Optimal Routing Schedules for Robots Operating in Aisle-Structures

Francesco Betti Sorbelli, Stefano Carpin, Federico Corò, Alfredo Navarra, and Cristina M. Pinotti

***Abstract*— In this paper, we consider the Constant-cost Orienteering Problem (COP) where a robot, constrained by a limited travel budget, aims at selecting a path with the largest reward in an aisle-graph. The aisle-graph consists of a set of loosely connected rows where the robot can change lane only at either end, but not in the middle. Even when considering this special type of graphs, the orienteering problem is known to be NP-hard. We optimally solve in polynomial time two special cases, COP-FR where the robot can only traverse full rows, and COP-SC where the robot can access the rows only from one side. To solve the general COP, we then apply our special case algorithms as well as a new heuristic that suitably combines them. Despite its light computational complexity and being confined into a very limited class of paths, the optimal solutions for COP-FR turn out to be competitive even for COP in both real and synthetic scenarios. Furthermore, our new heuristic for the general case outperforms state-of-art algorithms, especially for input with highly unbalanced rewards.**

I. Introduction

Numerous robotic tasks can be abstracted using a graph model featuring vertices associated with rewards and edges associated with costs. Vertices usually represent locations that the robots must visit to perform some task, while edges costs represent the energy or time spent while moving between locations. Models like these emerge for example when robots are used for environmental monitoring, surveillance, logistic, urban mobility, and precision agriculture, just to name a few. Owing to the fact that most robots and vehicles have to periodically stop to recharge their batteries or refuel, this type of tasks are naturally connected to the combinatorial optimization problem known in literature as *orienteering*. In orienteering one is given a graph where each vertex has a reward and each edge has a cost. The objective is to determine a path on the graph maximizing the sum of collected rewards while ensuring that the cost of the path does not exceed a given budget. An important feature of this problem is that if a vertex is visited more than once the reward is collected only once, while the cost associated with an edge is incurred every time it is traversed. As pointed out in [1], this optimization problem is computationally hard, and therefore it is of interest to study either heuristic solutions that perform well on large problem instances, or exact solutions when one considers special types of graphs or specific classes of paths. In [1] the orienteering problem was used to solve a routing problem associated with robots used for precision irrigation in vineyards. A vineyard imposes specific motion constraints because a robot operating in it can change row only when it is at either end of the row, but not when it is in the middle. The same motion constraints can be found in warehouses, which are typically organized with long rows of parallel shelves, and crossing in the middle is typically impossible due to stored goods [2]. Fig. 1 illustrates the two scenarios above. These motion constraints can be captured by the *aisle-graph* (a.k.a. *irrigation graph* [1]), formally defined in Sec. II. On such graphs, an associated orienteering problem can be defined to account for the limited battery runtime of the robot. In [1] it was shown that solving the orienteering problem on the special class of aisle-graphs is NP-hard, and two different heuristics were proposed. In this work we propose various improvements to the results presented in [1]. First, we provide an efficient, optimal algorithm to solve the orienteering problem on graphs when one restricts the solution to a specific class of paths called *full rows*, that were heuristically explored in [1]. Then, we introduce a new class of paths called *single column* that is related to one of the heuristics given in [1], and for this case we also provide an efficient, optimal solution. Finally, building upon the two optimal solutions found, we introduce a new heuristic that performs well in practical different scenarios. To evaluate the strength of our findings, we test our algorithms on synthetic instances based on various Zipf distributions [3], and on the same benchmarks proposed in [1] that are generated from a real world robotic precision irrigation application with graphs featuring more than 50,000 vertices.

(a) Vineyard. (b) Warehouse.

Fig. 1: Real examples of aisle-structures.

A. Related Work

The *orienteering* problem was first formalized and studied in [4], where its NP-hardness was also proven. Later, Blum et al. showed that the problem is APX-hard [5]. Due to its inherent computational complexity, numerous heuristic approaches are found in literature, and due to space constraints, the reader is referred to [6] for a comprehensive overview of heuristics and problem variants. A distinct line of research aims instead at solving the orienteering using exact

Francesco Betti Sorbelli, Alfredo Navarra, and Cristina M. Pinotti are with the University of Perugia, Italy. Stefano Carpin is with the University of California, Merced, CA, USA. Federico Corò is with the Gran Sasso Science Institute (GSSI), L'Aquila, Italy.

formulations [7]. These solutions, however, do not scale to instances as large as those we consider in this work.

Besides its theoretical interest, the orienteering problem finds applications also in robotics and automation [8]–[11]. In [1], it was shown that robotic routing problem described above in aisle-graphs can be formulated in terms of orienteering, and the most recent greedy solutions ad-hoc for aisle-graphs were proposed. In [2], the aisle-graphs are used to model a warehouse where an automated picking system is implemented. The warehouse has cabinets that form long lanes separated by aisles. In the automated picking system, the robot has to collect all the items requested by a customer order minimizing the distance (i.e., budget) traversed and changing lane only at the lane extreme, not in the middle. The picking problem has been solved by applying the well-known Christofides' algorithm for Traveling Salesman Problem [12].

Organization: The remainder of this paper is organized as follows. The formal problem definition is provided in Sec. II. Our novel algorithms are presented in Sec. III, together with their complexity analysis. In Sec. IV we evaluate the effectiveness of our approach on large scale instances, and in Sec. V we draw conclusions and sketch avenues for future work. Detailed proofs for the theorems discussed in this paper can be found in an extended version available on the Internet [13].

II. Problem Definition

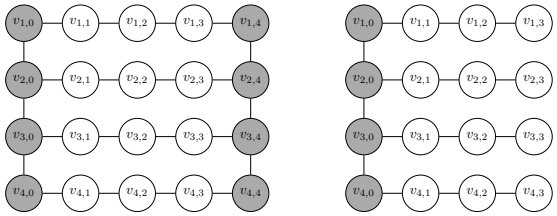
Let us consider an undirected *Aisle Graph* $A(m,n) = (V,E)$, where m and n denote the number of rows and columns, respectively. We define the set of vertices $V = \{v_{i,j} | 1 \leq i \leq m, 1 \leq j \leq n\} \cup v_{i,0}, v_{i,n+1} \forall i \in 1,\ldots,n$ and the set of edges E is built as follows:

- Each vertex $v_{i,j}$ with $1 \leq i \leq m$ and $1 \leq j \leq n$ has two edges, one toward $v_{i,j-1}$ and the other toward $v_{i,j+1}$;
- each vertex $v_{i,0}$ with $1 < i < m$ has three edges: one toward $v_{i-1,0}$, one toward $v_{i+1,0}$, and one toward $v_{i,1}$; symmetrically, each vertex of type $v_{i,n+1}$ with $1 < i < m$ has three edges: one toward $v_{i-1,n+1}$, one toward $v_{i+1,n+1}$, and one toward $v_{i,n}$. Accordingly, edges connected to the corner vertices $v_{1,0}$, $v_{m,0}$, $v_{1,n+1}$ and $v_{m,n+1}$ are then well defined.

The graph $A(m,n)$ has therefore m rows and $n+2$ columns. Note that, the vertices in the first and last columns (e.g., Fig. 2(a) with indices $j = 0$ and $j = n+1$, respectively) do not provide any reward, and their purpose is just to connect the m rows.

Problem 1 (Constant-cost Orienteering Problem (COP))**.** Let $A(m,n)$ be an aisle-graph, $v_s, v_d \in V$ be two of its vertices, $r : V \to \mathbb{R}_{\geq 0}$ be a reward function where $r(v_{i,j}) = 0$ if $j \in \{0, n+1\}$, and $c : E \to \mathbb{R}_{\geq 0}$ be a constant cost function on A, i.e., $c(e) = \alpha$ for each $e \in E$. The COP asks, for a given constant B, to find a path of maximum reward starting at v_s and ending at v_d of cost no greater than B.

W.l.o.g., it can be considered the case $c(e) = 1$ for each $e \in E$, whereas for our purposes we restrict to the case $v_s = v_d = v_{1,0}$. 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 $C = \{c_0,\ldots,c_{n+1}\}$ the set of rows and columns of A, respectively.



(a) $A(4,3)$. (b) $A_C(4,3)$.

Fig. 2: Examples of graphs: the interconnecting vertices are dark.

Moreover, we define an undirected *single column* graph $A_C(m,n) = (V,E)$ as an Aisle Graph which interconnects the m rows just using c_0. The set of vertices V and the set of edges E are equal to those of $A(m,n)$ with the only exception of column c_{n+1} which is missing in A_C and thus the vertices of type $v_{i,n}$, with $1 \leq i \leq m$, admit just one edge toward $v_{i,n-1}$. An instance of A_C is shown in Fig. 2(b).

III. Proposed Algorithms

In this section, we first optimally solve two special cases of COP: COP-FR (full row) which forces the robot to entirely traverse rows in A, and COP-SC (single column) which optimally solves COP on A_C.

COP-FR is a direct abstraction of real-world scenarios in which the robot can neither turn around nor reverse its movement. For example, if we are representing a warehouse with very narrow aisles or environments with multiple robots working simultaneously, in that case reversing the direction may be prohibited to avoid crashes, or impossible due to maneuvering limits. On the other hand, COP-SC may model instead robot motions in combed-shaped or star-shaped structures. Moreover, COP-SC can be easily modified to be used to solve COP with two robots working in parallel: one on the left side and one on the right side of an aisle-graph.

Lastly, we provide the HGC heuristic for solving COP on an aisle-graph by suitably combining the aforesaid optimum strategies.

A. Optimal Solutions for COP-FR

In this section we propose two polynomial algorithms, called OFR and OFR-I, to solve COP-FR. The latter is an efficient implementation of OFR. Both algorithms return a subset S of rows as a solution, i.e., $S \subseteq R$, such that the path starting and ending at $v_{1,0}$ has cost no greater than B and collects maximum reward.

The following properties hold for COP-FR:

- *P1:* A reasonable budget $B \geq 2(n+1)$ must hold since a smaller value makes impossible to traverse any row of the graph and coming back to $v_{1,0}$. In contrast if $B \geq (n+1)m + 2(m-1)$ then the problem becomes trivial since the robot can easily perform a complete visit of the graph.
- *P2:* In any optimal solution S, the robot performs a cycle traversing the rows in S in an increasing (or decreasing)

order with respect to their indices. Any other cycle would require a larger budget.

- *P3:* Any optimal solution S that starts and ends at $v_{1,0}$ must fully traverse an even number of rows to prevent the robot to get stuck in c_{n+1}.

1) The OFR *Algorithm:* We now introduce the OPT FULL-ROW (OFR) algorithm that optimally solves COP-FR. The pseudo-code of OFR is provided in Algorithm 1.

Algorithm 1: OFR(Graph A, Budget B)

```
1  V_cycle ← ∅, opt ← −∞
2  R_i ← Σ_{j=1}^{n} r(v_{i,j}) ∀i ∈ 1,...,m , R̃ ← SORT(R_i)
3  for m' ← 1, m do
4      V = ∅, val ← 0
5      B' ← B − 2(m' − 1)
6      if B' > 0 then // Build current solution
7          k(m') = 2 ⌊ B' / (2(n+1)) ⌋
8          if k(m') ≥ m' then
9              val ← Σ_{j=1}^{m'} R_j, UPDATETOUR(V)
10         else
11             j ← 1, s ← 1
12             val ← R_{m'}
13             while s ≤ k(m') − 1 do
14                 if i_j < m' then
15                     val ← val + R̃_{i_j}, UPDATETOUR(V)
16                     s ← s + 1
17                 j ← s + 1
18         if val > opt then
19             V_cycle ← V, opt ← val
```

OFR consists of two steps. In the first pre-processing step, for each row i a cumulative reward R_i, i.e., the reward collected by completely traversing the i-th row, is computed (Line 2), as $R_i = \sum_{j=1}^{n} r(v_{i,j})$. Then (Line 2), we sort the R_i values in decreasing order obtaining the row permutation $\tilde{R} = [i_1, \ldots, i_m]$ such that $\tilde{R}_{i_1} \geq \ldots \geq \tilde{R}_{i_m}$. This phase has cost $\mathcal{O}(mn + m \log m)$.

In the second step, exploiting the three properties above, the optimal solution with budget B can be easily determined if we fix the furthest row traversed by the algorithm.

Letting m' be the furthest traversed row (Line 3), the solution moving vertically spends $(m' - 1)$ units of budget to reach row m' from $v_{1,0}$, and additional $(m' - 1)$ units to go back to $v_{1,0}$. Any optimal solution can either first visit the selected rows up to row m' and then come back, or first reach the furthest row m' and then come back visiting the selected rows. The two options have the same vertical cost of $2(m' - 1)$: the vertical movements of the robot from row 1 to row m' alternate on c_0 and c_{n+1}. In the former case, the vertical movement from row 1 to the first row selected in the solution are performed on c_0. Once the first row is reached, the robot traverses it left to right, ends up in c_{n+1}, and moves vertically from the first to the second selected row on c_{n+1}. Since the second row is traversed right to left, the robot ends up in c_0, and starts again going down to reach the third selected row. After traversing the furthest row m', if m' is even, the robot goes back to $v_{1,0}$ traversing c_0. Otherwise, if m' is odd, the robot traverses row m' again from right to left and reaches $v_{1,0}$ moving on c_0.

Given that $2(m' - 1)$ is the budget required for the vertical movements, the residual budget $B' = B - 2(m' - 1)$ (Line 5) is spent to traverse as many even rows as possible (Line 7) that is $k(m') = 2 \left\lfloor \frac{B'}{2(n+1)} \right\rfloor$. The optimal solution than includes row m' and the $k(m') - 1$ rows with the largest reward among the first m' rows (Line 13) if $k(m') \leq m'$. If $k(m') > m'$ (Line 8), then all the m' rows are selected. If m' is odd, to make an even number of rows, a row, say m' for the sake of simplicity, is traversed twice. Note that, if $B < 2(m' - 1)$, the m'-th row cannot be reached and there is no feasible solution that includes row m'.

To find the optimal solution constrained to budget B, we compute the most profitable solution for any possible value of m' (Line 3) and we retain the solution with the largest overall reward (Line 18).

Finally, the procedure UPDATETOUR(V) constructs the path accordingly to the rows that the robot has to take, and adding the suitable vertical movements as explained above.

The total computational cost of the algorithm is $\mathcal{O}(m \cdot \max\{n, m\})$ since it spends $\mathcal{O}(mn + m \log m)$ for the pre-processing phase and $\mathcal{O}(m^2)$ for the second phase.

By the above discussion the next theorem can be stated.

Theorem 1. *Algorithm* OFR *optimally solves COP-FR in time* $\mathcal{O}(m \cdot \max\{n, m\})$.

2) The OFR-I *Algorithm:* We next present a faster implementation for OFR, called OFR-I. The idea is to reduce the cost of selecting the rows of the optimal solution from $\mathcal{O}(m^2)$ time for OFR to $\mathcal{O}(m)$ time for OFR-I. The pseudo-code of OFR-I is provided in Algorithm 2.

Initially, OFR-I performs the same pre-processing phase as the one of OFR. Then, in the second step, the optimal solution is calculated in the opposite way with respect to OFR. In fact, we first set m' as the farthest possible row, i.e., $m' = m$ (Line 4). Then, the main loop proceeds decreasing m' and, at each step, according to the value of m', the residual budget and the number of permitted rows to be considered are computed (Line 7). During the first iteration of the main loop, i.e., $m' = m$ (Line 8), the algorithm builds the current solution S by picking the first $k(m')$ rows in $\tilde{R}$ (Line 9). The indices of the top $k(m)$ rows with the largest rewards (Line 9) are inserted in the Boolean vector S (Line 10). Precisely, $S[i_j] = 1$ if i_j is one of the first $k(m)$ entries in $\tilde{R}$, and $S[i_j] = 0$ otherwise, with $1 \leq i_j \leq m$. Clearly, the insertion procedure can be done in constant time for each row. Note that using this approach we are not forcing row m' to belong to the current solution. It is worthy to note that when $m' = m$ the total reward obtained by OFR-I is larger or equal to that of OFR constrained to use row m because OFR-I takes the best possible top $k(m)$ rows and OFR only the top $k(m) - 1$ best possible rows plus the m-th. When $m' < m$, OFR-I evaluates the new residual budget.

Algorithm 2: OFR-I(Graph A, Budget B)

```
1  V_cycle ← ∅, opt ← −∞, old ← 0
2  S ← [1 ... m], j ← 1   // Current index
3  R_i ← Σ_{j=1}^{n} r(v_{i,j})  ∀i ∈ 1, ..., m, R̃ ← SORT(R)
4  for m' ← m, 1 do // Decreasing cycle
5      V = ∅, val ← 0
6      B' ← B − 2(m' − 1)
7      k(m') = 2 ⌊ B' / (2(n+1)) ⌋
8      if m' = m then // 1-st iteration
9          while j ≤ k(m') do
10             S[i_j] ← 1
11             val ← val + R̃_{i_j}, UPDATETOUR(V)
12             j ← j + 1      // j scans R̃
13         old ← k(m')
14     else if m' < k(m') then
15         for i ← 1, m' do
16             S[i] ← 1
17             val ← val + R_i, UPDATETOUR(V)
18         m' ← 1
19     else
20         c ← 0
21         if k(m') ≠ old then
22             c ← 2
23         if S[m' + 1] = 1 then
24             c ← c + 1, S[m' + 1] ← 0
25             val ← val − R_{m'+1}, UPDATETOUR(V)
26         t ← 1
27         while t ≤ c do // Finds c feasible rows
28             if i_j ≤ m' then
29                 S[i_j] ← 1
30                 val ← val + R̃_{i_j}, UPDATETOUR(V)
31                 t ← t + 1
32             j ← j + 1
33         old ← k(m')
34     if val > opt then
35         V_cycle ← V, opt ← val
```

As in OFR, the best solution using the rows from 1 to m' is searched. It may happen that 1) the number of permitted rows $k(m')$ increases with respect to $k(m'+1)$, and 2) the row $m'+1$ has to be removed from the current solution to be substituted by a row with index no larger than m'. In the first case, the number of permitted rows may increase since the length of the vertical movements decreases by 2 and thus, the residual budget increases by 2. According to Line 7, it must hold $c = k(m'+1) - k(m') = 2$. In the second case the algorithm is able to remove the $(m'+1)$-th row in constant time just by setting $S[m'+1] = 0$ and increasing by 1 the number c of rows to be added to the current solution (Line 23).

For every value of m', the algorithm adds c rows scanning $\tilde{R}$ starting from the position j of the last row inserted and ignoring the rows with index larger than m' (Line 27). If $c = 0$, the solution S is unchanged and the algorithm proceeds decreasing m'. Otherwise, $\tilde{R}$ is scanned from position j to find c new rows. Note that j never decreases because any row discarded during a previous iteration cannot be inserted in this iteration either. Namely, whenever a row i_j in $\tilde{R}$ is discarded from S it is because its row index is greater than the current value m', i.e., $i_j > m'$, and since m' decreases, i_j cannot be reinserted in any subsequent iteration. The algorithm terminates when at least one of the following conditions is satisfied:

1) $m' < k(m')$: in such a case the current solution takes just the first m' rows (Line 14);
2) $m' = 1$: which is the main condition of the algorithm.

Even if more than c positions of $\tilde{R}$ may be visited for each value of m' (Line 27), overall during the second phase (Line 4), no more than $|\tilde{R}| = m$ rows are visited. Since any row visited is handled in constant time, to update the solutions for all the values of m' costs $\mathcal{O}(m)$.

The total cost of OFR-I is then $\mathcal{O}(m \cdot \max\{n, \log m\})$ since it spends $\mathcal{O}(mn + m \log m)$ for the pre-processing phase and $\mathcal{O}(m)$ for the second phase. Thus, the overall time complexity of OFR-I is never worse than that of OFR, but ignoring the first pre-processing step whose results could be passed in input to the algorithm along with the reward graph, OFR-I is much faster than OFR. Indeed, OFR-I costs $\mathcal{O}(m)$ time, while OFR $\mathcal{O}(m^2)$ time.

By the optimality of OFR and by the above discussion the next theorem can be stated.

Theorem 2. *Algorithm* OFR-I *optimally solves COP-FR in time* $\mathcal{O}(m \cdot \max\{n, \log m\})$.

B. Optimal Solution for COP-SC

For COP-SC, since there is only one column in A_C, the robot is forced to go back-and-forth on each selected row. We devise a dynamic programming algorithm, called OPT SINGLE-COLUMN (OSC), that optimally solves COP-SC in polynomial time. Clearly, OSC sub-optimally solves COP.

During the initialization, we create two tables T and R of size $m \times (n+1)$ and $m \times (\frac{B}{2}+1)$, respectively. Table R has columns $j = 0, 1, \ldots, \frac{B}{2}$. We initialize table T as follows: $T[i,j] = \sum_{k=0}^{j} r(v_{i,k}) = \sum_{k=1}^{j} r(v_{i,k})$ 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 $0 \leq j \leq n$. This phase has cost $\mathcal{O}(mn)$. In table R, let $R[i,b]$ be the largest reward that can be attained with budget $2b$ visiting the first i vertices of row 0. Let $S[i,b]$ be the last column visited in row i to obtain the reward $R[i,b]$.

As for COP-FR, COP-SC there are some general properties that will be exploited by our dynamic programming solution:

- *PA:* If $B \geq 2nm + 2(m-1)$ then the robot can visit the whole graph.
- *PB:* An optimal solution must traverse the selected rows in increasing (or decreasing) order w.r.t. their indices.
- *PC:* An optimal solution visits a row at most once. Two (or more) visit of the same row can be in fact replaced by the only visit that reaches the furthest vertex.
- *PD:* For each row the algorithm has to select the last vertex to visit. If no vertex is visited, still $v_{i,0}$ has to be

traversed to reach the subsequent row $i+1$.

The optimal solution that visits the first j vertices in row i, with $j \geq 0$, gains $T[i,j]$ reward from row i, and spends budget $2b-2j-2$ to reach the previous row $i-1$. Namely, we reserve 2 units of budget to change row and $2j$ units to traverse row i. We thus have the next recurrence. The first row of table R is defined as follows:

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

Then, for each subsequent row $R[i,b]$, $i \geq 2$:

$$R[i,b] = \begin{cases} -\infty & b < i-1 \\ 0 & b = i-1 \\ \max\limits_{0 \leq j \leq \min\{b-i,n\}} \{R[i-1,b-j-1]+T[i,j]\} & b > i-1 \end{cases}$$

Note that $j \leq \min\{b-i,n\}$ is obtained by observing that $j \leq n$, $b-j-1 \geq 0$ and more strictly $b-j-1 \geq i-1$ to avoid $-\infty$ reward. If $b = i-1$ and $i \geq 2$, $R[i,b]$ is just 0 because there is only enough budget to reach row i traversing c_0. Precisely, $R[i,b] = \max_{j=0}\{R[i-1,b-j-1]+T[i,j]\} = R[i-1,b-1]$ and recursively back up to $R[1,0] = 0$.

Table S is then filled recalling for each position the column that has given the maximum reward, i.e., $S[i,b] \leftarrow j = \arg\max R[i,b]$. 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 we do not know in advance the furthest row reached by the optimal solution. The solution is computed in $O(m)$ tracing back the choices using the table S. Note that with a single execution of OSC, we compute the optimal reward for any value b, with $0 \leq b \leq \frac{B}{2}$. 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 $\mathcal{O}(m\frac{B}{2})$ to retrieve the solution, and takes $\mathcal{O}(mn + m\frac{B}{2})$ space. Since by property PA budget B is upper bounded by $\mathcal{O}(mn)$, then algorithm OSC is strictly polynomial in the size of the input.

By the above discussion and sub-optimality arguments, the next theorem can be stated.

Theorem 3. *Algorithm* OSC *optimally solves COP-SC in time* $\mathcal{O}(mn\frac{B}{2})$.

C. Heuristic solution for COP

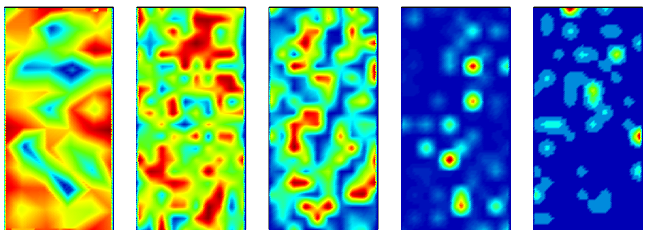
We propose an heuristic, called HEURISTIC GENERAL-CASE (HGC), that combines the sub-optimal solutions of COP returned by OFR-I and OSC algorithms in three strategies: $H1$, $H2$, and $H3$. $H1$ prefers full rows, while $H2$ and $H3$ select partial rows on both the left and right side plus some full rows to complete the tour: $H3$ traverses only two full rows (i.e., the minimum number of rows to reach the right side and be back at $v_{1,0}$), while $H2$ replaces all sufficiently long partial rows with full rows. Precisely:

- *H1:* First OFR-I is applied. If some budget p there remains, then clearly $p < 2(n+1)$. OSC is applied to select partial rows on both the sides of the aisle-structure A, constrained to p and to the requirement to return in $v_{1,0}$. In particular, the partial rows can be selected among all the rows of the left side not yet selected, and among the rows of the right side that the robot passes in front traversing the tour created for OFR-I.
  It is worthy to note that the reward of $H1$ always dominates that of OFR-I.
- *H2:* Here first OSC is applied with the full budget B, obtaining a temporary solution S_L then again OSC is applied with the full budget B but starting and returning in $v_{1,n}$, hence optimizing on the right side and obtaining another temporary solution S_R. Successively, S_L and S_R are analyzed so as to select an even number of full rows. These will be the rows where S_L and S_R combined together select at least $\frac{n}{2}$ elements. If none of such rows exist, then bounds $\frac{n}{3}$, $\frac{n}{4}$, ..., will be considered just to select two rows. The final solution will then include the selected full rows. According to the residual budget, the solution will be completed with all possible partial rows similarly to $H1$;
- *H3:* Here firstly two rows are selected: the first one and the furthest row reachable with the budget B. Successively, OSC is applied with the residual budget selecting the partial rows as in $H1$.

Heuristic HGC then returns the best solution among $H1$, $H2$, $H3$, OFR-I and OSC.

IV. SIMULATIONS

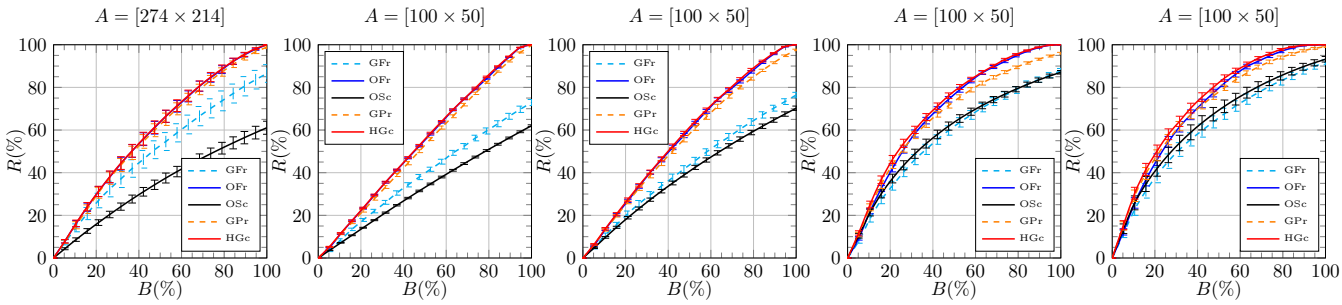
We test our algorithms on the same benchmarks proposed in [1] that are generated from a real world robotic precision irrigation application. The reward map of one of them is illustrated in Fig. 3(a), where the cold and hot colors represent low and high rewards, respectively. We also test our algorithms on synthetic instances whose rewards are based on various Zipf distributions [3]. The synthetic instances consist of graphs $A(100,50)$ and $A(50,100)$. The rewards are random integers in the interval $[0,100)$ that follow a Zipf distribution of parameter $\theta = \{0, 0.9, 1.8, 2.7\}$. To avoid a punctiform reward in A and make it more continuous as it is in the real world, we generate $\frac{m \cdot n}{5 \cdot 5} = \frac{50 \cdot 100}{5 \cdot 5}$ random numbers and we assign each of them to a 5×5 sub-graph. When $\theta = 0$, the rewards are uniformly distributed in $[0,100)$ (see Fig. 3(b)), while when θ increases the lowest rewards become more and more frequent (see Figs. 3(c)-3(e)).



(a) Real (b) $\theta = 0$ (c) $\theta = 0.8$ (d) $\theta = 1.9$ (e) $\theta = 2.7$

Fig. 3: Reward maps for real and synthetic graph instances.

For each A and θ, we run our new algorithms OFR, OSC, HGC and contrast them with the two greedy heuristics GREEDY FULL-ROW (GFR) and GREEDY PARTIAL-ROW (GPR) presented in [1]. Both OFR and OFR-I optimally solve COP-FR, so we do not need to test both.



(a) Real (b) $\theta = 0$ (c) $\theta = 0.8$ (d) $\theta = 1.9$ (e) $\theta = 2.7$

Fig. 4: The performances for the real benchmark and synthetic data with different values of θ.

The GFR and GPR heuristics choose a subset of full and partial rows to be traversed, respectively. At each selection, the robot computes the budget required to collect rewards from its current position, and prefers row/vertices with maximum reward per unit of budget. The time complexities of GFR and GPR are $\mathcal{O}(m^2)$ and $\mathcal{O}(m^2 n)$, respectively. Fixed m and n (i.e., the size of A) and θ, we populate 30 random graphs. For each graph, we test each algorithm with budget B increasing in 20 steps from the minimum to the maximum value (see *P1*, Sec. III-A). Since the overall reward differs for each graph (as rewards are randomly generated), we return the reward gained as a percentage of the overall reward. Finally, we plot the average of the results on the 30 instances, along with their 95% confidence interval. Fig. 4 compares the reward over the different algorithms on the real and synthetic data and different θ. When $\theta = 0$, the performance of OFR increases linearly with the budget and it is absolutely comparable with that of the general heuristics GPR and HGC even if it is computationally lighter. GFR instead performs poorly with respect to OFR. OSC is the worse in the uniform case and not surprisingly it gains about half of the reward of OFR. In fact, OSC requires the double of the budget than OFR to explore a single row. When θ increases, and rewards become scarcer, the percentage of gained reward increases faster than the budget: at $\theta = 1.9$ with 50% of the budget, OFR gains already more than 70% of the reward. The performances of OFR is coincident with that of HGC up to $\theta = 1.9$, and about 1–2% less when $\theta = 2.7$. For large values of θ, the performance of GPR slightly downgrades especially for budget between 50% and 80%, whereas that of OSC upgrades although OSC explores only the left side of A. In fact, when the reward is unbalanced, for OSC it is easier to decide which vertices to take, while GPR may waste budget on useless vertical movements because the rewards are widely scattered. OSC reaches the same performance of HGC and GPR when $\theta \geq 1.9$ and $B \leq 20\%$.

Fig. 4(a) compares the algorithms behavior on a set of 20 real world robotic precision irrigation graphs, collected in the open fields by the authors of [9]. HGC and OFR achieve the best results, and generally improves by less than 2% the performance of GPR (with a peak of 5% when $B < 20\%$). In general, the GFR heuristic performs better on the real data than on the synthetic ones. Vice-versa, OSC performs better on the synthetic data than on the real ones. According to our experiments, the performance on real graphs are comparable with that of synthetic graphs of moderate skewness.

Observations: Although all experiments show excellent performance for GPR and even better for OFR, we are aware that one can hand-pick graphs where such algorithms behaves rather poorly compared the optimal solution. For example, consider a graph $A = A(m, n)$ built as follows: for each vertex $v_{i,2}$ with $i \in 1, \ldots, m-1$ we set $r(v_{i,2}) = 2i - \epsilon$; we set reward $r(v_{m,m-2}) = m - 1 + 2(m-2) = 2m - 3$ for the vertex $r(v_{m,m-2})$ of the last row; and finally we set reward equal to zero for all the other vertices. Let $B = 2(m-2) + 2(m-1)$ the given budget. The OFR will select the furthest row m' it can reach constrained on B and the previous row $m' - 1$ for a total reward that cannot be larger than $2m - 5 + m - \epsilon < 3m - \epsilon$. The GPR will choose in the first iteration the vertex $v_{m,m-2}$ consuming the whole budget for a total reward of $r(v_{m,m-2}) = m - 1 + 2(m-2) = 2m - 5$. The OSC algorithm will compute a solution composed by the first $m - 1$ rows with a total reward of $\sum_{i=2}^{m-2}(i - \epsilon) = \frac{1}{2}(m-3)(m-2\epsilon)$. Since the optimal solution will gain at least the same reward as OSC, both OFR and GPR are away from the optimum by an arbitrarily large factor m.

Similar examples can be found where OSC looses, while GPR and OFR are much closer to the optimum. In conclusion, since OSC and OFR seem to show opposite behaviors in the same instance (like in the example above) and since HGC applies both, our aim for HGC is to limit performance losses. On the one hand we could not find any dramatic instance for HGC as for the other heuristics. On the other hand we are looking forward for formal arguments to guarantee some limited approximation ratio.

V. Conclusion

We have shown how to optimally solve in polynomial time the two special cases COP-FR and COP-SC of COP. We have also simulated the new optimal proposed algorithms OFR and OSC as well as the new heuristic HGC in the special vineyard context where COP was originally defined. Although HGC gains in percentage few units over the previous best algorithm, it can be still valuable in high constrained scenario. As future work, we will undertake further investigations for providing formal guarantees on the quality of the solutions in the general case.

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] 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.

[3] 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.

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

[5] 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 Computing*, vol. 37, no. 2, pp. 653–670, 2007.

[6] A. Gunavan, H. C. Lin, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications," *European Journal of Operational Research*, vol. 255, no. 2, pp. 315–332, 2016.

[7] M. Fischetti, 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.

[8] S. Jorgensen, R. Chen, M. 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.

[9] 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.

[10] 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.

[11] R. Pěnička, J. Faigl, and M. Saska, "Physical orienteering problem for unmanned aerial vehicle data collection planning in environments with obstacles," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 3005–3012, 2019.

[12] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.

[13] F. Betti Sorbelli, S. Carpin, F. Corò, A. Navarra, and C. M. Pinotti, "Optimal routing schedules for robots operating in aisle-structures," *CoRR*, vol. abs/12345678, 2019. [Online]. Available: http://arxiv.org/abs/12345678