horizontal movements for visiting vertices up to v; in Figure 5c (case $i \leq i_{\max}$), the increment of cost only considers the partial horizontal movements for completing the visit of the i-th row up to v. From the above considerations, Eq. (2) is satisfied as equality from both $\mathcal{R}$ and $\mathcal{C}$, and thus such functions are modular.

IV. OPTIMAL SOLUTION FOR OASP

We devise a dynamic programming algorithm, called Optimum Single Access (OPTSA), that optimally solves OASP in polynomial time.

During the initialization, two tables T and R of size $m \times n$ and $m \times (\frac{B}{2} + 1)$, respectively, are created. Table R has columns $j = 0, 1, \ldots, \frac{B}{2}$. Instead, Table T is initialized as follows: $T[i,j] = \sum_{k=1}^{j} \mathcal{R}(v_{i,k})$ for each i, j which represents, fixed a row i, the cumulative reward up to the j-th column starting from the leftmost side, with $1 \leq j \leq n$. Clearly, $T[i,1] = 0$. The initialization of table T has time and space cost $\mathcal{O}(mn)$ since its size is $m \times n$ and the cells of each row can be filled in constant time from left to write: $T[i,j] = T[i,j-1] + \mathcal{R}(v_{i,j})$. The entry $R[i,b]$ records the largest reward that can be attained with budget $2b$, where $0 \leq b \leq \frac{B}{2}$, visiting only the first i rows. Finally, let $Q[i,b]$ be the last column visited in row i to obtain the reward $R[i,b]$.

Some general properties will be exploited by our dynamic programming solution:

- If $B \geq 2nm + 2(m-1)$ then the robot can visit the whole graph.
- Any optimal solution traverses the selected rows either in increasing or decreasing order with respect to their row indices.
- Any optimal solution visits each row at most once.
- For each row, the algorithm has to select the last vertex to visit. If no vertex is visited, still $v_{i,1}$ has to be traversed to reach the subsequent row $i + 1$.

The optimal solution that visits the first j vertices in row i, with $j \geq 1$, gains $T[i,j]$ reward from row i, and spends $2j$ budget to serve rows i starting from row $i - 1$. Namely, we reserve 2 units of budget to change row and $2(j-1)$ units to traverse row i. We thus have the next recurrence. The first row of table R is defined as follows:

$$R[1,b] = \begin{cases} T[1, b+1] & 0 \leq b \leq n-1 \\ T[1, n] & n \leq b \leq \frac{B}{2} \end{cases}$$

Note that for $b = 0$, $T[1,0] = \mathcal{R}(v_{1,1}) = 0$. Then, for each subsequent row $R[i,b]$, $i \geq 2$:

$$R[i,b] = \begin{cases} -\infty & b < i-1 \\ 0 & b = i-1 \\ \max_{1 \leq j \leq \min\{b-i+2, n-1\}} \{R[i-1, b-j] + T[i,j]\} & b > i-1 \end{cases}$$
(4)

Note that $j \leq \min\{b-i+1, n-1\}$ is obtained by observing that $j \leq n-1$, $b - (j-1) - 1 \geq 0$ and more strictly $b - (j-1) - 1 \geq i - 2$ to have enough budget to reach row $i - 1$. If $b = i - 1$ and $i \geq 2$, $R[i, b]$ is 0 because there is only enough budget to reach row i traversing column 1.

Table Q is then filled recalling for each position the column that has given the maximum reward, i.e., $Q[i, b] \leftarrow j = \arg\max R[i, b]$. Finally, the reward of the optimal solution with budget $\frac{B}{2}$ is found by calculating $\max_{1 \leq i \leq m} R[i, \frac{B}{2}]$ because the furthest row reached by the optimal solution, is not known in advance. Having fixed the last row, the solution is computed in $\mathcal{O}(m)$ tracing back the choices using the table Q. Note that with a single execution of OPTSA, the optimal reward for any value b, with $0 \leq b \leq \frac{B}{2}$, is computed. That is, the maximum reward for budget b is the maximum in column b of table R.

The algorithm runs in time $\mathcal{O}(mn\frac{B}{2})$ plus $\mathcal{O}(m)$ to retrieve the solution, and takes $\mathcal{O}(mn + m\frac{B}{2})$ space. Since the maximum budget is $B \geq 2nm + 2(m-1)$, i.e., B is upper bounded by $\mathcal{O}(mn)$, then algorithm OPTSA is strictly polynomial in the size of the input.

By the above discussion, the next theorem can be stated.

Theorem 1. *Algorithm* OPTSA *optimally solves OASP in time* $\mathcal{O}(mn\frac{B}{2})$.

Proof. The running time is obvious. Let us define $R(i, b)$ as the optimal profit to reach any vertex of row i when the budget is $2b$. The solution $Q(i, b)$ associated with $R(i, b)$ has maximum profit and must traverse the vertices $v_{1,1}, \ldots, v_{i,1}$ of c_1, i.e., the first column with no rewards.

For the correctness, we use induction. For the base case, $i = 1$ and any b, or any row i and $b \leq i - 1$, the correctness follows from the above discussion.

Inductive Step: When computing $R[i, b]$ by the induction hypothesis, we have that $R[i - 1, b - j]$ for any $1 \leq j \leq \min\{b - i + 2, n - 1\}$ are already computed correctly. Since any optimal solution $Q[i, b]$ visits the rows in increasing index order, traverses only once row i, up to any c_j (recall that $j = 1$ means that the row is not visited), and must traverse the vertices $v_{1,1}, \ldots, v_{i,1}$ of c_1, $Q[i, b]$ is built starting from a sub-problem that traverses the vertices $v_{1,1}, \ldots, v_{i-1,1}$ and some vertices of row i. Then, $Q[i, b]$ is based on a sub-problem that considers up to row $i - 1$, leaves 1 unit of budget to reach row i, and leaves $(j - 1)$ units of budget to reach vertex j in row i, for some $1 \leq j \leq b - i$. Hence, the value of $R[i, b]$ in Eq. (4) is correct.

Now, assume by contradiction, that there exists a solution $Q'[i, b] \neq Q[i, b]$ with cost $R'[i, b] > R[i, b]$, and that is the first time that Eq. (4) does not provide the optimum. Let vertex $v_{i,j}$ be the vertex reached by $Q'[i, b]$ in row i. The solution $Q'[i, b]$ gains $T[i, j]$, and $R'[i, b] - T[i, j]$ corresponds to the profit of a sub problem $Q[i - 1, b - j]$ for which $R[i - 1, b - j]$ is optimal. That is, $R'[i, b] - T[i, j] = R[i - 1, b - j]$ and thus $R[i, b] = R[i - 1, b - j] + T[i, j] = R'[i, b]$.

Finally, since we do not know in advance the furthest row that belongs to the optimal solution, the optimal solution with budget B is found in column $\frac{B}{2}$, and precisely it is $\max_{1 \leq i \leq m} R[i, \frac{B}{2}]$. □

Example. For the example in Figure 3, assuming a budget $B = 16$, the Table R is iteratively created from its first row.

R	0	1	2	3	4	5	6	7	8
1	0	3	4	8	9	9	9	9	9

Here, we assume to visit only the first row. Specifically, $R[1, 0] = 0$ because with no budget there is no reward; $R[1, 1] = 3$ because with a budget of $2 \cdot 1 = 2$, the maximum obtainable reward is 3; $R[1, 2] = 4$ because with a larger budget $2 \cdot 2 = 4$, it is possible to visit the third column, and hence the cumulative reward is $3 + 1 = 4$; $R[1, 3] = 8$ because spending a budget of $2 \cdot 3 = 6$, the cumulative reward is $3 + 1 + 4 = 8$, and so on. Clearly, $R[1, 4] = \ldots = R[1, 8] = 9$ because the available budget is sufficiently enough for completely visiting the first row as a whole. The second row of table R is the following one.

R	0	1	2	3	4	5	6	7	8
2	$-\infty$	0	3	4	11	17	20	21	25

Assuming to visit also the second row, $R[2, 0] = -\infty$ because, with no budget, there is no chance for computing a cycle that visits any vertex in the second row and going back to $v_{1,1}$. $R[2, 1] = 0$ because with a budget of 2, it is possible only to visit $v_{2,1}$ and go back, without any reward since the elements on the first column do not provide reward. The first interesting value is $R[2, 4] = 11$, because with budget 8, forcing to visit the second row, it is possible to arrive up to $v_{2,4}$ which gives a cumulative reward of $1 + 1 + 9 = 11$, which is the maximum possible. Then, $R[2, 7] = 21$ because with budget 14, visiting $v_{1,2}$ with cumulative reward of $3 + 1 = 4$, and visiting $v_{2,5}$ with cumulative reward $1 + 1 + 9 + 6 = 17$, the total reward is $4 + 17 = 21$.

The same strategy, applying Eq. (4), permits to build the final table R.

R	0	1	2	3	4	5	6	7	8
1	0	3	4	8	9	9	9	9	9
2	$-\infty$	0	3	4	11	17	20	21	25
3	$-\infty$	$-\infty$	0	3	10	19	28	31	32
4	$-\infty$	$-\infty$	$-\infty$	0	3	10	19	28	31

The maximum achievable reward assuming to visit only the first row is 9, i.e., full visit of the row because there is available budget. In the case of considering also the second row, it is not possible to do two full visits for both the first and second rows, since the cost will be larger than the budget, hence the maximum achievable reward is 25. The same could be done up to the third row, i.e., maximum budget 32, and finally up to the fourth, i.e., 31. It is easy to see that the optimal solution visits only the first three rows obtaining a total reward of 32. The optimal solution is extracted from table Q, which is:

Q	0	1	2	3	4	5	6	7	8
1	1	2	3	4	5	5	5	5	5
2	$-\infty$	1	1	1	4	5	5	5	5
3	$-\infty$	$-\infty$	1	1	3	4	5	5	5
4	$-\infty$	$-\infty$	$-\infty$	1	1	1	1	1	1

The solution is extracted from the index in Q that has the maximum reward in R, i.e., $Q[3,8] = 5$, so $v_{3,5}$ is visited. Then, the farthest column to be visited from the previous row is $Q[3-1, 8-Q[3,8]] = Q[2,3] = 1$, i.e., up to $v_{2,1}$. Finally, the last column of row 1 to be visited is $Q[2-1, 3-Q[2,3]] = Q[1,2]$, i.e., up to $v_{1,2}$. In conclusion, the optimal solution is: $S = \{v_{1,1}, v_{1,2}, v_{1,3}, v_{2,1}, v_{3,1}, v_{3,2}, v_{3,3}, v_{3,4}, v_{3,5}\}$.

V. PROPOSED GREEDY ALGORITHMS

In this section, we propose two basic greedy algorithms, called Greedy Max Element (GDYME) and Greedy Max Cumulative (GDYMC), for OASP; and two additional greedy approximation algorithms, called Approximate Max Ratio Element (APXMRE), and Approximate Max Ratio Cumulative (APXMRC). We also compare the time and space complexity of these algorithms with the optimal algorithm, Optimum Single Access (OPTSA).

A. The GDYME Algorithm

The Greedy Max Element (GDYME) algorithm is a simple greedy algorithm for OASP. The main idea behind it is to visit the vertices in decreasing order of their rewards, i.e., vertex $v_{i,j}$ with current largest reward is either added to the solution set S, if it is reachable with the residual budget or discarded. In addition to the vertex $v_{i,j}$ selected by the greedy strategy, at each step, the robot collects on row i all the available rewards from $v_{i,1}, \ldots, v_{i,j}$. This is a simplification of the fact that at some point all vertices passed through to reach $v_{i,j}$ would be chosen by GDYME as they require no budget to be reached. Hence, selected the vertex $v_{i,j}$, the current solution set S adds $v_{i,j}$ along with all the vertices $v_{i,k}$ with $1 \le k < j$ of the i-th row. Recall that the reward of a vertex is collected only once even if the robot passes through it several times.

Algorithm 1: GDYME (Graph A, Budget B)

1   $S \leftarrow v_{1,1}$
2   $R \leftarrow$ BUILDMHEAP($\mathcal{R}(v_{i,j})$), $1 \le i \le m, 1 \le j \le n$
3   **while** $\mathcal{C}(S) \le B$ **do**
4      $v_{i,j} \leftarrow$ EXTRACTARGMAX(R)
5   **if** $\mathcal{C}(S \cup v_{i,j}) \le B$ **then**
6         $S \leftarrow S \cup v_{i,k}, 1 \le k \le j$
7   EXTRACTHEAP($v_{i,k}$), $1 \le k \le j$

8   **return** $\mathcal{R}(S)$

Algorithm 1 first initializes S with the home vertex $v_{1,1}$, whose reward is null by definition (Line 1) and subsequently creates a max-heap R with rewards of all the vertices in V (Line 2). This preprocessing takes $\mathcal{O}(mn)$ time.

Next, the main loop (Line 3) iteratively selects and extracts from the heap structure R the root $v_{i,j} \in V$ (Line 4), and verifies whether $v_{i,j}$ is reachable with respect to the available budget or not (Line 5). Recalling that the minimum path to visit a set of vertices of the aisle-graph A is unique, the cost $\mathcal{C}(S \cup v_{i,j})$ can be computed in constant time incrementally from $\mathcal{C}(S)$ (see Figure 5).

As already pointed out, if adding vertex $v_{i,j}$ is feasible (from Line 6), all the vertices $v_{i,k}, 1 \le k \le j$, will be added to S.

Namely, adding these vertices does not require extra budget because they are on the path to $v_{i,j}$. At the end of the main loop, the total reward $\mathcal{R}(S)$ is returned (Line 8) in $\mathcal{O}(|S|)$ time.

The main loop (Line 3) is executed at most $\frac{B}{2}$ times because the addition of a new vertex to S requires at least 2 units of budget. Since each item inserted in S is extracted from R, the algorithm spends $\mathcal{O}(|S| \log(mn))$ time to handle R. Clearly $|S|$ is upper bounded by B. Moreover B can be obviously considered as upper bounded by $\mathcal{O}(mn)$, since the minimum budget that allows to visit the whole input aisle-graph is $2(mn + m)$. Any larger budget would lead to have a trivial solution, i.e., a full visit. Hence, the total time complexity of GDYME is $\mathcal{O}(mn + B + |S| \log(mn)) = \mathcal{O}(mn \log(mn))$; and its space complexity is $\mathcal{O}(mn)$.

Example. For the example in Figure 3 and a budget $B = 16$, the first element greedily added to the solution set S is $v_{2,4}$ with value 9. Consequently, all the other elements from the 2-th row are put in S. From $v_{2,4}$ there is enough budget for visiting other vertices, and in fact the next available vertex the with largest reward is $v_{3,4}$, with reward 9. Now, selecting $v_{3,4}$, there is no sufficient budget for visiting other vertices. Finally, the solution is $S = \{v_{1,1}, v_{2,1}, v_{2,2}, v_{2,3}, v_{2,4}, v_{3,1}, v_{3,2}, v_{3,3}, v_{3,4}\}$ with total reward 30.

B. The GDYMC Algorithm

The Greedy Max Cumulative (GDYMC) algorithm keeps the same greedy strategy as GDYME, but selects the vertex with the maximum cumulative reward. In the preprocessing step, for each vertex $v_{i,j}$, the algorithm initially sums up in a matrix R the cumulative reward $R[i,j] = \sum_{k=1}^{j} \mathcal{R}(v_{i,j})$, i.e., the sum of the rewards of all the vertices belonging to the i-th row up to the j-th column. There is only one vertex with maximum cumulative reward to be selected in each row, and at the beginning, this vertex is $v_{i,n}$ because the cumulative reward is a monotone function. During the execution of the algorithm, the candidate vertices, one per each row, are stored in a max-heap.

Algorithm 2: GDYMC (Graph A, Budget B)

1   $S \leftarrow v_{1,1}$
2   $R[i,j] \leftarrow \sum_{k=1}^{j} \mathcal{R}(v_{i,k}), 1 \le i \le m, 1 \le j \le n$
3   $M \leftarrow$ BUILDHEAP($R[i,n]$), $1 \le i \le m$
4   **while** $\mathcal{C}(S) \le B$ **do**
5      $v_{i,j} \leftarrow$ EXTRACTARGMAX(M)
6   **if** $\mathcal{C}(Sv_{1,1} \cup v_{i,j}) \le B$ **then**
7         $S \leftarrow S \cup_{1 \le k \le j} v_{i,k}$
8   **else**
9   INSERTHEAP($M, R[i, j-1]$)

10   **return** $\mathcal{R}(S)$

Algorithm 2 sets the initial solution $S = \{v_{1,1}\}$ (Line 1), and computes the cumulative rewards row by row (Line 2). Then, a max-heap M is built (Line 3) using the largest cumulative rewards of each row, i.e., $R[i,n]$, for $1 \le i \le m$. This preprocessing phase takes $\mathcal{O}(mn)$ in time and space.

The main greedy loop iterates until the solution exhausts the budget for the robot (Line 4). At each step, the vertex $v_{i,j} \in V$ with the largest value $R[i,j]$ in M (Line 5) is selected. If there is enough budget to visit such a vertex, $v_{i,j}$ is added to the current solution S along with all the vertices on its left side (Line 7) on row i. Note that no other vertices in row i can be inserted later and the size of M is decreased by one. If the budget condition (Line 6) is not satisfied, it implies that there is not enough budget to include $v_{i,j}$ in S, and hence $v_{i,j}$ is discarded. In this case, notice that not even the elements on the right of j can be reached. Thus, the remaining largest cumulative reward $R[i,j-1]$ in row i, which is associated with vertex $v_{i,j-1}$ on the left of the vertex just discarded, is inserted into M (Line 9).

When the budget is exhausted, the total reward $\mathcal{R}(S)$ is returned (Line 10).

The main loop (Line 3) is executed at most $\frac{B}{2}$ times. The cost for extracting the maximum element of a heap or inserting a new element into a heap of size m is $\mathcal{O}(\log(m))$, and the time for updating S and computing $\mathcal{R}(S)$ is bounded by the size of S which is $\mathcal{O}(B)$. Thus, the total time complexity is $\mathcal{O}(mn + B\log(m))$ and space complexity is $\mathcal{O}(mn + m)$.

Example. For the example in Figure 3 and a budget $B = 16$, the vertex with the largest cumulative reward within the available budget is $v_{3,5}$ with an overall cumulative 28. Then, the next vertex is $v_{1,3}$ with overall value of 4. After that, no any other vertex can be chosen and the total obtained reward is 32 and $S = \{v_{1,1}, v_{1,2}, v_{1,3}, v_{3,1}, v_{3,2}, v_{3,3}, v_{3,4}, v_{3,5}\}$. In this case, GDYMC returns the optimal reward.

C. The APXMRE Algorithm

The Approximate Max Ratio Element (APXMRE) algorithm selects, at each step, the feasible vertex that maximizes the ratio between its reward and the cost (budget) to add the selected vertex to the current greedy solution. In addition to the greedy solution, APXMRE also builds a second solution that contains the item with the maximum reward reachable with the budget B. The algorithm returns the maximum between the two solutions. We will prove in Section VI that APXMRE guarantees an approximation ratio of $\frac{1}{2}(1 - \frac{1}{e})$.

Algorithm 3 initializes two solutions S_1 and S_2, and $r_{\max}$ which stores the maximum current reward (Line 1). Here S_1 is built in the preprocessing phase by checking if there is enough budget to reach each vertex from the home $v_{1,1}$ and go back. To compute S_2, a two-dimensional matrix H of pointers is constructed (Line 2). Each pointer $H[i,d]$ points to a vector of length n. In the preprocessing phase (Line 3), the vectors pointed by $H[i,d]$, for $1 \leq i \leq m$ and $0 \leq d \leq i$, are initialized. In Line 6, the vector of n positions pointed by $H[i,d]$ stores in position j, for $1 \leq j \leq n$, the ratio between the reward $\mathcal{R}(v_{i,j})$ and the length $2(j-1) + 2d$ of the cycle from vertex $v_{i-d,1}$ to $v_{i,j}$. Such a cycle consists of a vertical subpath of length d and a horizontal subpath of length $j-1$.

In other words, the vector pointed by $H[i,d]$ stores the cost of reaching the vertices in row i when the farthest row already visited is row $i_{\max} = i - d$. In this case, to reach a vertex of row i we have to consider extra d vertical steps. While

---

Algorithm 3: APXMRE (Graph A, Budget B)

1   $S_1 \leftarrow \emptyset, r_{\max} \leftarrow -\infty, S_2 \leftarrow v_{1,1}$
2   $H \leftarrow [i,d][j], 1 \leq i \leq m, 0 \leq d \leq i - 1, 1 \leq j \leq n$
3   **for** $i \leftarrow 1, m$ **do**
4   **for** $d \leftarrow 0, i - 1$ **do**
5   **for** $j \leftarrow 1, n$ **do**
6        $H[i,d][j] \leftarrow \frac{\mathcal{R}(v_{i,j})}{2(j-1)+2d}$
7   **if** $2(j-1) + 2d \leq B$ **then**
8   **if** $\mathcal{R}(v_{i,j}) > r_{\max}$ **then**
9          $r_{\max} \leftarrow \mathcal{R}(v_{i,j})$
10          $S_1 \leftarrow v_{i,j}$

11   $i_{\max} \leftarrow 1$
12   **while** $\mathcal{C}(S_2) \leq B$ **do**
13     $M \leftarrow [i], 1 \leq i \leq m$
14   **for** $i \leftarrow 1, m$ **do**
15   **if** $i \leq i_{\max}$ **then**
16        $M[i] \leftarrow \text{MAX}(H[i,0])$
17   **else**
18        $M[i] \leftarrow \text{MAX}(H[i, i - i_{\max}])$
19     $v_{i,j} \leftarrow \text{ARGMAX}(M)$
20   **if** $\mathcal{C}(S_2 \cup v_{i,j}) \leq B$ **then**
21       $S_2 \leftarrow S_2 \cup_{1 \leq k \leq j} v_{i,k}$
22   **for** $k \leftarrow j + 1, n$ **do**
23        $H[i,0][k] \leftarrow \frac{\mathcal{R}(v_{i,j})}{2(j-k)}$
24     $i_{\max} \leftarrow \text{MAX}(i, i_{\max})$
25   **return** $\max\{\mathcal{R}(S_1), \mathcal{R}(S_2)\}$

S_2 grows, the farthest row in S_2 increases and the cost for reaching the vertices of row i changes (see Figure 5). The ascending order of the ratios (and in particular the maximum) is however not preserved when the traveling cost (i.e., the ratio denominator) changes due to the change of the last row $i_{\max}$ of the solution. Therefore, for each row i, $1 \leq i \leq m$, we precompute the ratios considering all the possible distances $H[i,d]$, $0 \leq i \leq d$. This preprocessing phase takes $\mathcal{O}(m^2 n)$ in time and space.

After the preprocessing procedure (loop Line 3), the current furthest visited row $i_{\max}$ (Line 11) is set to 1 and the main loop starts (Line 12). A vector M of length m is created (Line 13) using for each row the root of $H[i, i - 1]$. Then vertex $v_{i,j}$ in M with the largest ratio is greedily chosen (Line 19) and inserted in S_2 if feasible. The current solution set S_2 is updated including the vertices in row i on the left of $v_{i,j}$ (Line 21). Moreover, for any vertex in row i on the right of the selected vertex, i.e., $v_{i,k}$ for $j + 1 \leq k \leq n$, the ratios are updated in $H[i,0]$ (Line 23). This is because S_2 already reached the vertex $v_{i,j}$ in row i, and only a small horizontal distance has to be traversed to reach any remaining vertex in row i. Finally, the value of the maximum visited row $i_{\max}$ is updated (Line 24).

Before proceeding to the next vertex selection, for each vertex $v_{i,j}$ we reason about the budget to be spent for the vertical movements. Indeed, if $i \leq i_{\max}$ (Line 16), given the solution S_2 already reaches i, the actual ratios for row i are pointed by $H[i,0]$ which considers only the horizontal distances to reach vertices $v_{i,j}$ (Line 18). If $i > i_{\max}$, some budget for the vertical steps has to be spent since row i has

never been reached. Hence, the actual ratios for row i are pointed by $H[i, i - i_{\max}]$ which considers $i - i_{\max}$ vertical distance. In practice, since the last row in S_2 varies, we change the costs of reaching the vertices, as explained in Figure 5.

At the very end, the algorithm returns the best solution between subsets S_1 and S_2 (Line 25).

The main cycle (Line 12) is executed at most $\frac{B}{2}$ times, resulting in $\mathcal{O}(B(m+n))$ time because the cycle (Line 14) costs only $\mathcal{O}(m)$ time. Thus the overall time and space complexity are $\mathcal{O}(m^2 n + B(m + n))$ and $\mathcal{O}(m^2 n)$, respectively.

Example. For the example in Figure 3 and a budget $B = 16$, the vertex with the largest ratio between its reward and its cost for adding it at the solution set S is $v_{1,2}$ with ratio 1.5 (i.e., reward 3 and cost 2). The next vertex selected is $v_{2,4}$ with ratio $9/8 = 1.125$. The cost is 8 because in the current solution there are only vertices belonging to the first row, so the costs for exploring the second row take into account two more vertical units. Now, the new vertex to add is $v_{2,5}$ with ratio $6/2 = 3$. This is due to the fact that 6 is its reward, but only 2 is the additional cost for visiting it, since $v_{2,4}$ already belongs to S. The next possible vertex is $v_{1,4}$, and hence the solution has total reward 25 and $S = \{v_{1,1}, v_{1,2}, v_{1,3}, v_{1,4}, v_{2,1}, v_{2,2}, v_{2,3}, v_{2,4}, v_{2,5}\}$.

D. The APXMRC Algorithm

The Approximate Max Ratio Cumulative (APXMRC) algorithm is a greedy strategy that runs similar to APXMRE algorithm. The only difference in the greedy rule is that the subvectors (Algorithm 3, Line 6) consider, at the numerator, the cumulative reward instead of the single vertex reward, i.e.,

$$H[i, d][j] \leftarrow \frac{\sum_{k=1}^{j} \mathcal{R}(v_{i,k})}{2(j - 1) + 2d}.$$

Accordingly, APXMRC has the same time and space complexity as APXMRE.

Example. For the example in Figure 3 and a budget $B = 16$, the selected vertex with the maximum ratio between the cumulative reward and the cost for reaching it is $v_{3,5}$ with ratio $28/12 = 2.333$. Then, the next one is $v_{1,2}$ with ratio $3/2 = 1.5$. Finally, the next one is $v_{1,3}$ with ratio $1/2 = 0.5$. This is due to the fact the solution already had included the vertex $v_{1,2}$. In conclusion, the solution has total reward 32 and $S = \{v_{1,1}, v_{1,2}, v_{1,3}, v_{3,1}, v_{3,2}, v_{3,3}, v_{3,4}, v_{3,5}\}$. In this case, APXMRE returns the optimal reward.

Comparison of Algorithms

In Table I we summarize the time and space complexity of all the algorithms we presented, distinguishing between the time complexity of the preprocessing phase and the main algorithm phase. Recalling that B is upper bounded by $\mathcal{O}(mn)$, when $B = \Theta(mn)$, OPTSA requires $\mathcal{O}(\max\{m, n\}^4)$. Thus, all our non-optimal algorithms are faster than OPTSA, even including the preprocessing time. The most expensive greedy algorithms are APXMRE and APXMRC requiring $\mathcal{O}(\max\{m, n\}^3)$ operations. The time complexity of APXMRE and APXMRC is dominated by the

preprocessing time which is larger than that of OPTSA. Algorithms GDYME and GDYMC are always faster than OPTSA, requiring $\mathcal{O}(\max\{m, n\}^2 \log m)$ time, but they slightly lose in reward-performance. In terms of space, GDYME and GDYMC are more efficient than OPTSA which requires the same space as APXMRE and APXMRC.

TABLE I
COMPARISON BETWEEN THE ALGORITHMS.

Algorithm	Time complexity		Space complexity
	Preprocessing	Main	
OPTSA	$\mathcal{O}(mn)$	$\mathcal{O}(Bmn)$	$\mathcal{O}(mn + Bm)$
GDYME	$\mathcal{O}(mn)$	$\mathcal{O}(B \log(mn))$	$\mathcal{O}(mn)$
GDYMC	$\mathcal{O}(mn)$	$\mathcal{O}(B \log(m))$	$\mathcal{O}(mn + m)$
APXMRE	$\mathcal{O}(m^2 n)$	$\mathcal{O}(B(m + n))$	$\mathcal{O}(m^2 n)$
APXMRC	$\mathcal{O}(m^2 n)$	$\mathcal{O}(B(m + n))$	$\mathcal{O}(m^2 n)$

VI. GUARANTEED BOUND FOR APPROXIMATION RATIO

In this section, we analyze the performance guarantee of our approximation algorithms. Specifically, we initially highlight how the selected element by the greedy step improves the objective function in Lemma 1, then we show how the partial greedy solution is bounded with respect to the optimal solution in Lemma 2, and finally we prove the guaranteed approximation bound of algorithms APXMRE and APXMRC exploiting the two aforementioned Lemmas in Theorem 1.

Let G_i be the solution obtained from APXMRE at step i. Moreover, at each iteration i, APXMRE adds to the solution set G_{i-1} the element x_i that maximizes the ratio between the marginal gain and the marginal cost of adding such an element (see Algorithm 3, Line 19). That is,

$$x_i = \arg\max_{v_{i,j} \in V} \frac{\mathcal{R}(G_{i-1} \cup v_{i,j}) - \mathcal{R}(G_{i-1})}{\mathcal{C}(G_{i-1} \cup v_{i,j}) - \mathcal{C}(G_{i-1})} \quad (5)$$

For our problem $\mathcal{R}(G_{i-1} \cup v_{i,j}) - \mathcal{R}(G_{i-1}) = \mathcal{R}(x_i)$, due to the modularity property. In Lemma 1 we prove that the inclusion of the element selected by the greedy step (Eq. (5)) improves the objective function by at least a quantity proportional to the current distance to the optimum. Namely, we are able to bound the improvement of the greedy step from below the improvement on the objective function. Let the solution $G_{\ell+1}$ from APXMRE the first iteration when it violates the budget constraint and stops.

Lemma 1. *For any $i = 1, \ldots, \ell + 1$, it holds that*

$$\mathcal{R}(G_i) - \mathcal{R}(G_{i-1}) \geq \frac{\mathcal{R}(\mathrm{OPT}) - \mathcal{R}(G_{i-1})}{B}(\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})),$$

where OPT *is the optimal solution considering budget B.*

Proof. Suppose OPT is the optimal solution considering budget B. For any $i = 1, \ldots, \ell + 1$ we have

$$\mathcal{R}(\mathrm{OPT}) - \mathcal{R}(G_{i-1}) \leq \mathcal{R}(\mathrm{OPT} \cup G_{i-1}) - \mathcal{R}(G_{i-1})$$
$$= \mathcal{R}(\mathrm{OPT} \setminus G_{i-1} \cup G_{i-1}) - \mathcal{R}(G_{i-1}).$$

Assume $\mathrm{OPT} \setminus G_{i-1} = \{y_1, \ldots, y_m\}$, where y_i are vertices in OPT but not in solution G_{i-1}. Let us now define for any

$j = 1, \ldots, m$ the marginal reward for any element y_j as $Z_j = \mathcal{R}(G_{i-1} \cup \{y_1, \ldots, y_j\}) - \mathcal{R}(G_{i-1} \cup \{y_1, \ldots, y_{j-1}\})$. Thus,

$$\mathcal{R}(\text{OPT}) - \mathcal{R}(G_{i-1}) \leq \mathcal{R}(\text{OPT}\backslash G_{i-1} \cup G_{i-1}) - \mathcal{R}(G_{i-1}) =$$
$$\mathcal{R}(G_{i-1} \cup \{y_1, \ldots, y_m\}) - \mathcal{R}(G_{i-1}) =$$
$$\mathcal{R}(G_{i-1} \cup \{y_1, \ldots, y_m\}) - \mathcal{R}(G_{i-1} \cup \{y_1, \ldots, y_{m-1}\}) +$$
$$\mathcal{R}(G_{i-1} \cup \{y_1, \ldots, y_{m-1}\}) -$$
$$\mathcal{R}(G_{i-1} \cup \{y_1, \ldots, y_{m-2}\}) + \ldots - \mathcal{R}(G_{i-1}) = \sum_{j=1}^{m} Z_j.$$

Notice that, for any j,

$$\frac{Z_j}{\mathcal{C}(G_{i-1} \cup y_j) - \mathcal{C}(G_{i-1})} \leq \frac{\mathcal{R}(G_{i-1} \cup y_j) - \mathcal{R}(G_{i-1})}{\mathcal{C}(G_{i-1} \cup y_j) - \mathcal{C}(G_{i-1})}$$
$$\leq \frac{\mathcal{R}(G_i) - \mathcal{R}(G_{i-1})}{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})} \tag{6}$$

where the first inequality holds due to the modularity of $\mathcal{R}$ (precisely, Eq. (1) where $Y = G_{i-1} \cup \{y_1, \ldots, y_j\}$ and $X = G_{i-1} \cup y_j$ and $X \subseteq Y$) and the second inequality holds due to the greedy rule, according to Eq. (5). Therefore,

$$\mathcal{R}(\text{OPT}) - \mathcal{R}(G_{i-1}) \leq \sum_{j=1}^{m} Z_j$$
$$\leq \frac{\mathcal{R}(G_i) - \mathcal{R}(G_{i-1})}{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})} \sum_{j=1}^{m} \mathcal{C}(G_{i-1} \cup y_j) - \mathcal{C}(G_{i-1})$$
$$\leq \frac{\mathcal{R}(G_i) - \mathcal{R}(G_{i-1})}{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})} \sum_{y \in \text{OPT}} \mathcal{C}(y).$$

Since function c can be computed in polynomial time, we can bound the sum of the costs over all the elements with the cost of the optimal solution that at its turn cannot exceed B. Hence,

$$\mathcal{R}(\text{OPT}) - \mathcal{R}(G_{i-1}) \leq \frac{\mathcal{R}(G_i) - \mathcal{R}(G_{i-1})}{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})} \sum_{y \in \text{OPT}} \mathcal{C}(y)$$
$$\leq \mathcal{C}(\text{OPT}) \frac{\mathcal{R}(G_i) - \mathcal{R}(G_{i-1})}{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})} \leq B \frac{\mathcal{R}(G_i) - \mathcal{R}(G_{i-1})}{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})}. \quad \square$$

In the following, Lemma 2 proves that the solution of the greedy algorithm at each iteration can be bounded with respect to the optimal solution.

Lemma 2. *For any $i = 1, \ldots, \ell + 1$ it holds that*

$$\mathcal{R}(G_i) \geq \left[1 - \prod_{k=1}^{i} \left(1 - \frac{\mathcal{C}(G_k) - \mathcal{C}(G_{k-1})}{B}\right)\right] \mathcal{R}(\text{OPT}),$$

where OPT *is the optimal solution considering budget B.*

Proof. For $i = 1$, the proof follows directly from Lemma 1 since $G_0 = \emptyset$, $\mathcal{R}(G_0) = 0$ and $\mathcal{C}(G_0) = 0$. For $i > 1$, using

Lemma 1 and inductive hypothesis on $\mathcal{R}(G_{i-1})$, it holds:

$$\mathcal{R}(G_i) = \mathcal{R}(G_{i-1}) - \mathcal{R}(G_{i-1}) + \mathcal{R}(G_i)$$
$$\geq \mathcal{R}(G_{i-1}) + \frac{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})}{B}(\mathcal{R}(\text{OPT}) - \mathcal{R}(G_{i-1}))$$
$$= \left(1 - \frac{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})}{B}\right) \mathcal{R}(G_{i-1}) +$$
$$\frac{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})}{B} \mathcal{R}(\text{OPT})$$
$$\geq \left(1 - \frac{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})}{B}\right) \cdot$$
$$\left[1 - \prod_{k=1}^{i-1} \left(1 - \frac{\mathcal{C}(G_k) - \mathcal{C}(G_{k-1})}{B}\right)\right] \mathcal{R}(\text{OPT}) +$$
$$\frac{\mathcal{C}(G_i) - \mathcal{C}(G_{i-1})}{B} \mathcal{R}(\text{OPT})$$
$$= \left(1 - \prod_{k=1}^{i} \left(1 - \frac{\mathcal{C}(G_k) - \mathcal{C}(G_{k-1})}{B}\right)\right) \mathcal{R}(\text{OPT}). \quad \square$$

Finally, exploiting Lemma 2, we prove that APXMRE provides a constant approximation ratio.

Theorem 2. *Algorithm* APXMRE *guarantees a $\frac{1}{2}\left(1 - \frac{1}{e}\right)$ approximation ratio for OASP.*

Proof. By applying Lemma 2, by the fact that $\mathcal{C}(G_{\ell+1}) > B$ (since it violates the constraint), and recalling that for $a_1, \ldots, a_n \in \mathbb{R}$ such that $\sum_{i=1}^{n} a_i = A$, the function $(1 - \Pi_{i=1}^{n}(1 - \frac{a_i}{A}))$ achieves its minimum at $a_1 = \ldots = a_n = \frac{A}{n}$, we have:

$$\mathcal{R}(G_{\ell+1}) \geq \left[1 - \prod_{k=1}^{\ell+1} \left(1 - \frac{\mathcal{C}(G_k) - \mathcal{C}(G_{k-1})}{B}\right)\right] \mathcal{R}(\text{OPT})$$
$$\geq \left[1 - \prod_{k=1}^{\ell+1} \left(1 - \frac{\mathcal{C}(G_k) - \mathcal{C}(G_{k-1})}{\mathcal{C}(G_{\ell+1})}\right)\right] \mathcal{R}(\text{OPT})$$
$$\geq \left[1 - \left(1 - \frac{1}{(\ell+1)}\right)^{\ell+1}\right] \mathcal{R}(\text{OPT}) \geq \left(1 - \frac{1}{e}\right) \mathcal{R}(\text{OPT}).$$

Since r is a modular function we know that the marginal gain selecting element $x_{\ell+1}$ at step $\ell + 1$ is less than or equal to the gain given by a solution that contains the element with maximum gain, i.e., $\mathcal{R}(G_{\ell+1}) - \mathcal{R}(G_\ell) = \mathcal{R}(x_{\ell+1}) \leq \mathcal{R}(x^*)$, where $\mathcal{R}(x_{\ell+1})$ is the marginal gain obtained adding the $(\ell+1)$-th vertex selected by greedy rule and $x^* = \arg\max_{x \in V : \mathcal{C}(x) \leq B} \mathcal{R}(x)$, (i.e., x^* is the vertex with the maximum reward reachable with budget B), it follows:

$$\mathcal{R}(G_\ell) + \mathcal{R}(x^*) \geq \mathcal{R}(G_{\ell+1}) \geq \left(1 - \frac{1}{e}\right) \mathcal{R}(\text{OPT}), \text{ and}$$

$$\max\{\mathcal{R}(G_\ell), \mathcal{R}(x^*)\} \geq \frac{1}{2}\left(1 - \frac{1}{e}\right) \mathcal{R}(\text{OPT}). \quad \square$$

In Algorithm 3, $x^* = S_1$ and $\mathcal{R}(S_1) = \mathcal{R}(x^*)$. Moreover, $\mathcal{R}(S_2) \geq \mathcal{R}(G_\ell)$ because S_2 contains not only the vertex selected by the greedy rule $\{x_1, \ldots, x_\ell\}$, but also the vertices on its left on the same row (see Algorithm 3, Line 21).

It is worth mentioning that the same approximation ratio also holds for APXMRC. Lemma 1 holds even when the

greedy algorithm picks a subset of vertices in a row. APXMRC considers the cumulative reward of a partial row, that is not already in the solution, up to the vertex in consideration. The cumulative reward $\mathcal{R}(G_i \cup y_j)$ for any j is greater than or equal to that obtained by picking only the last element y_j, whereas the cost $\mathcal{C}(G_i \cup y_j)$ remains the same since the cost for visiting only the last element y_j or all the elements up to y_j, does not change. Therefore, the second inequality in Eq. (6) of Lemma 1 remains true even if $G_i \setminus G_{i-1}$ is a subset of vertices in a row of the graph. So, repeating the reasoning of Lemma 2 and Theorem 2, we prove that algorithm APXMRC guarantees the same approximation ratio as APXMRE.

VII. PERFORMANCE EVALUATION

We evaluate the performance of the new algorithms {GDYME, GDYMC, APXMRE, APXMRC}, and compare them with the optimal algorithm OPTSA. For each scenario, each algorithm is tested with an increasing budget $B = 2n, \ldots, 2(mn + m)$, and we plot the average of the results on 30 instances (10 in the case of real data) along with their 95% confidence interval. Specifically, in Figures 7 and 8, organized in paired plots, we report:

- on the left plot of each pair, the collected reward $\mathcal{R}$ in percentage (100% means all the available rewards have been collected), with respect to the used budget B in percentage (100% means $B = 2(mn + m)$);
- on the right plot of each pair, the ratio $\rho = \frac{\mathcal{R}(\text{ALG})}{\mathcal{R}(\text{OPTSA})}$ between the reward of ALG = {GDYME, GDYMC, APXMRE, APXMRC } and the reward of OPTSA. Recall OPTSA optimally solves OASP.

(a) $\theta = 0$. (b) $\theta = 0.9$. (c) $\theta = 1.8$. (d) $\theta = 2.7$. (e) Real.
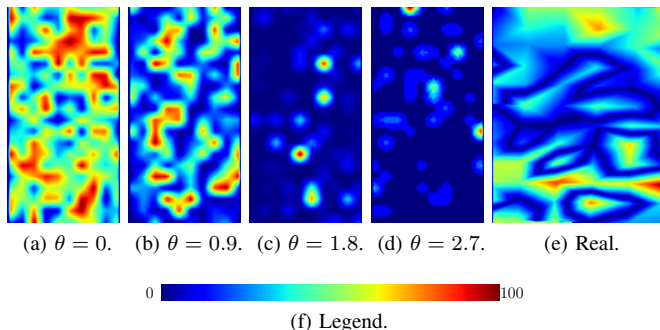
0 [====] 100

(f) Legend.

Fig. 6. Reward maps for synthetic (a)–(d), and real (e) graph instances. Synthetic graphs are $A(100, 50)$ while real graph is $A(274, 214)$. The heat-maps, rendered after a *bilinear interpolation* using matplotlib software, highlight the areas with large and small rewards with hot and cold colors, respectively (see legend (f)). Synthetic data are randomly generated, while real data are taken from real observations [39]. In particular, real moisture levels are then converted in actual rewards through a suitable interpolation [40].

A. On Synthetic Data

In this section, we evaluate the performance of the presented algorithms on generated synthetic data. We assume the rewards are properly randomized following the Zipf distribution [41], which is characterized by a single parameter θ that rules the occurrences of the reward values. In our evaluations, the rewards are integers in the interval $[0, 100]$. When $\theta = 0$,

the rewards are uniformly distributed in $[0, 100]$, while when θ increases, the smallest rewards become more and more frequent than the largest ones. In our setting, we assume $\theta = \{0, 0.9, 1.8, 2.7\}$.

Examples of random reward maps are illustrated in Figures 6a–6d, where the hot and cold colors represent high and low rewards, respectively. In our applications mentioned at the beginning, a hot color represents a place where the robot has to operate on a plant or pick an item from the shelves with a high reward (high priority), while a cold color represents a location with a low reward (low priority). When $\theta = 0$ (Figure 6a), the heat map illustrates rewards uniformly distributed, while when $\theta = 1.8$ (Figure 6c), the heat map features highly unbalanced rewards, with very dispersed positions with high reward. In our experiments, we finally vary the size of the aisle-graphs considering two different layouts: more rows than columns ($m > n$, $A(100, 50)$) and more columns than rows ($n > m$, $A(50, 100)$).

For each A and θ, we run the five algorithms. Fixing m, n, and θ, we generate 30 different reward distributions and solve them with each algorithm. For each random distribution, each algorithm is tested with an increasing budget. Since the overall reward is randomly generated and differs for each graph, we return the reward gained as a percentage of the overall reward of each graph. Finally, we plot the average of the results along with their 95% confidence interval.

Figure 7 compares the reward collected by the different algorithms for several values of θ. When $\theta = 0$, the percentage of the collected reward almost increases linearly with the budget; this is reasonable since the rewards are uniformly distributed, and so also the gain for unit of cost is uniformly distributed. There are no subareas of the map that are more important to visit because their reward per unit is larger.

When $\theta \geq 1.9$, there are instead localized subareas with higher rewards. In such cases, the robot prioritizes those areas (hot areas of the heat maps) and collects a high percentage of reward (up to 60% when $\theta = 1.9$ and up to 80% when $\theta = 2.7$) already with a small budget $B \approx 20\%$. Up to $B \approx 60\%$, the value of θ makes the difference in the percentage of the collected reward. With $B \approx 80\%$, instead, θ does not meaningfully impact the reward. This means that when $B \geq 80\%$, there is enough budget to cover the most meaningful part of the map in any case, and it is not important to first select the more promising paths.

When $\theta \leq 0.8$, the collected reward highly depends on the algorithms for any value of $B \leq 60\%$, showing that the strategy followed by the algorithms is important. In principle, there are no paths that are definitely more promising than others and it is difficult to decide which one is best. When $m > n$, the large number m of rows makes the decision even harder. Finally, all the algorithms report more or less the same rewards (all the curves are near to each other), independently of the θ value when $B \geq 80\%$. In this case, the larger budget balances the algorithm cleverness.

To confirm that the problem difficulty increases with the value of m, observe that the best performing Algorithm APXMRC performs better when $m < n$ than when $m > n$. The collected reward by APXMRC (solid red line) almost
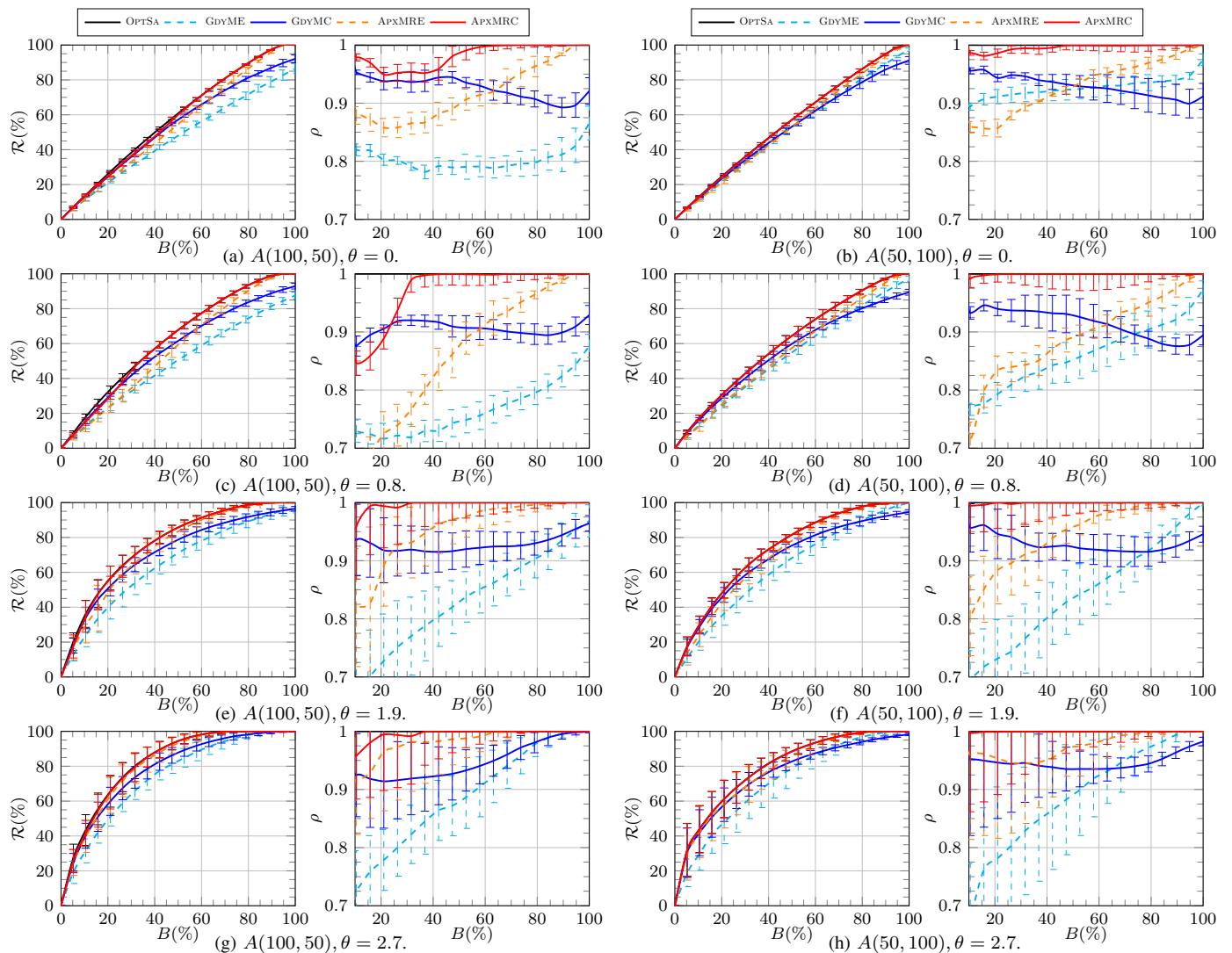
Fig. 7. Performance comparison of the proposed GDYME, GDYMC, APXMRE, APXMRC, and OPTSA algorithms.

coincides with that of the optimal Algorithm OPTSA (solid black line) for the scenarios with $m < n$. In fact, the ratio $\rho = \frac{\mathcal{R}(\text{APXMRC})}{\mathcal{R}(\text{OPTSA})} \to 1$ when $m < n$ for any value of θ (e.g., Figure 7b), while when $m > n$ the ratio $\rho \to 1$ only for higher values of θ (e.g., Figure 7g). When θ is small and m is large, APXMRC does not find the optimal solution, which instead is computed by OPTSA. Note that APXMRC always outperforms APXMRE (dashed red line). The ratio between APXMRE and OPTSA is well above 0.9 (but below the ratio of APXMRC) when $B \geq 40\%$, and around 0.8 otherwise. This is because APXMRE greedily adds the best vertex $v_{i,j}$ having the maximum ratio between its reward and the cost for reaching it, without including the reward of the vertices it passes through in its selection criterion. Instead, APXMRC considers also the reward of such vertices for the selection criterion. Similarly, the reward collected from GDYMC (solid blue line) is greater than that of GDYME (dashed blue line). In all plots, the ratio ρ between GDYMC and OPTSA is above 0.9. Instead, the ratio between GDYME and OPTSA increases with B from 0.7 to 0.9, and more. When B is small, GDYME

performs much worse than GDYMC because it can spend a lot of budget to cover isolated high reward items. When $B \geq 80\%$ and $m < n$, GDYME outperforms GDYMC probably because the single reward becomes the most important selection criterion to distinguish among important and less important items.

In almost all the plots, up to $B = 40\%$, GDYMC outperforms APXMRE, that is, the cumulative reward beats the criteria that consider the cost. It seems that GDYMC gives the precedence to the cluster of items with high reward independently of how far they are, while APXMRE can prefer the closest item even if is has small reward. Namely, a small cost at the denominator of the selection criterion amplifies minimal rewards at the numerator for APXMRE. This behavior of GDYMC is particularly important because GDYMC is computationally more efficient in time and space than APXMRE and APXMRC. So when the budget is limited and the computational efficiency matters, GDYMC is the best algorithm to use.

Lastly, it can be experimentally seen that the reward achieved by APXMRE and APXMRC is much larger than the

guaranteed reward. In our experiments, both APXMRE and APXMRC collect more than 90% of the optimum reward, i.e., the ratio $\rho \geq 0.9$, while the guaranteed approximation ratio proved in Section VI is ≈ 0.32.

B. On Real Data

To further asses the effectiveness of the proposed algorithms, in this section we evaluate their performance on a set of instances coming from real data in a vineyard irrigation scenario. These data sets were obtained from a large scale commercial vineyard located in central California [39]. The section we consider has 274 rows and 214 columns (i.e., aisle-graph $A(274, 214)$), resulting in approximately $60,000$ vines in the aisle-graph. For each (internal) vertex $v \in V$, the reward is set to $\mathcal{R}(v) = |T - m(v)|$, where T is a constant indicating the desired soil moisture in the vineyard (provided by a human expert), and $m(v)$ is the soil moisture at vertex v. This reward is the difference between desired moisture and actual moisture, thus revealing how underwatered or overwatered a vine is. Due to the large size of the ranch, soil moisture values were manually sampled at discrete locations using a probe equipped with a GPS. From the finite set of samples, a soil moisture map for the whole block was obtained using the Kriging algorithm for interpolation [40].

Figure 6e shows one of the soil moisture maps used in our experiments on real data. In particular, data were collected every two weeks from this vineyard and used to produce ten soil moisture maps. These maps (e.g., Figure 6e) were then used to test the proposed algorithms, and the results were averaged across each reward map. For reasons of similarity with the synthetic data, we have also tested our results on the transposed soil moisture maps of size 214 rows and 274 columns (i.e., aisle-graph $A(214, 274)$) with $m < n$. This is reasonable since the structure of the constrained aisle graph and the soil moisture are independent of each other.

For each of the ten scenarios, we run GDYME, GDYMC, APXMRE, APXMRC, and OPTSA, with an increasing budget. Finally, we plot the average of the results along with their confidence interval.

Figure 8 compares the reward for different algorithms on the real ten soil moisture maps. The plots in Figure 8 on the left matches the trend already seen in the plots in Figure 7 (synthetic data) when $\theta = 0.8$ because the collected reward $\mathcal{R}$ when $B = 20\%$ is slightly above 20%, when $B = 40\%$ is above 60%, and $B = 80\%$ is slightly above 80%. This is not surprising because in the literature Zipf distributions with parameter $\theta = 0.8$ model several real distributions.

In the right plots of Figure 8, the most interesting difference between the synthetic and the real data happens for algorithm GDYME (dashed blue line)and GDYMC (solid blue line). In the synthetic data, GDYMC outperforms GDYME; while in real data, GDYMC and GDYME perform the same. This means that GDYME, which takes into account for selection only the single maximum reward but adds the rewards of all traversed vertices, performs the same selection as GDYMC, which takes into account for selection the cumulative reward. In this case it appears that the cumulative reward is dominated by the largest reward.

The ratio between APXMRC and OPTSA tends to 1 already when $B = 20\%$ for both cases $m > n$ and $m < n$. The performance of the case $m < n$ is more similar to that of the case $m > n$ in the real scenario than in the synthetic one. This is arguably due to the fact that m and n are much closer in relative magnitude, i.e., $\frac{274}{214} = 1.28$, than 100 and 50 whose ratio is $\frac{100}{50} = 2$. The APXMRE algorithm poorly performs when $B \leq 40\%$, as it happens for the synthetic data only when $m > n$. In contrast, when $m < n$, APXMRE is always above GDYMC, which is the same as GDYME. This can be explained imagining the transpose of Figure 6e: all the rows look quite similar among them and so it is more profitable to collect the reward from the closest rows. In conclusion, the performance of the proposed algorithms on the real scenario is well above the performance guaranteed for every percentage of B.
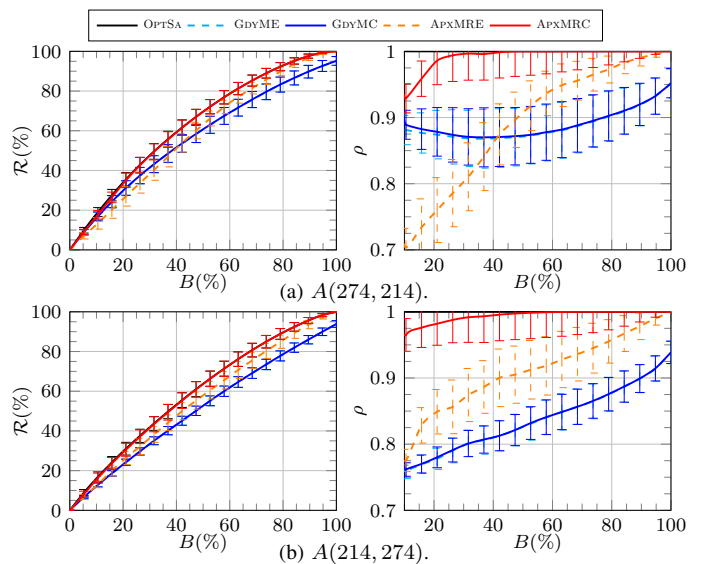

Fig. 8. Performance comparison of the proposed algorithms on real graphs.

VIII. CONCLUSIONS

In this paper, we focused on OASP, a route planning problem in constrained aisle-graphs with single access subject to a given budget, and aiming to maximize an assigned reward. We first proposed a polynomial-time algorithm to optimally solve OASP. This solution, although optimal, has a computational cost that can become problematic when the size of the aisle-graph grows. Thus, we proposed four simpler and faster greedy algorithms with reduced computational cost in time and space. For two of them, we exploit submodularity properties and achieve a provably guaranteed approximation ratio of $\frac{1}{2}(1 - \frac{1}{e})$. Finally, we extensively evaluated the new four algorithms with respect to the optimal solution on synthetic and real data. As future work, it would be worth to apply similar approaches to different topologies for the input graph. Different applications as well as different robot capabilities may lead to interesting scenarios. We also extend our investigation to the case where swarms of robots can jointly accomplish tasks.

References

[1] T. C. Thayer, S. Vougioukas, K. Goldberg, and S. Carpin, "Routing algorithms for robot assisted precision irrigation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2221–2228.

[2] J. Xiong, Z. Liu, R. Lin, R. Bu, Z. He, Z. Yang, and C. Liang, "Green grape detection and picking-point calculation in a night-time natural environment using a charge-coupled device (ccd) vision sensor with artificial illumination," *Sensors*, vol. 18, no. 4, p. 969, 2018.

[3] T. Botterill, S. Paulin, R. Green, S. Williams, J. Lin, V. Saxton, S. Mills, X. Chen, and S. Corbett-Davies, "A robot system for pruning grape vines," *Journal of Field Robotics*, vol. 34, no. 6, pp. 1100–1122, 2017.

[4] Mecalux, *Conventional pallet racking*, 2021, https://www.mecalux.com/pallet-racks/conventional-pallet-racking.

[5] B. L. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Research Logistics (NRL)*, vol. 34, no. 3, pp. 307–318, 1987.

[6] F. Betti Sorbelli, S. Carpin, F. Corò, A. Navarra, and C. M. Pinotti, "Optimal routing schedules for robots operating in aisle-structures," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4927–4933.

[7] F. Betti Sorbelli, F. Coro, S. K. Das, A. Navarra, and C. M. Pinotti, "Speeding-up routing schedules on aisle-graphs," in *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2020, pp. 69–76.

[8] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala, "New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen," *SIAM Journal on computing*, vol. 28, no. 1, pp. 254–262, 1998.

[9] B. Aghezzaf and H. E. Fahim, "Iterated local search algorithm for solving the orienteering problem with soft time windows," *SpringerPlus*, vol. 5, no. 1, p. 1781, 2016.

[10] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff, "Approximation algorithms for orienteering and discounted-reward tsp," *SIAM Journal on Comp.*, vol. 37, no. 2, pp. 653–670, 2007.

[11] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications," *European Journal of Op. Res.*, vol. 255, no. 2, pp. 315–332, 2016.

[12] M. Fischetti, J. J. S. Gonzalez, and P. Toth, "Solving the orienteering problem through branch-and-cut," *INFORMS Journal on Computing*, vol. 10, no. 2, pp. 133–148, 1998.

[13] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff, "Approximation algorithms for orienteering and discounted-reward tsp," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, 2003, pp. 46–55.

[14] C. Chekuri, N. Korula, and M. Pál, "Improved algorithms for orienteering and related problems," *ACM Transactions on Algorithms*, vol. 8, no. 3, Jul. 2012.

[15] K. Chen and S. Har-Peled, "The orienteering problem in the plane revisited," in *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, 2006, p. 247–254.

[16] T. Thayer, S. Vougioukas, K. Goldberg, and S. Carpin, "Multi-robot routing algorithms for robots operating in vineyards," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 3, pp. 1184–1194, 2020.

[17] M. Mann, B. Zion, D. Rubinstein, R. Linker, and I. Shmulevich, "The orienteering problem with time windows applied to robotic melon harvesting," *Journal of Optimization Theory and Applications*, vol. 168, no. 1, pp. 246–267, 2016.

[18] D. Thakur, M. Likhachev, J. Keller, V. Dobrokhodov, K. Jones, J. Wurz, and I. Kaminer, "Planning for opportunistic surveillance with multiple robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.*, 11 2013, pp. 5750–5757.

[19] J. Yu, M. Schwager, and D. Rus, "Correlated orienteering problem and its application to persistent monitoring tasks," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1106–1118, 2016.

[20] G. Best and G. A. Hollinger, "Decentralised self-organising maps for the online orienteering problem with neighbourhoods," in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2019, pp. 139–141.

[21] S. Jorgensen, R. H. Chen, M. B. Milam, and M. Pavone, "The team surviving orienteers problem: routing teams of robots in uncertain environments with survival constraints," *Autonomous Robots*, vol. 42, no. 4, pp. 927–952, 2018.

[22] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions-I," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

[23] S. Jorgensen, R. H. Chen, M. B. Milam, and M. Pavone, "The matroid team surviving orienteers problem: Constrained routing of heterogeneous teams with risky traversal," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5622–5629.

[24] W. Xu, Z. Xu, J. Peng, W. Liang, T. Liu, X. Jia, and S. K. Das, "Approximation algorithms for the team orienteering problem," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1389–1398.

[25] W. Xu, W. Liang, Z. Xu, J. Peng, D. Peng, T. Liu, X. Jia, and S. K. Das, "Approximation algorithms for the generalized team orienteering problem and its application," *IEEE/ACM Transactions on Networking (to appear)*, 2020.

[26] M. Roberts, D. Dey, A. Truong, S. Sinha, S. Shah, A. Kapoor, P. Hanrahan, and N. Joshi, "Submodular trajectory optimization for aerial 3d scanning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5324–5333.

[27] R. Ghuge and V. Nagarajan, "Quasi-polynomial algorithms for submodular tree orienteering and other directed network design problems," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020, pp. 1039–1048.

[28] G. Shi, L. Zhou, and P. Tokekar, "Robust multiple-path orienteering problem: Securing against adversarial attacks," *arXiv preprint arXiv:2003.13896*, 2020.

[29] T. C. Thayer, S. Vougioukas, K. Goldberg, and S. Carpin, "Multi-robot routing algorithms for robots operating in vineyards," in *Proceedings of the International Conference on Automation Science and Engineering*, 2018, pp. 7–14.

[30] H. D. Ratliff and A. S. Rosenthal, "Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem," *Operations research*, vol. 31, no. 3, pp. 507–521, 1983.

[31] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[32] H. Cambazard and N. Catusse, "Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane," *European Journal of Operational Research*, vol. 270, no. 2, pp. 419–429, 2018.

[33] F. Betti Sorbelli, F. Corò, C. M. Pinotti, and A. Shende, "Automated picking system employing a drone," in *15th International Conference on Distributed Computing in Sensor Systems, DCOSS 2019, Santorini, Greece, May 29-31, 2019*, 2019, pp. 633–640.

[34] G. Cornnejols, M. Fisher, and G. Nemhauser, "Location of bank accounts of optimize float: An analytic study of exact and approximate algorithm," *Management Science*, vol. 23, pp. 789–810, 1977.

[35] M. Conforti and G. Cornuéjols, "Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the rado-edmonds theorem," *Discrete applied mathematics*, vol. 7, no. 3, pp. 251–274, 1984.

[36] M. Sviridenko and J. Ward, "Tight bounds for submodular and supermodular optimization with bounded curvature," *arXiv preprint arXiv:1311.4728*, 2013.

[37] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Information processing letters*, vol. 70, no. 1, pp. 39–45, 1999.

[38] A. Krause and C. Guestrin, *A note on the budgeted maximization of submodular functions*. Carnegie Mellon University. Center for Automated Learning and Discovery, 2005.

[39] T. C. Thayer, S. Vougioukas, K. Goldberg, and S. Carpin, "Multi-robot routing algorithms for robots operating in vineyards," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2018, pp. 14–21.

[40] M. A. Oliver and R. Webster, "Kriging: a method of interpolation for geographical information systems," *International Journal of Geographical Information System*, vol. 4, no. 3, pp. 313–332, 1990.

[41] C. Tullo and J. Hurford, "Modelling zipfian distributions in language," in *Proceedings of language evolution and computation workshop/course at ESSLLI*, 2003, pp. 62–75.

Francesco Betti Sorbelli received the Bachelor and Master degrees *cum laude* in Computer Science from the University of Perugia, Italy, in 2007 and 2010, respectively, and his Ph.D. in Computer Science from the University of Florence, Italy, in 2018. He was a postdoc researcher at University of Perugia in 2018 under the supervision of Prof. Cristina M. Pinotti. Currently, he is a postdoc at the Missouri University of Science and Technology University, USA, under the supervision of Prof. Sajal K. Das. His research interests include algorithms design, combinatorial optimization, unmanned vehicles.

Alfredo Navarra is Associate Professor since 2015 at the Mathematics and Computer Science Dept, University of Perugia, Italy. He received his Ph.D. in Computer Science in 2004 from "Sapienza" University of Rome. Before joining the University of Perugia in 2007, he has been with various international research institutes like the INRIA of Sophia Antipolis, France; the Dept of Computer Science at the Univ. of L'Aquila, Italy; the LaBRI, Univ. of Bordeaux, France. His research interests include algorithms, computational complexity, distributed computing and networking.

Stefano Carpin is Professor in the Department of Computer Science and Engineering at the University of California, Merced. He received his *Laurea* and Ph.D. degrees in electrical engineering and computer science from the University of Padova, Italy in 1999 and 2003, respectively. From 2003 to 2006 he held faculty positions with Jacobs University Bremen, Germany. Since 2007 he has been with the School of Engineering at UC Merced, where he established and leads the robotics laboratory. His research interests include mobile and cooperative robotics for service tasks, and robot algorithms. He served as associated editor for the IEEE Transactions on Automation Science and Engineering and for the IEEE Transactions in Robotics and he is the founding chair for the Department of Computer Science and Engineering at the University of California, Merced.

Federico Corò received the Bachelor and Master degrees *cum laude* in Computer Science from the University of Perugia, Italy, in 2014 and 2016, respectively and his Ph.D. in Computer Science in 2019 at Gran Sasso Science Institute (GSSI), L'Aquila, Italy. Currently, he is a postdoc researcher in the Department of Computer Science at La Sapienza in Rome, Italy. His research interests include several aspects of theoretical computer science, including combinatorial optimization, network analysis, and the design and efficient implementation of algorithms.

Cristina M. Pinotti received the Master degree *cum laude* in Computer Science from the University of Pisa, Italy, in 1986. In 1987-1999, she was Researcher with the National Council of Research in Pisa. In 2000-2003, she was Associate Professor at the University of Trento. Since 2004, she is a Full Professor at the University of Perugia. Her current research interests include the design and analysis of algorithms for wireless sensor networks and communication networks.

Sajal K. Das is a professor of computer science and Daniel St. Clair Endowed Chair at Missouri University of Science and Technology. His research interests include wireless sensor networks, mobile and pervasive computing, cyber-physical systems and IoT, smart environments, cloud computing, cyber security, and social networks. He serves as the founding Editor-in-Chief of Elsevier's Pervasive and Mobile Computing journal, and as Associate Editor of several journals including the IEEE Transactions of Mobile Computing, IEEE Transactions on Dependable and Secure Computing, and ACM Transactions on Sensor Networks. He is an IEEE Fellow.