Computers and Operations Research 114 (2020) 104826

Contents lists available at ScienceDirect

Computers and Operations Research

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

# An integer linear programming model for efficient scheduling of UGV tasks in precision agriculture under human supervision

Lohic Fotio Tiotsop*, Antonio Servetti, Enrico Masala

*Control and Computer Engineering Department Politecnico di Torino Corso Duca degli Abruzzi 24, Torino 10129, Italy*

ARTICLE INFO

*Article history:*
Received 25 January 2019
Revised 11 October 2019
Accepted 14 October 2019
Available online 14 October 2019

*Keywords:*
Data acquisition
Farming activities
UGV Navigation
Integer linear programming
Service robotics

ABSTRACT

In precision agriculture more and more robots are being used to perform tasks that may include some farming activities, such as pruning, inspection or spraying, assigned to the robot as a result of a previous analysis activity or autonomously identified by the machine itself. In this sensitive scenario, reporting difficult situations to a decision maker, e.g., a human operator or some sophisticated software tools that cannot be integrated with the robot, could be useful to perform the correct action that the machine has to execute. Unfortunately, this key aspect is still neglected in current literature that focuses, instead, on fully automated operations by robots. Moreover, it is necessary to consider that in rural areas it often happens that successful data communication can only be achieved in certain locations in the field. In this context, we aim to address all the previous shortcomings by formulating a more comprehensive optimization problem, which also models the necessity to report to a central location and get instructions on the task to be done before proceeding to perform each action. After presenting two alternative analytical formulations of the problem, i.e. an integer linear programming model (ILP) and a mixed integer linear programming model, we propose a branch and bound algorithm that is guaranteed to find the global minimum cost solution in terms of navigation time. Simulation results show that our proposed algorithm performs about 20 to 30 times faster with respect to commercial linear programming solvers using any of the two analytical models proposed. Moreover, we also propose further improvements to reduce computational time while maintaining solution optimality. Finally, some insight into the development of future heuristics is given by analyzing the speed of convergence towards the optimal solution.



## 1. Introduction

The amount of robots employed in agriculture applications is constantly increasing. Two types of robots are typically used: unmanned ground vehicles (UGV) and unmanned aerial vehicles (UAV) (Zecha et al., 2013). Examples are rovers that navigate the cultivation to perform specific agriculture tasks, e.g., inspection or physical operations such as collecting samples, pruning, spraying (Bonadies et al., 2016). UAVs, instead, are mainly used for imagery tasks, i.e., to take pictures for later processing in order to understand the status and requirements of different parts of the cultivation (Candiago et al., 2015; Lukas et al., 2016). Collaboration between UAVs and UGVs is also being actively explored (Vasudevan et al., 2016; Vu et al., 2018).

Despite some operations can be done autonomously by the robots, sometimes communication with a central location is needed, so that a human operator or an algorithm, more sophisticated compared to the one that can run on the robot, can be used in the loop to take decisions and act accordingly in a specific terrain position. In fact, a recent survey about the usage of agricultural robots (Bechar and Vigneault, 2016) pointed out the lack of models considering such aspect in the current literature, while at the same time highlighting its necessity to correctly model actual problems.

Note that this work relies on an agricultural scenario to present the models just because it seems to be one of the most challenging real applications, but it can also cover other real situations, such as the one described in the following. In case of industrial operations being performed by robots with human supervision, there could be $A$ ($A$ typically large) jobs located in different places to be performed by a robot and a supervisor operator may just have a rough knowledge of what is the exact job to be done at each location. There are also $B$ locations where the robots can collect what is needed to perform a job. The proposed models can be used to schedule the $A$ jobs minimizing the time required to complete them. At any job location the robot first performs an inspection,

* Corresponding author.
*E-mail addresses:* lohic.fotiotiotsop@polito.it (L. Fotio Tiotsop), antonio.servetti@polito.it (A. Servetti), enrico.masala@polito.it (E. Masala).

https://doi.org/10.1016/j.cor.2019.104826
0305-0548/

then moves to a point of type *B*, if necessary, by sending in the meanwhile a report to the supervisor operator that decides what to do and what is needed to perform the job and informs the robots that then act accordingly. Another example is repair companies: they may rely on the proposed models to schedule their logistic operations. The customer locations would represent points of interests and the company warehouses are the points to be visited to perform repairs at the point of interests. More precisely, when an operator leaves the main warehouse in the morning, at each customer location he/she either fixes the problem (if the description was exhaustive) or collects information about the problem and what is needed, then visits a company warehouse in order to pick up all he needs and go back later to the customer location to fix the problem. More in general, our models can be used for efficiently planning operations requiring data acquisition, data elaboration or transmission, and finally an action.

In this work we focus on the specific agriculture scenario considering and modelling the constraint imposed by communicating with a decision maker, e.g., a human operator, before performing physical actions. Considering the optimization problem, we assume to have a rough preliminary knowledge about the terrain and how a UGV can move in the area. In other words, obstacle positions and possible navigation paths are known. This can be obtained by previous knowledge or by means of a survey, e.g., made using a UAV or other similar technologies. During the survey, points of interests (POIs) are also identified, i.e., areas to be visited by the UGV in order to gather information, decide what to do and act accordingly. Since the decision of the action to perform in each POI might be difficult to take only relying on the on-board processing capabilities or simply might require human judgment not available through algorithms, we assume that after the UGV has gathered information by moving to the POI and taking, e.g., a picture, it must communicate with a central location to get an answer about the farming action to perform. The UGV must then perform the action returning to the POI, if necessary.

However, it is well know that communications, including wireless, in rural areas might be difficult (Nandi et al., 2016), mainly due to two reasons. The first reason is, sometimes, the presence of massive physical constraints, such as hills, rocks, subsidence (MacCartney et al., 2016). The second reason is the typically scarce coverage of 3G/4G cellular technology, since there is small incentive for network operators to provide good quality coverage of such areas due to, e.g., the low people density. As a consequence, depending on the physical position, either no communication is possible or only wireless communications are available but with locally-deployed infrastructure such as Wi-Fi access points or similar systems, potentially tuned to communicate over longer distance compared to the usual home devices. In our scenario, we assume that we roughly know that in certain places on the terrain communication is possible, whereas in others this is not possible at all. Such information could come as a result of a previous survey, or considering the relative position of the devices, the terrain topography and similar features.

Our goal is to provide an efficient algorithm to solve the navigation problem of the UGV on the terrain while minimizing the time needed to perform all the tasks in the previously determined POIs, considering that before performing a task in a POI it has to i) visit the POI and collect some data about its state, ii) visit a location where those data can be transmitted and instructions on the task to be done are received, iii) go to the POI and perform the action. Obviously the UGV does not have to perform the three steps sequentially for each POI, but it can visit multiple POIs before moving to a place where it can communicate, and then it can go back to each one of them, in any order, to execute the received instructions. Minimizing the time taken by the UGV is economically advantageous for many reasons. For instance, if the UGV is rented, it can operate in more places therefore serving more than one customer, thus the company renting it can maximize its earnings. Conversely, if the UGV needs to be bought, a lower number of them is needed to perform all the activities in the timeframe required by the specific agriculture scenario so that the UGV could potentially be employed for other activities.

This paper significantly extends our preliminary work (Fotio Tiotsop et al., 2019, to be published). In particular, we present an improved version of our proposed branch and bound algorithm which relies on the computation of a lower bound to reduce the execution time. Moreover, an additional mixed ILP model employing dummy vertices is presented for comparison purposes. Finally, this paper includes an extensive set of performance results on several different graph instances, i.e. fifty instances for each level of complexity, both using the two ILP models and the proposed algorithm.

The paper is organized as follows. Section 2 reviews related work in the field, whereas Section 3 explains how to represent the graph which is the basic input data for our algorithm. Then, the optimization problem is analytically formulated in Section 4 as two ILP problems, using two different approaches, to study its characteristics. In Section 5 a specific algorithm for the problem is designed and implemented in the form of a branch and bound algorithm, and some efficient pruning and lower bounding strategies are proposed. Section 6 shows some practical experimental results on realistic graphs, followed by conclusions in Section 7.

## 2. Related work

Sensing technologies are more and more used in agricultural applications to perform crop monitoring (Castellini et al., 2017; Dong et al., 2017). For instance, imaging sensors in the visible spectrum (Chang et al., 2017) or at other wavelengths (Khanal et al., 2017) are often used. Image data can then be elaborated to detect cultivation type, e.g., vineyard Comba et al. (2015), or to detect single plants and potential anomalies (Primicerio et al., 2017a). A more comprehensive overview of such technologies can be found in Matese and Di Gennaro (2015).

Potential issues can be identified and transformed into a set of POIs so that, on the basis of the cultivation map, a UGV can move in such positions in order to perform a more detailed analysis, e.g., taking closer pictures or perform local measures with sensors, to later perform some specific physical action (Das et al., 2015; Pei et al., 2014). For the purpose of solving the UGV movement problem, the cultivation field can be abstracted in the form of a graph with nodes (i.e., the POIs) connected by edges whose weight represent the cost, in terms of time, needed to move from one point to the other.

Graph visiting problems have been well investigated from the theoretical point of view. The travelling salesman problem (TSP) or more in general the vehicle routing problem (VRP) and some variants have been intensively studied in literature Eksioglu et al. (2009); Fadda et al. (2018); Huang et al. (2018); Laporte (1992); Marinakis et al. (2007); Salavati-Khoshghalb et al. (2017). In short, they consider a set of vehicles that can transport goods and a graph whose nodes represent a set of customers, each one characterized by a given demand. Each edge represents the cost that should be sustained to move from a customer to another. The VRP aims at determining which customers should be served by any vehicle and the schedule of the operations of any vehicle in order to minimize the total cost satisfying the capacity constraint of each vehicle and the demand of each customer. In case of a single vehicle, the problem is reduced to the TSP.

Reducing field work time in agriculture by optimally scheduling agricultural tasks is often assimilated in the literature to graph

visiting problem. In Bochtis and Sørensen (2010); Seyyedhasani and Dvorak (2017, 2018) the authors explain how the VRP can be used to gain efficiency in field logistics. In particular in Seyyedhasani and Dvorak (2017) the numerical experiments show that it is possible to save up to 32% of the time if the operations are guided by the solution of the proposed VRP instead of an intuitive approach. The TSP is used in Zhou et al. (2014) to schedule the farming activities in fields characterized by the presence of a large number of obstacles.

While the goal of these works is to minimize the time needed to complete the assigned farming tasks, similarly to our navigation problem, nevertheless they are substantially different in the fact that they do not consider the communication aspect and thus the interaction with an external decision maker, e.g., a human operator or any other external tool, whose presence coordinates the operations and addresses critical situations.

Some VRP variants are somehow more similar to our case, for instance the so called vehicle routing problem with intermediate stops (VRP-IS) Schiffer et al. (2019). An intermediate stop (IS) is a stop that occurs at a node whose visit is necessary to keep the vehicle operational or to perform any other action necessary to pursuit the main task assigned to the vehicle. ISs have been considered for replenishment (Crevier et al., 2007), unloading of waste (Kim et al., 2006), refueling (Bousonville et al., 2011; Schiffer and Walther, 2017), rest (Vansteenwegen et al., 2012) and synchronization requirements (Drexl, 2012). Similarly to the VRP-IS, our problem considers, besides the POIs where physical actions are required, the points covered by wireless network where the UGV might stop to transfer the data collected until that point and get instructions about the correct actions to perform later at the POIs. In our problem the UGV is hence constrained to visit each POIs again once the data acquired during the first visit have been transmitted. This clearly introduces further constraints compared to the cited works.

The more practical problem of computing the exact physical path corresponding to each edge in our graph is addressed in the robotics research field by works focusing on robot navigation problems. For instance, one of the most common problems is the so-called path-planning, in which a trajectory must be computed so that a robot can move to a certain target position while fulfilling certain constraints, i.e., avoiding obstacles and forbidden or dangerous conditions (Basaca-Preciado et al., 2014; Ferentinos et al., 2002; Gasparetto et al., 2015). More complex constraints can be taken into account, e.g., the slope of the terrain, as in Contente et al. (2016) that presents the case of a UGV that needs to visit all the rows in a vineyard.

Once edge weights and node positions are known, our problem resembles (but it is not the same) the Steiner TSP (STSP) Zhang et al. (2016). Given a graph, a cost for each edge of the graph and a subset of nodes that represent customers, the STSP aims at finding the minimum-cost tour that passes through each customer node. Edges may be traversed more than once, and nodes visited more than once, if necessary. Such a problem, despite being somehow similar to ours, presents an important difference. In that problem, all clients have to be visited at least once. In ours, we require to visit at least *twice* the POI nodes making sure that at least one of the visits at a given POI occurs *after* the transmission of the related information. This further increases the complexity, since time variables need to be introduced in order to manage the chronology of the tasks.

## 3. Graph representation

In our work we assume that a picture has been used to derive a graph that represents the field in which the UGV must operate. The task of creating a graph from a picture by detecting the different types of areas and how they are connected and which is the optimal movement path between different areas is a problem well addressed in literature (see Nex and Remondino (2014); Primicerio et al. (2017a); Sona et al. (2016); Torres-Sánchez et al. (2014)). Therefore, here we assume that such a graph is available. In more details, in our scenario the graph includes nodes, which correspond to physical locations in the terrain. Such nodes are connected with edges whose weight represents the cost of moving from one node to the other. Obstacles and possible movement paths are already modeled in such weights.

Concerning nodes, they can be classified into three types.

- Type A: The nodes that we want to visit, in which a physical operation has to be performed by the robot.
- Type B: The nodes in which we are sure that wireless communication can be established in order to both transmit the information collected at previous 'A' nodes and receive instructions about what to do in those nodes.
- Type AB: The nodes with the characteristics of both 'A' and 'B'. When visited, the physical operation can be immediately performed after data communication has taken place, since it is possible to immediately communicate and receive instructions.

Detecting 'A' nodes heavily depends on the specific task to do (e.g., detecting potential weeds, or places where plant pruning operations might be necessary). The same applies for 'B' nodes, in which wireless communication models, combined with the terrain geography can be used to determine the possible coverage, or maybe the information is simply available from a previous survey.

## 4. Mathematical formulations

In this section two alternative models of the problem are presented. Both models need to address the problem of modeling the movements of the UGV that needs to traverse nodes and edges an unknown number of times. The first formulation addresses the issue by means of an ordered set of displacements (i.e., a movement from a node to another) that the UGV must follow to complete the required tasks. In this formulation the UGV can pass more than one time on the same node, whereas in the second model this condition is considered by means of dummy vertices (i.e., replica nodes to consider multiple visits).

### 4.1. Displacement-based formulation (Model I)

Let us define the following elements:

- $\mathcal{A}$: the set of nodes to visit (Type 'A' or 'AB')
- $\mathcal{B}$: the set of nodes where wireless communication can be established (Type 'B' or 'AB')
- $\mathcal{V} = \mathcal{A} \cup \mathcal{B}$ set of all nodes
- $\mathcal{K}$: the set of displacements of the UGV
- $\mathcal{E}$: the set of edges connecting the different nodes
- $A$, $B$, $E$ and $K$ respectively the cardinality of $\mathcal{A}, \mathcal{B}, \mathcal{E}$ and $\mathcal{K}$
- $s$ the depot node or the central location
- $t_{ij}$: the time required to cover the edge $(i, j) \in \mathcal{E}$
- $x_{ij}^k$: a boolean variable equal to 1 if during its k-th displacement the UGV moves along the edge $(i, j) \in \mathcal{E}$
- $y_i^k$: a boolean variable equal to 1 if during the k-th displacement the UGV transfers the data collected from the node $i \in \mathcal{A}$

The problem is formulated as follows

$$\min_{x,y} \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{E}} x_{ij}^k t_{ij} \tag{1}$$

st:

$$\sum_{\{j \in \mathcal{A} \cup \mathcal{B} : (s,j) \in \mathcal{E}\}} x_{sj}^1 = \sum_{\{i \in \mathcal{A} \cup \mathcal{B} : (i,s) \in \mathcal{E}\}} x_{is}^K = 1, \tag{2}$$

$$\sum_{(i,j)\in\mathcal{E}} x_{ij}^k = 1 \qquad \forall k \in \mathcal{K}, \tag{3}$$

$$\sum_{k\in\mathcal{K}} y_i^k = 1 \qquad \forall i \in \mathcal{A}, \tag{4}$$

$$\sum_{k\in\mathcal{K}} \sum_{\{i\in\mathcal{A}\cup\mathcal{B}:(i,j)\in\mathcal{E}\}} x_{ij}^k \geq \begin{cases} 2 & \forall j \in \mathcal{A}\setminus\mathcal{B}, \\ 1 & \forall j \in \mathcal{A}\cap\mathcal{B}, \end{cases} \tag{5}$$

$$x_{ij}^{k-1} \leq \sum_{\{i\in\mathcal{A}\cup\mathcal{B}:(j,i)\in\mathcal{E}\}} x_{ji}^k \qquad \forall (i,j) \in \mathcal{E} \quad \forall k \in (\mathcal{K}\setminus\{1\}), \tag{6}$$

$$y_i^k \leq \sum_{\{j\in\mathcal{B}:(i,j)\in\mathcal{E}\}} x_{ij}^k \qquad \forall i \in \mathcal{A} \quad \forall k \in \mathcal{K}, \tag{7}$$

$$y_j^k \leq \sum_{1\leq t\leq k} \sum_{\{i\in\mathcal{A}\cup\mathcal{B}:(i,j)\in\mathcal{E}\}} x_{ij}^t \qquad \forall j \in \mathcal{A} \quad \forall k \in \mathcal{K}, \tag{8}$$

$$y_j^k \leq \begin{cases} \sum_{k+1\leq t\leq K} \sum_{\{i\in\mathcal{A}\cup\mathcal{B}:(i,j)\in\mathcal{E}\}} x_{ij}^t & \forall j \in \mathcal{A}\setminus\mathcal{B} \quad \forall k \in \mathcal{K}\setminus\{\mathrm{K}\}, \\ 0 & \forall j \in \mathcal{A}\setminus\mathcal{B} \quad k = K \end{cases} \tag{9}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall (i,j) \in \mathcal{E} \quad \forall k \in \mathcal{K}, \tag{10}$$

$$y_i^k \in \{0,1\} \qquad \forall i \in \mathcal{A} \quad \forall k \in \mathcal{K}, \tag{11}$$

Eq. (1) requires the minimization of the total time required to perform all the operations, Eq. (2) imposes that the path of the UGV should start and end at the depot. The constraints in Eq. (3) ensure that each displacement of the UGV corresponds to one edge in the solution and Eq. (4) requires that the data collected from any node $i \in \mathcal{A}$ are transmitted once and only once. The constraints in Eq. (5) ensure that the UGV visits, i) at least twice, the nodes of interest in the set $\mathcal{A}\setminus\mathcal{B}$ where it is not possible to communicate and ii) at least once, the other nodes of interest in the set $\mathcal{A}\cap\mathcal{B}$. Eq. (6) are flow conservation constraints. The fact that the UGV might transfer from a node only if such node belongs to the set $\mathcal{B}$ is given by Eq. (7). Eq. (8) requires that the UGV transfers the information related to a given node in the set $\mathcal{A}$ only if such node has already been visited, Eq. (9) requires that, once the information corresponding to the visit of a node in the set $\mathcal{A}\setminus\mathcal{B}$ where communication is not possible has been transmitted, the UGV must visit the node again once more. Finally Eqs. (10) and (11) ensures that the variables of the optimization problem have a binary value.

One of the main reason of the complexity of the problem in Eqs. (1)–(11) is that it includes the parameter $K$ whose value is unknown a priori. $K$ actually represents the number of displacements the UGV needs to perform before completing the required tasks. This issue could be solved by assigning a very large value to $K$ and allowing the UGV not to move to a new node to match the $K$ value. Unfortunately, the number of binary variables in the problem increases with $K$, therefore the complexity of the problem easily increases. On the other hand, if the value of $K$ is underestimated the problem might become infeasible or the model may lead to a feasible solution that is not the optimal one. Another alternative to tackle such difficulty could be the column generation approach (CGA) whose implementation requires to formulate a restrictive master problem (RMP) associated with the problem in Eqs. (1)–(11) and progressively add new columns and thus new variables to the RMP until the minimum number of variables necessary to get the optimal solution is reached. More details about the CGA can be found in Wilhelm (2001); Zhao et al. (2018). The main drawback of this approach is the difficulty of getting a RMP associated with a given integer linear programming problem that guarantees both the effectiveness and the efficiency of the CGA.

The minimum value for $K$ needed to obtain the optimal solution is strongly related to the topology of the underlying graph of the problem, nevertheless the following proposition holds:

**Proposition 4.1.** *Let $L$ denote the number of edges that constitutes the shortest path between a pair of nodes in $\mathcal{V}$ having the largest number of edges and $AB$ the cardinality of $\mathcal{A}\cap\mathcal{B}$. If $K \geq L(3A - 2AB + 1)$ then the solution provided by the model in Eqs. (1)–(11) is the optimal one.*

**Proof.** First, consider the case in which the graph is complete and thus $L = 1$. Denoting by $N_i$ the total number of times the UGV visits and leaves a node $i \in \mathcal{V}$ before any solution is found, it follows that

$$\sum_{i\in\mathcal{V}} N_i = 2K. \tag{12}$$

Since each node $a \in \mathcal{A}\setminus\mathcal{B}$ has to be visited twice and that since the graph is complete, extra visits cannot occur in $a$, therefore

$$N_a = 4. \tag{13}$$

Furthermore, a node $b \in \mathcal{B}$ is visited to transmit the data collected at one or more nodes $a \in \mathcal{A}$, hence

$$\sum_{b\in\mathcal{B}} N_b \leq 2A. \tag{14}$$

Using (13) and (14) and considering that the UGV needs to leave the depot $s$ and return back there at the end of the tour (thus 2 more displacements are to be added to $N_s$):

$$\begin{aligned} 2K = \sum_{i\in\mathcal{V}} N_i &= \sum_{a\in\mathcal{A}\setminus\mathcal{B}} N_a + \sum_{b\in\mathcal{B}} N_b \leq 4(A - AB) + 2A + 2 \\ &= 6A - 4AB + 2 \end{aligned}$$

Hence if $K \geq 3A - 2AB + 1$ any feasible solution will be explored. In the more general case in which the graph is connected but not complete, extra visits to a given node than those needed for data collection, transmission or to perform some action are expected to occur. For instance the UGV might move from a B node to another B node to reach an A node. By definition of $L$, any operation requiring a single displacement, when the graph is complete, can be performed after at most $L$ displacements of the UGV in case the graph is connected but not complete. Hence, all the feasible solutions will be examined if

$$K \geq L(3A - 2AB + 1). \tag{15}$$

□

### 4.2. Dummy-Vertices-based formulation (Model II)

In this section, we explore an alternative formulation of the problem. Following the approach of Schiffer and Walther (2017) we use the so-called dummy vertices to take into account that vertices can be visited more than once. Let us define the following elements:

- $d$ the number of dummy replications of each vertex
- $\mathcal{V}_d$ set of all vertices including the dummy vertices
- $\mathcal{E}_d$ set of all edges including the dummy edges
- $\mathcal{D}_i = \{i_1, i_2, \ldots, i_d\}$ set of the $d$ dummy vertices associated with the vertex $i \in \mathcal{V}$, e.g., $s_1$ and $s_d$ are respectively the first and the last dummy vertex associated to the depot $s$
- $x_{ij}$=1 if the arc $(i,j) \in \mathcal{E}_d$ is in the solution and 0 otherwise
- $\tau_i$ the time at which the vertex $i \in \mathcal{V}_d$ is visited
- $y_{ij}$ =0 if $\tau_j \notin [\tau_{i_1}\, \tau_{i_d}]$ $\forall i \in \mathcal{A}\setminus\mathcal{B}$, $j \in \mathcal{B}_d$
- $T = d\sum_{(i,j)\in\mathcal{E}} t_{ij}$

The problem can then be alternatively formulated as follows:

$$\min_{x,y} \sum_{(i,j)\in\mathcal{E}_d} x_{ij}t_{ij} \tag{16}$$

st:

$$\sum_{i\in\mathcal{V}_d,(i,j)\in\mathcal{E}_d} x_{ij} \le 1 \qquad \forall j \in \mathcal{V}_d \tag{17}$$

$$\sum_{j\in\mathcal{V}_d,(i,j)\in\mathcal{E}_d} x_{ij} - \sum_{j\in\mathcal{V}_d,(i,j)\in\mathcal{E},} x_{ji} = 0 \qquad \forall i \in \mathcal{V}_d, i \neq s_1, i \neq s_d \tag{18}$$

$$\sum_{i\in\mathcal{V}_d,(i,j)\in\mathcal{E}_d} x_{ij} \ge 1 \qquad \forall j \in \mathcal{A}\cap\mathcal{B} \tag{19}$$

$$\sum_{j\in\mathcal{V}_d,(j,i_1)\in\mathcal{E}_d} x_{ji_1} \ge 1 \qquad \forall i \in \mathcal{A}\setminus\mathcal{B} \tag{20}$$

$$\sum_{j\in\mathcal{V}_d,(j,i_d)\in\mathcal{E}_d} x_{ji_d} \ge 1 \qquad \forall i \in \mathcal{A}\setminus\mathcal{B} \tag{21}$$

$$\tau_{i_k} + T\left(1 - \sum_{j\in\mathcal{V}_d,(j,i)\in\mathcal{E}_d} x_{ji_k}\right) \ge \tau_{i_1} \qquad \forall i \in \mathcal{A}\setminus\mathcal{B}, k = 1,2,\dots,d \tag{22}$$

$$\tau_{i_k} - T\left(1 - \sum_{j\in\mathcal{V}_d,(j,i)\in\mathcal{E}_d} x_{ji_k}\right) \le \tau_{i_d} \qquad \forall i \in \mathcal{A}\setminus\mathcal{B}, k = 1,2,\dots,d \tag{23}$$

$$\sum_{j\in\mathcal{B}_d} y_{ij} \ge 1 \qquad \forall i \in \mathcal{A}\setminus\mathcal{B} \tag{24}$$

$$\sum_{i\in\mathcal{V}_d,(i,j)\in\mathcal{E}_d} x_{ij} \ge y_{kj} \qquad \forall j \in \mathcal{B}_d, k \in \mathcal{A}\setminus\mathcal{B} \tag{25}$$

$$\tau_{i_1} - T(1-y_{ij}) \le \tau_j \le \tau_{i_d} + T(1-y_{ij}) \quad i \in \mathcal{A}\setminus\mathcal{B}, j \in \mathcal{B}_d \tag{26}$$

$$\tau_i + t_{ij}x_{ij} - T(1-x_{ij}) \le \tau_j \quad (i,j) \in \mathcal{E}_d \tag{27}$$

$$\sum_{j\in\mathcal{V}_d,(j,s_1)\in\mathcal{E}_d} x_{s_1 j} = 1 \tag{28}$$

$$\sum_{j\in\mathcal{V}_d,(j,s_d)\in\mathcal{E}_d} x_{js_d} = 1 \tag{29}$$

$$\tau_{s_1} - T\left(1 - \sum_{i\in\mathcal{V}_d,(i,j)\in\mathcal{E}_d} x_{ij}\right) \le \tau_j \le \tau_{s_d} + T\left(1 - \sum_{i\in\mathcal{V}_d,(i,j)\in\mathcal{E}_d} x_{ij}\right) \quad j \in \mathcal{V}_d, j \neq s_1, j \neq s_d \tag{30}$$

$$x_{ij} \in \{0,1\} \quad (i,j) \in \mathcal{E}_d \tag{31}$$

$$y_{ij} \in \{0,1\} \quad i \in \mathcal{A}\setminus\mathcal{B}, j \in \mathcal{B}_d \tag{32}$$

$$\tau_i \in [0,+\infty) \quad i \in \mathcal{V}_d \tag{33}$$

The model can be interpreted as follows. The minimization of the time required to complete the tour is expressed in (16). The constraints (17) state that any dummy node should be visited at most once. Constraints (18) establish flow conservation by requiring that the number of incoming arcs of each node (except the vertices $s_1$ and $s_d$) is equal to the outgoing ones. Constraints (19) enforce that each node of type AB is visited at least once. The constraints (20)–(23) handle the two visits needed in each node of type A where data transmission is not allowed. In fact $\forall i \in \mathcal{A}\setminus\mathcal{B}$ it is required that the associated dummy nodes $i_1$ and $i_d$ should be visited. Furthermore, the first and the last visit in $i$ should occur using respectively $i_1$ and $i_d$. Constraints (24) and (25) ensure that the data collected in each node $i \in \mathcal{A}\setminus\mathcal{B}$ are transmitted from some node $j \in \mathcal{B}_d$ before the last visit to node $i$ occurs, while (26) models the relation between the variables $y_{ij}$, $\tau_j$, $\tau_{i_1}$, $\tau_{i_d}$. Time constraints are expressed in (27). The constraints (28) and (29) enforce the connectivity between the starting location and the other nodes. Constraints (30) require that the tour starts and ends at the depot $s$. Finally, (31)–(33) determine the domain of the decision variables.

## 5. Proposed solution

To avoid an unreasonable increase in the number of decision variables, we propose to employ a graph visiting algorithm coupled with a branch and bound approach so that all possible solutions are explored while complexity is minimized. For the sake of convenience we remind the goal of the problem. Starting from the depot $s$, the UGV must visit every node $a \in \mathcal{A}$, acquire some data, find a location $b \in \mathcal{B}$ where the data can be sent, receive instructions, go back to $a$, perform the required task and, at the end, return back to $s$ spending the lowest possible amount of time.

Let us denote by $a_1$, $a_2$, ..., $a_A$ the $A$ nodes to be visited. While the UGV is moving through the graph, the state of the UGV itself is completely defined by

$$\mathbb{S} = (n_1, n_2, \dots, n_A, u, c)$$

where $u$ is the node the UGV is visiting, $c$ is the total cost of the edges traversed so far and

$$n_i = \begin{cases} 0 & \text{if the node } a_i \text{ has not yet been visited} \\ 1 & \text{if the node } a_i \text{ has been visited} \\ 2 & \text{if the data associated with node } a_i \text{ has been transmitted} \\ 3 & \text{if the task required for the node } a_i \text{ has been performed} \end{cases}$$

**Definition 5.1.** The state $\mathbb{S} = (n_1, n_2, \dots, n_A, u, c)$ is said to be a feasible solution of the problem if $u = s$ and

$$n_i = 3 \quad \forall i \in \{1,2,\dots,A\}$$

The state $\mathbb{S} = (n_1, n_2, \dots, n_A, u, c)$ is branched taking into consideration all possible displacements of the UGV from the node $u$ to any other node connected to $u$ by one edge and this leads to the construction of the state tree as illustrated in Example 5.1.

**Example 5.1.** Consider the network in Fig. 1 in which the node $a_1 \in \mathcal{A}\setminus\mathcal{B}$, $a_2b_2 \in \mathcal{A}\cap\mathcal{B}$, $s \in \mathcal{B}$ and $b_1 \in \mathcal{B}$. The network shows two nodes of interest $a_1$ and $a_2b_2$ and hence the generic state is given by ($n_1$, $n_2$, $u$, $c$) in which $n_1$ and $n_2$ are associated with $a_1$ and $a_2b_2$ (see Eq. ()) In Fig. 2 we represent part of the tree of the states that are generated by the algorithm. The root of the tree is the state (0, 0, $s$, 0) corresponding to the initial location. From node $s$, if the

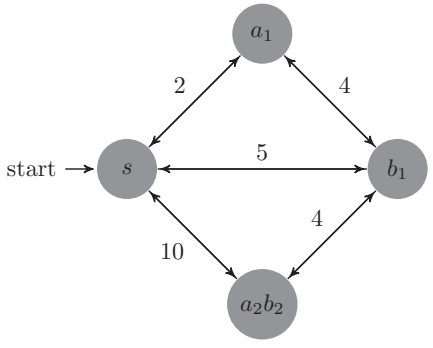


**Fig. 1.** Network of the example 5.1.

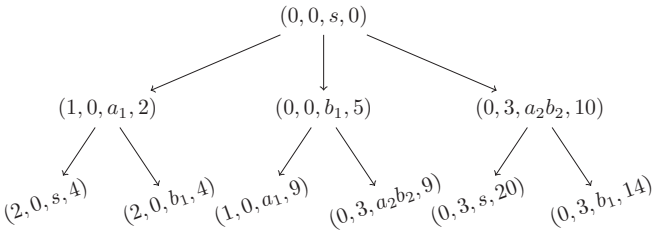


**Fig. 2.** State tree of the example 5.1.

UGV moves to the node $a_1$, $n_1$ takes value 1 and the new state will be (1, 0, $a_1$, 2); if the UGV moves to node $b_1$, the new state will be (0, 0, $b_1$, 5). Finally, if the UGV moves to node $a_2b_2$, since communication is possible there, the UGV will send the corresponding information and then it will perform the required task. Hence $n_3$ will assume value 3 and the state will be (0, 3, $a_2b_2$, 10). Such reasoning is applied recursively to all the newly generated states. Fig. 2 shows also the next expansion for the next level of the visit.

The algorithm implements two main operations: branch and bound. The first one (branch) consists in expanding the state tree by generating new states from its leaves, as illustrated in Example 5.1, and the second one (bound) consists in reducing the state tree by means of a pruning procedure that terminates the visit on the leaves of the state tree that can not lead to the optimal solution. The decision on what leaf can or can not lead to an optimal solution, i.e. pruning, is based on the following two rules:

1. If the lower bound associated with any state among those to be expanded (the leaves) is greater than the cost of any already computed feasible solution, such state is closed, i.e., it is no more considered for branching.
2. If any state among those to be branched differs from a state $\mathbb{S}$ already reached just for its cost, such state is closed if its cost is greater than the one of $\mathbb{S}$.

In order to derive a lower bound associated with a state $\mathbb{S} = (n_1, n_2, \ldots, n_A, u, c)$, first let us consider the set

$$\mathcal{V}_{\mathbb{S}} = \{a \in \mathcal{A} \mid n_a \neq 3\}.$$

At the state $\mathbb{S}$, each node $a \in \mathcal{V}_{\mathbb{S}} \setminus \mathcal{B}$ such that $n_a = 0$ is still to be visited at least twice and those with $n_a = 1$ or $n_a = 2$ should still be visited at least once. Thus, in (34), each node $a \in \mathcal{V}_{\mathbb{S}} \setminus \mathcal{B}$ has still to be visited at least $2 - \min(n_a, 2)$ times. Furthermore, each node $b \in \mathcal{V}_{\mathbb{S}} \cap \mathcal{B}$ still has to be visited at least once. The information acquired in each node $a \in \mathcal{V}_{\mathbb{S}} \setminus \mathcal{B}$ has then to be transmitted by visiting some of the nodes $b \in \mathcal{B}$ and, finally, the UGV has to go back to the depot $s$. Since we are searching for a lower bound of the time needed to perform the aforementioned operations, we relax the flow conservation constraints, hence we assume that: i) the UGV visits and leaves each node $j \in \mathcal{V}_{\mathbb{S}}$ using the cheapest edge connected to it ($\min_{i \in \mathcal{V}} t_{ij}$); ii) the transmissions occurs only once at some node $b \in \mathcal{V}_{\mathbb{S}} \cap \mathcal{B}$ if there is any, otherwise it occurs at the node $b \in \mathcal{B}$ that is reachable from any node that is still to be visited assuming to spend the smallest possible time ($\min_{i \in \mathcal{V}_{\mathbb{S}}, j \in \mathcal{B}} t_{ij}$); iii) the UGV will go back to the depot using the lowest amount of time needed to reach it from a node $i \in \mathcal{V}_{\mathbb{S}}$ ($\min_{i \in \mathcal{V}_{\mathbb{S}}} t_{is}$). Such relaxations clearly lead to a total time that is smaller than or equal to the time that is actually needed and hence any solution derived from the state $\mathbb{S}$ is lower bounded by

$$LB_{\mathbb{S}} = c + \left( \sum_{a \in \mathcal{V}_{\mathbb{S}} \setminus \mathcal{B}} (2 - \min(n_a, 1)) \min_{i \in \mathcal{V}} t_{ia} \right) + \left( \sum_{b \in \mathcal{V}_{\mathbb{S}} \cap \mathcal{B}} \min_{i \in \mathcal{V}} t_{ib} \right) + (1 - \min(1, |\mathcal{V}_{\mathbb{S}} \cap \mathcal{B}|)) * \min_{i \in \mathcal{V}_{\mathbb{S}}, j \in \mathcal{B}} t_{ij} + \min_{i \in \mathcal{V}_{\mathbb{S}}} t_{is}. \quad (34)$$

It is worth noting that the lower bound proposed in (34) is not particularly complex nor computationally expensive. Therefore it can be considered a reasonably simple example of how the proposed algorithm can be improved. Nevertheless, the use of such lower bound can systematically reduce the time needed to find an optimal solution, as shown in Section 6.

Another example about how the proposed algorithm can be improved is to make available to the algorithm itself an initial feasible solution so that the lower bound can be effectively used right from the start. Such initial feasible solution is computed using a nearest neighbour algorithm that implements the steps in Algorithm 1.

**Algorithm 1** Compute Initial Feasible Solution.

1: run the Bellman Ford algorithm to find the shortest path between each pair of nodes $(i, j)$, $j \in \mathcal{V}$, $i \in \mathcal{V}$ and its total cost $c_{ij}$;
2: $\mathcal{T} = \mathcal{A}$, C=0;
3: find the node $\hat{a} \in \mathcal{T}$ such that $c_{s\hat{a}} = \min_{a \in \mathcal{A}} c_{sa}$ and set $C$=$C + c_{s\hat{a}}$
4: $\mathcal{T} = \mathcal{T} \setminus \{\hat{a}\}$
5: if $\mathcal{T} \neq \emptyset$, find the node $b \in \mathcal{T}$ such that $c_{\hat{a}b} = \min_{i \in \mathcal{T}} c_{\hat{a}i}$; set $C$=$C + c_{\hat{a}b}$, let $\hat{a} = b$, go back to step 4.
6: find the node $i \in \mathcal{B}$ such that $c_{\hat{a}i} = \min_{j \in \mathcal{B}} c_{\hat{a}j}$; set $C$=$C + c_{\hat{a}i}$
7: $C = 2C$
8: return C

The underlying idea is the following: starting from the depot $s$, the UGV chooses and visits the closest type A node (i.e., the one reachable in the lowest possible amount of time); then, from that node, it selects the next node in the set of those to be visited using the same criteria; this process is repeated until all nodes of type A are visited. Then, a type B node is visited to transmit the acquired data. Finally, all the nodes are visited again following back the path used for data collection, i.e., the cost is doubled. Clearly, this is not the best solution, but it is a value that can be immediately compared with the computed lower bound also in the initial phases when a feasible solution would not be available otherwise.

Also, another important factor to consider in order to make the algorithm faster is that while expanding the state tree, leaves can be considered in several different orders by means of a priority value. The simplest strategy to assign such a value is to use a monotonically decreasing value each time a new leaf is created. In such a way leaves are considered in creation order. Other more advances strategies are possible and they will be presented and investigated in Section 6.

When branching a leaf $\mathbb{S}$, it may happen that all the states generated from $\mathbb{S}$ are canceled when the two pruning rules are applied. In this case $\mathbb{S}$ is closed and no more considered. Note that closed states also include feasible solutions which clearly do not need to be further expanded. The expansion of the state tree and the pruning procedure are then iteratively applied until all the leaves of the state tree are closed. In order to formally present the algorithm we introduce the following definitions:

- $\mathcal{G}$ the graph associated with the problem;
- $\mathcal{S}_T$ the state tree;
- $N_U$ the number of nodes adjacent to node $U$ in the graph $\mathcal{G}$;
- $\mathcal{L}_g$ the list of all new states generated when branching the leaf having highest priority;
- $\mathcal{L}_p$ the list of all the states in $\mathcal{L}_g$ that are not pruned after the function *Prune*() has been called;
- $\mathcal{F}$ the list containing the closed states;
- {} the empty list
- *Pop*($\mathcal{L}$) a function that takes as input a list of states and returns the state $\mathbb{S}$ with highest priority;
- *Branch*($\mathbb{S}, \mathcal{G}$) a function that takes as input a state $\mathbb{S} = (n_1, n_2, \ldots, n_A, U, C)$ and the graph, branches the received state

and returns a list of $N_U$ states, one for each of the adjacent nodes to $U$ (see Example 5.1);
- *Insert*$(\mathcal{L}_p, \mathcal{S}_T)$ a function that inserts in the state tree the list of states received as input;
- *Prune*$(\mathcal{S}_T, \mathcal{L}_g)$ a function that takes as input the state tree $\mathcal{S}_T$ and the list of newly generated states $\mathcal{L}_g$ then eventually prunes some states from $\mathcal{S}_T$ and $\mathcal{L}_g$ according to the aforementioned pruning rules and returns the list of remaining states from $\mathcal{L}_g$;
- *GetSolution*$(\mathcal{S}_T)$ a function that extracts from the state tree the branch that leads to the leaf state that is a feasible solution with the minimum cost.

The algorithm consists of the steps defined in Algorithm 2.

**Algorithm 2** Compute Optimal Path.

$\mathcal{S}_T = \{(0, 0, \ldots, S, 0)\}$
$\mathcal{F} = \{\}$
**while** $((\mathbb{S} = Pop(\mathcal{S}_T \setminus \mathcal{F})) \neq \{\})$ **do**
  $\mathcal{L}_g = Branch(\mathbb{S}, \mathcal{G})$
  $\mathcal{L}_p = Prune(\mathcal{S}_T, \mathcal{L}_g)$
  **if** $(\mathcal{L}_p == \{\})$ **then**
    $\mathcal{F} = \mathcal{F} \cup \mathbb{S}$
  **else**
    $Insert(\mathcal{L}_p, \mathcal{S}_T)$
  **end if**
**end while**
$Sol = GetSolution(\mathcal{S}_T)$

It is worth noting that the proposed algorithm solves the problem *without making any assumption* neither on the value of $K$ nor on the number of dummy vertices associated with each node. The value of $K$ can be simply computed at the end by just counting the number of displacements performed by the UGV.

## 6. Numerical results and discussion

The numerical experiments conducted in this section aim at comparing the complexity of solving the problems using both the integer and mixed integer linear programming problems represented respectively by Model I and Model II and our proposed algorithm.

Two sample graphs are shown respectively in Fig. 3 and Fig. 4. A different labelling has been adopted for these graphs in order to make them more readable when superimposed on a map. Here we use symbols instead of letters:

- the *squares* represent nodes of type *A*
- the *diamonds* represent nodes of type *B*
- the *stars* represent nodes of type *AB*

While Fig. 3 is a simple example drawn up from scratch, the graph in Fig. 4 represents a more realistic case that has been obtained by mapping a vineyard of the Langhe area in Piedmont

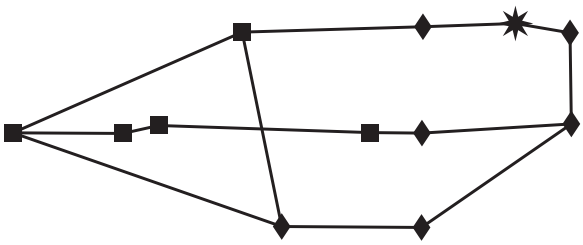

**Fig. 3.** Sample graph drawn from scratch to mimic a possible field. Here the nodes represent: POIs that need to be visited (squares), places where it is possible to communicate (diamonds), or locations that are both (stars).

**Fig. 4.** Undirected graph that represents the available paths for the UGV on a real vineyard. Nodes are marked as described in Fig. 3 and represent the type *A, B, AB* nodes with the addition of some "transit" nodes marked as empty circles.

(Italy) on an undirected graph with 33 nodes and 39 edges representing the paths across the 9 rows of the vineyard.

First, in order to quantify how the effectiveness and efficiency of Model I is affected by the value of $K$, we first find the lowest value of $K$ that yields the optimal solution in Model I by running our branch and bound algorithm. Then, Model I is solved by means of the integer linear programming solver available in the CPLEX software IBM using different estimations of $K$ including the previous one. We remind that the number $K$ of displacements of the UGV required to perform all the tasks in the shortest possible time is dependent on the topology of the considered graph. Choosing $K$ according to (15) guarantees to find the optimal solution. However, to explore the performance provided by commercial solvers such as CPLEX, we tested different $K$ values. To better quantify the different amount of time required as a function of $K$, we propose to evaluate the following ratio:

$$\rho = \frac{T_K}{T_{opt}}$$

where $T_K$ and $T_{opt}$ are the times required by CPLEX to solve the problem using a certain value of $K$ and the lowest $K$ value that yields the optimal solution, respectively. Table 1 presents the results obtained for the graph in Fig. 3. The experiments have been performed using IBM ILOG CPLEX Optimization Studio 12.9.0.0 on a Dell PowerEdge T640 with an Intel Xeon 4114 2.2 GHz 64 bit deca-core processor and 32 GB of DDR4 2400 MT/s memory. If $K$ is less

**Table 1**
Time ($T_K$) taken by the CPLEX solver for the graph in Fig. 3 for different $K$ values in Model I.

| | $K$ | $T_K$ (s) | $\rho$ | Cost |
|---|---|---|---|---|
| underestimation | 10 | 2.16 | 0.12 | – |
| of the $K$ value | 12 | 3.88 | 0.22 | – |
| | 15 | 6.17 | 0.36 | 100 |
| $K$ optimum value | 18 | 17.33 | 1.00 | 94 |
| overestimation | 20 | 40.05 | 2.31 | 94 |
| of the $K$ value | 22 | 120.01 | 6.92 | 94 |
| | $> 25$ | $> 1000$ | $> 57.70$ | 94 |

**Table 2**
Time (s) required to solve Model I with a time limit of two hours. For each number of nodes *N*, ten problem instances (#1 ... #10) have been randomly generated. When the time limit has been reached we report, in brackets, how much the best solution found by the model exceeds the optimal one.

| N | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 14.76 | 493.69 | 0.77 | 83.87 | 0.96 | 469.49 | 3.82 | *(7.30%)* | *(3.11%)* | *(1.02%)* |
| 12 | 2319.38 | 2.60 | 1647.31 | 4.35 | 1647.31 | 4.35 | 3.52 | *(7.30%)* | *(0.00%)* | *(2.18%)* |
| 13 | 7.61 | 40.61 | 4090.54 | 823.92 | 346.69 | 7.34 | *(7.47%)* | *(4.57%)* | *(0.54%)* | *(0.00%)* |
| 14 | 4670.32 | 18.86 | 856.01 | *(7.59%)* | *(0.61%)* | *(1.56%)* | *(8.47%)* | *(14.97%)* | *(3.39%)* | *(5.26%)* |
| 15 | 1723.71 | 2057.74 | 4100.92 | 429.9 | *(0.00%)* | *(0.00%)* | *(0.00%)* | *(14.33%)* | *(30.49%)* | *(5.72%)* |
| 16 | 2700.44 | 3920.32 | *(3.83%)* | *(3.38%)* | *(0.00%)* | *(7.02%)* | *(2.25%)* | *(11.34%)* | *(8.18%)* | *(6.87%)* |
| 17 | *(34.07%)* | *(3.33%)* | *(14.74%)* | *(38.41%)* | *(6.06%)* | *(35.94%)* | *(37.23%)* | *(27.65%)* | *(39.01%)* | *(53.03%)* |
| 18 | *(34.11%)* | *(8.96%)* | *(9.21%)* | *(25.69%)* | *(11.24%)* | *(28.14%)* | *(36.87%)* | *(55.70%)* | *(29.28%)* | *(18.50%)* |

**Table 3**
Time (s) required to solve Model II with a time limit of two hours. For each number of nodes N, the same ten instances of Table 2 have been used. When the time limit has been reached we report, in brackets, how much the best solution found by the model exceeds the optimal one.

| N | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 5.73 | 8.79 | 4.78 | 40.25 | 3.56 | 12.55 | 1.47 | 32.55 | 30.45 | 12.77 |
| 12 | 21.04 | 7.58 | 8.90 | 27.78 | 8.90 | 27.78 | 8.40 | 17.34 | 43.07 | 19.14 |
| 13 | 10.93 | 38.17 | 44.28 | 12.31 | 7.40 | 7.32 | 49.01 | 137.86 | 5.31 | 39.72 |
| 14 | 24.87 | 483.22 | 14.37 | 82.69 | 46.27 | 114.50 | 116.87 | 54.01 | 32.55 | 597.56 |
| 15 | 10.63 | 133.05 | 138.65 | 2570.43 | 234.08 | 770.36 | 97.99 | 217.67 | 532.98 | 1327.23 |
| 16 | 27.95 | 21.52 | 454.72 | 436.22 | 474.42 | 262.23 | 1757.05 | 1229.74 | 1237.98 | *(0.00%)* |
| 17 | 207.71 | 3261.04 | 626.26 | 1653.33 | 2485.79 | *(11.45%)* | *(18.56%)* | *(7.34%)* | *(22.15%)* | *(8.51%)* |
| 18 | 412.53 | 1058.85 | 2258.76 | *(11.02%)* | *(9.62%)* | *(12.41%)* | *(10.20%)* | *(12.41%)* | *(9.47%)* | *(22.74%)* |

than 18, i.e., the lowest value that allows to compute the optimal solution, the CPLEX solver either finds a cost higher than the optimal one or no solution at all. When $K = 18$, the CPLEX solver finds the optimal solution, as expected. A rather small overestimation of $K$, e.g., $K = 22$ instead of $K = 18$, makes CPLEX about 7 times slower, and even worse for higher values.

To present a more comprehensive set of results, we now focus our attention on the topology in Fig. 4 which is far more complex than the one in Fig. 3. Due to our interest in the agricultural application, this topology will be taken as the basis to randomly generate many different graphs with a variable number of nodes to visit and thus different complexity. In practice, starting from the full graph, some of its type *A* nodes are transformed into "transit" nodes in which the UGV must not perform any operation (they are neither of type *A* or *B* or *AB*). Also, in all experiments, the time required to cover an edge ($t_{ij}$) is directly proportional to the length of the edge itself.

More precisely, Table 2 and 3 report the time taken by CPLEX to find the optimal solution using Model I and Model II, for different combinations of number of labelled nodes *N* (hence the sum of *A, B* and *AB* nodes) and position of such nodes. When CPLEX exceeded the given time limit of two hours, we report a percentage in brackets (instead of the duration of the computation) that represents how much the best solution found by the model exceeds the optimal solution. Also, note that all the values shown in the tables have been obtained in the best possible conditions, i.e., when the value of *K* leading to the optimal solution is known (for Model I) and when the number of dummy vertices to use is known (for Model II).

The results in Tables 2 and 3 show that Model II performs better than Model I almost all the times. For Model I the execution time is quite large and can exceed the given time limit also for relatively small graph instances. Both models, when interrupted after two hours of computation, are able to provide an intermediate best solution that in some cases is very close to the optimal one, but that tends to derive more and more for complex graph instances. Our proposal, instead, is always able to compute a solution in the given time limit and, when the CPLEX Model II is able to find a solution, performs 20 to 30 times faster than that.

Regarding our proposed algorithm, four cases have been considered. They are named *normal, reduced, advanced*, and *lower bound* in the following. The *normal* approach corresponds to the algorithm that runs on the graph as built from the real sample vineyard shown in Fig. 4. In such a graph we have reported also some additional nodes (marked as circles in the figure) to represent the path of the UGV in the vineyard and not only the connections between the nodes of type *A, B, AB*. We define such nodes as "transit" nodes because in our algorithm, when such nodes are visited, the state remains unchanged except for the cost of the path that is updated consequently. Clearly, introducing new nodes into a graph increases the total number of nodes which has a direct impact on the problem's complexity.

Therefore, in order to minimize the complexity, the graph from Fig. 4 can be reduced by producing an equivalent "reduced" graph that has less nodes because all the nodes that are just transit nodes have been substituted by direct edges between the adjacent nodes, so that the same topology is maintained but the number of nodes is reduced. This is referred to as the *reduced* approach in the latter, which always runs on the "reduced" graph version.

The *advanced* approach is similar to the *reduced* approach, i.e., it runs on the "reduced" graph, but the priority value described in Section 5 has been changed. The priority is set to the inverse of the cost of the state (named *costonly* in the following) thus the state tree is always expanded from the leaf state with the lower cost. Following the minimum cost path, and not a path given by the ordering of the nodes, the pruning function is more efficient and a larger number of states are discarded by the prune rules because the algorithm already found an equivalent state with a lower cost.

Finally, the *lower bound* approach works as the *advanced* one but at each new state it computes a lower bound, as previously described in Section 5, to prune branches faster, hence to reduce execution time.

Results are reported in Table 4 as the average on 50 different random graphs, for different numbers of nodes *N*. As it might be expected, the complexity scales exponentially with the number of nodes to be visited (i.e., transmitted and re-visited). However, the cost of finding the optimal solution is heavily reduced if the *advanced* approach is employed. In fact, in the *reduced* approach,

**Table 4**
Complexity of the various approaches, in terms of number of nodes in the state tree ($\mathcal{S}_T$ size) and execution time (seconds). Each row reports the average on 50 different random instances, for different numbers of nodes $N$.

| | *normal* | | *reduced* | | *advanced* | | *lower bound* | |
|---|---|---|---|---|---|---|---|---|
| $N$ | $\mathcal{S}_T$ size | time | $\mathcal{S}_T$ size | time | $\mathcal{S}_T$ size | time | $\mathcal{S}_T$ size | time |
| 11 | 52,529 | 3.0 | 5879 | 0.5 | 1121 | 0.1 | 852 | 0.1 |
| 12 | 258,049 | 15.2 | 34,623 | 3.7 | 4275 | 0.5 | 3222 | 0.4 |
| 13 | 979,446 | 60.4 | 158,299 | 16.8 | 16,957 | 1.9 | 12,842 | 1.7 |
| 14 | 4,018,175 | 267.7 | 715,881 | 76.2 | 65,044 | 8.1 | 45,478 | 6.3 |
| 15 | 14,663,488 | 1014.4 | 2,789,828 | 279.7 | 245,102 | 30.1 | 164,629 | 21.8 |
| 16 | 55,298,726 | 4135.3 | 12,438,650 | 1308.6 | 988,588 | 128.0 | 625,578 | 86.7 |
| 17 | - | - | 57,007,645 | 5906.8 | 3,631,947 | 475.6 | 2,866,574 | 407.4 |
| 18 | - | - | - | - | 10,687,414 | 1388.2 | 7,998,251 | 1127.1 |

**Table 5**
Time (s) required to solve the problem using the best variant of the proposed algorithm (with lower bounds). For each number of nodes N, the same ten instances of Table 2 have been used.

| N | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 0.09 | 0.12 | 0.09 | 0.06 | 0.10 | 0.07 | 0.08 | 0.11 | 0.11 | 0.12 |
| 12 | 0.38 | 0.37 | 0.45 | 0.46 | 0.45 | 0.46 | 0.36 | 0.42 | 0.48 | 0.48 |
| 13 | 1.25 | 1.07 | 1.65 | 2.03 | 1.88 | 1.54 | 1.41 | 1.82 | 2.36 | 2.19 |
| 14 | 6.94 | 7.67 | 6.73 | 8.13 | 5.09 | 7.41 | 6.42 | 9.97 | 6.43 | 4.60 |
| 15 | 21.65 | 16.52 | 17.45 | 22.96 | 19.00 | 21.05 | 17.31 | 22.87 | 29.73 | 29.83 |
| 16 | 83.43 | 66.34 | 82.52 | 107.13 | 114.55 | 76.65 | 81.24 | 94.96 | 76.73 | 86.66 |
| 17 | 411.03 | 507.35 | 352.81 | 328.25 | 391.64 | 446.76 | 409.63 | 303.40 | 404.87 | 449.02 |
| 18 | 1057.29 | 1067.60 | 905.80 | 1374.41 | 1371.11 | 1432.13 | 872.12 | 1200.19 | 1232.31 | 1054.64 |

every time a new node to be visited is added to the graph, the size of the state tree is increased by a factor of 4.40, while in the *advanced* approach only by a factor of 3.71. Here the $S_T$ size corresponds to the number of nodes in the state tree as defined in Section 5. This value equals the number of times the loop in Algorithm 2 is run, thus it is directly proportional to the average execution time, also shown in Table 4. Additional time reductions are possible if lower bound techniques are used to prune states faster while exploring them, as done by the *lower bound* approach, or if an initial feasible solution is provided right at the start of the algorithm. While the lower bound consistently allows to reduce the computation time with all the instances, the initial solution does not always provide a significant gain because the priority used to visit the nodes is already based on a minimum cost criterion.

To better visualize the data, we also plot the time (on a logarithmic scale) as a function of $N$ in Fig. 5. It is clear that the best approach is the *lower bound* one, which can consistently outperform the others, keeping approximately the same distance from the *advanced* one in terms of relative time reduction, which is about 20%. We did not test graphs with more than $N = 18$ because in that case also our algorithm would have rapidly exceeded the two hours time bound we used for the experiments. In fact, the exponential trend at which the computation time increases can be clearly deduced from Fig. 5.

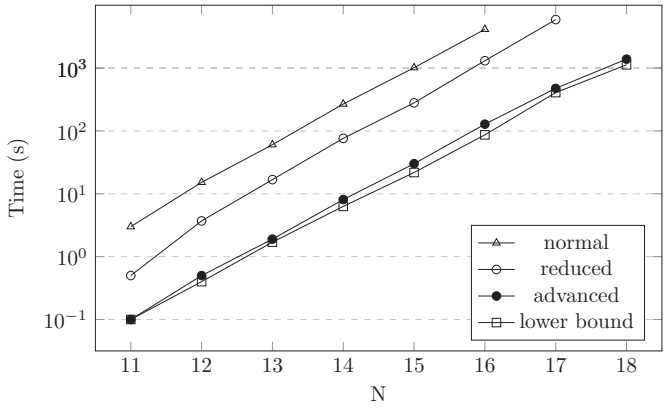


**Fig. 5.** Time required by the different approaches of our proposed algorithm, as a function of the number of nodes $N$.

Table 5 shows the individual times required by the best variant (*lower bound)* of our proposed algorithm for each number of nodes $N$, on the same ten different randomly generated graphs as in Tables 2 and 3. It is clear that our algorithm provides lower execution time. Moreover, it does not require to estimate any other parameter to be fed as input as it happens for Model I and Model II. Despite the execution time of the ILP formulations can, occasionally, be somehow close to the one of our algorithm, Table 5 clearly shows that our algorithm exhibits much more consistency across different problem instances, which is not the case for the ILP formulations. For example, the CPLEX solver using Model II happens to exceed the given two hours time bound starting from $N = 16$ and at more than half of the times for the more complex graph instances. For the less complex instances we computed that, on average (over the 50 tested instances), our proposed algorithm is about 20 to 30 times faster as it can be seen also by comparing the results for the few instances presented in Tables 3 and 5.

Note also that our algorithm has been implemented, to speed up development, in the *python* scripting language, which is not particularly optimized for speed. Therefore the shown time difference could potentially be improved by rewriting our algorithm in native code.

Note that in this work we always focused on finding the guaranteed optimal solution to the problem. In fact, also with the *advanced* and *lower bound* approaches, the optimality of the solution is preserved.

In order to present a more comprehensive view of the algorithm, we also investigated how fast the algorithm converges towards the optimal solution. Therefore, knowing the cost value of the optimal solution, we plotted how far is the current solution at each step of the algorithm with respect to the optimal one. We compared two different priority functions, i.e., the *costonly* one

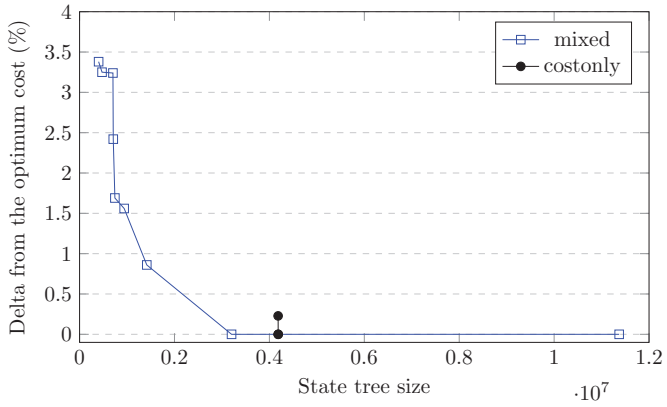


**Fig. 6.** Analysis of the optimality of the intermediate solutions found using two different priority measures. The *costonly* measure terminates earlier with a small total state tree size, but finds a feasible solution only at the very end of the execution. The *mixed* measure makes the algorithm move faster towards a (suboptimal) feasible solution, but delays its completion.

which has just been described, and the *mixed* one where the priority is defined as follows

$$\pi_{\mathbb{S}} = \frac{\frac{1}{3A}\sum_{i=1}^{A} n_i}{c} \tag{35}$$

where the state is $\mathbb{S} = (n_1, n_2, \ldots, n_A, u, c)$. The basic idea is that the numerator in (35) provides an estimation of how close a state is from a feasible solution. In fact, considering that $n_i$ can assume increasing values, from 0 (the initial state) to 3 (the final state), the higher the value of the sum, the more the visit is near to its completion. The $\frac{1}{3A}$ coefficient normalizes the numerator for the feasible solution to 1, since $n_i$ ranges from 0 to 3. If the next leaf to be branched is chosen according to (35), it is expected to reach a (somewhat "good") feasible solution faster than in the case we use the *costonly* priority measure. That is because the priority value defined in (35) takes in consideration both how much of the visit has already been completed and how much it costs, thus it may tend to favour a path that completes faster the visit of the nodes, than just a path with the minimum cost.

Fig. 6 shows that, for the priority function in (35), with less than 1/10 of the size of the fully grown state tree, the current solution differs from the optimal one only for less than 3.5%. This is an encouraging result to be explored in future work where more efficient heuristics could be investigated.

## 7. Conclusions

In this work we proposed an algorithm to solve the navigation problem of a UGV that aims to minimize the time needed to perform repeated acquisition, communication and action tasks on a predetermined set of positions. Starting from the graph, we proposed both a modeling approach based on integer linear programming, to be tackled through commercial solvers, and a branch and bound algorithm specifically designed for the problem. The algorithm has been explained in details by means of detailed pseudocode and examples. Results showed that the proposed algorithm is able to provide an improvement of 20 to 30 times over commercial linear programming solvers when compared on many instances that can be solved to optimality in reasonable computational time. Tests have been conducted on both synthetically generated input data and data extracted from a real world case. Moreover, faster variants of the algorithm have also been proposed while solution optimality is maintained. While our proposals can help to speed up computation compared to the ILP approach, the complexity still increases exponentially with the number of nodes due to the nature of the problem. Therefore, future work will be devoted to investigate efficient heuristics able to find good solutions with acceptable response times, maybe even able to run on the robots themselves. As a first step in this direction, in this work we also investigated how fast the proposed algorithm converges to the optimal solution, providing an insight into how it is possible to extend this work by designing efficient heuristics.

## Acknowledgments

This work has been supported in part by the Politecnico di Torino Interdepartmental Center for Service Robotics (PIC4SeR) https://pic4ser.polito.it.

## References

Basaca-Preciado, L.C., Sergiyenko, O.Y., Rodríguez-Quinonez, J.C., Garcia, X., Tyrsa, V.V., Rivas-Lopez, M., Hernandez-Balbuena, D., Mercorelli, P., Podrygalo, M., Gurko, A., Tabakova, I., Starostenko, O., 2014. Optical 3D laser measurement system for navigation of autonomous mobile robot. Opt. Laser. Eng. 54, 159–169.

Bechar, A., Vigneault, C., 2016. Agricultural robots for field operations: concepts and components. Biosyst. Eng. 149, 94–111.

Bochtis, D., Sørensen, C., 2010. The vehicle routing problem in field logistics: part II. Biosyst. Eng. 105 (2), 180–188.

Bonadies, S., Lefcourt, A., Gadsden, S.A., 2016. A survey of unmanned ground vehicles with applications to agricultural and environmental sensing. In: Autonomous Air and Ground Sensing Systems for Agricultural Optimization and Phenotyping, 9866. International Society for Optics and Photonics, p. 98660Q.

Bousonville, T., Hartmann, A., Melo, T., Kopfer, H., 2011. Vehicle routing and refueling: the impact of price variations on tour length. In: Logistikmanagement. University of Bamberg Press, pp. 83–101.

Candiago, S., Remondino, F., De Giglio, M., Dubbini, M., Gattelli, M., 2015. Evaluating multispectral images and vegetation indices for precision farming applications from UAV images. Remote Sens. 7 (4), 4026–4047.

Castellini, A., Farinelli, A., Minuto, G., Quaglia, D., Secco, I., Tinivella, F., 2017. EXPO-AGRI: Smart automatic greenhouse control. In: IEEE Biomedical Circuits and Systems Conference (BioCAS), pp. 1–4.

Chang, A., Jung, J., Maeda, M.M., Landivar, J., 2017. Crop height monitoring with digital imagery from unmanned aerial system (UAS). Comput. Electron. Agricult. 141, 232–237.

Comba, L., Gay, P., Primicerio, J., Aimonino, D.R., 2015. Vineyard detection from unmanned aerial systems images. Comput. Electron. Agricult. 114, 78–87.

Contente, O., Lau, N., Morgado, F., Morais, R., 2016. A path planning application for a mountain vineyard autonomous robot. In: Robot 2015: Second Iberian Robotics Conference. Springer, pp. 347–358.

Crevier, B., Cordeau, J.-F., Laporte, G., 2007. The multi-depot vehicle routing problem with inter-depot routes. Eur. J. Opera. Res. 176 (2), 756–773.

Das, J., Cross, G., Qu, C., Makineni, A., Tokekar, P., Mulgaonkar, Y., Kumar, V., 2015. Devices, systems, and methods for automated monitoring enabling precision agriculture. In: IEEE International Conference on Automation Science and Engineering (CASE). IEEE, pp. 462–469.

Dong, J., Burnham, J.G., Boots, B., Rains, G., Dellaert, F., 2017. 4D crop monitoring: Spatio-temporal reconstruction for agriculture. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 3878–3885.

Drexl, M., 2012. Synchronization in vehicle routing-a survey of VRPs with multiple synchronization constraints. Transp. Sci. 46 (3), 297–316.

Eksioglu, B., Volkan. Vural, A., Reisman, A., 2009. The vehicle routing problem: a taxonomic review. Comput. Ind. Eng. 57, 1472–1483.

Fadda, E., Tadei, R., Perboli, G., Tiotsop, L.F., 2018. The multi-path traveling salesman problem with dependent random cost oscillations. In: Seventh Intl. Workshop on Freight Transportation and Logistics (ODYSSEUS), Cagliari, Italy, pp. 368–371.

Ferentinos, K., Arvanitis, K., Sigrimis, N., 2002. Heuristic optimization methods for motion planning of autonomous agricultural vehicles. J. Glob. Optim. 23 (2), 155–170.

Fotio Tiotsop, L., Servetti, A., Masala, E., 2019, to be published. Optimally scheduling complex logistics operations involving acquisition, elaboration and action tasks. In: Proceedings of the 5th International Forum on Research and Technologies for Society and Industry (RTSI), Florence, Italy.

Gasparetto, A., Boscariol, P., Lanzutti, A., Vidoni, R., 2015. Path planning and trajectory planning algorithms: ageneral overview. In: Motion and operation planning of robotic systems. Springer, pp. 3–27.

Huang, Z., Zheng, Q.P., Pasiliao, E., Boginski, V., Zhang, T., 2018. A cutting plane method for risk-constrained traveling salesman problem with random arc costs. J. Glob. Optim..

Khanal, S., Fulton, J., Shearer, S., 2017. An overview of current and potential applications of thermal remote sensing in precision agriculture. Comput. Electron. Agricult. 139, 22–32.

Kim, B.-I., Kim, S., Sahoo, S., 2006. Waste collection vehicle routing problem with time windows. Comput. Opera. Res. 33 (12), 3624–3642.

Laporte, G., 1992. The vehicle routing problem: an overview of exact and approximate algorithms. Eur. J. Opera. Res. 59, 345–358.
Lukas, V., Novák, J., Neudert, L., Svobodova, I., Rodriguez-Moreno, F., Edrees, M., Kren, J., 2016. The combination of UAV survey and landsat imagery for monitoring of crop vigor in precision agriculture. ISPRS-Int. Arch. Photogram. Remote Sens. Spat. Inform. Sci. 41, 953–957.
MacCartney Jr., G.R., Sun, S., Rappaport, T.S., Xing, Y., Yan, H., Koka, J., Wang, R., Yu, D., 2016. Millimeter wave wireless communications: New results for rural connectivity. In: Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges. ACM, New York, NY, USA, pp. 31–36.
Marinakis, Y., Migdalas, A., Pardalos, P.M., 2007. A new bilevel formulation for the vehicle routing problem and a solution method using a genetic algorithm. J. Glob. Optim. 38 (4), 555–580.
Matese, A., Di Gennaro, S.F., 2015. Technology in precision viticulture: a state of the art review. Int. J. Wine Res. 7, 69–81.
Nandi, S., Thota, S., Nag, A., Divyasukhananda, S., Goswami, P., Aravindakshan, A., Rodriguez, R., Mukherjee, B., 2016. Computing for rural empowerment: enabled by last-mile telecommunications. IEEE Commun. Mag. 54 (6), 102–109.
Nex, F., Remondino, F., 2014. UAV For 3D mapping applications: a review. Appl. Geom. 6 (1), 1–15.
Pei, W., Lan, Y., Xiwen, L., Zhiyan, Z., Wang, Z., Wang, Y., 2014. Integrated sensor system for monitoring rice growth conditions based on unmanned ground vehicle system. Int. J. Agricul. Biol. Eng. 7 (2), 75.
Primicerio, J., Caruso, G., Comba, L., Crisci, A., Gay, P., Guidoni, S., Genesio, L., Ricauda Aimonino, D., Vaccari, F.P., 2017. Individual plant definition and missing plant characterization in vineyards from high-resolution UAV imagery. Eur. J. Remote Sens. 50 (1), 179–186.
Salavati-Khoshghalb, M., Gendreau, M., Jabali, O., Rei, W., 2017. A hybrid recourse policy for the vehicle routing problem with stochastic demands. EURO J. Transport. Logist. 1–30.
Schiffer, M., Schneider, M., Walther, G., Laporte, G., 2019. Vehicle routing and location routing with intermediate stops: a review. Transp. Sci. 53 (2), 319–343.
Schiffer, M., Walther, G., 2017. The electric location routing problem with time windows and partial recharging. Eur. J. Opera. Res. 260 (3), 995–1013.
Seyyedhasani, H., Dvorak, J.S., 2017. Using the vehicle routing problem to reduce field completion times with multiple machines. Comput. Electron. Agricult. 134, 142–150.
Seyyedhasani, H., Dvorak, J.S., 2018. Dynamic rerouting of a fleet of vehicles in agricultural operations through a dynamic multiple depot vehicle routing problem representation. Biosyst. Eng. 171, 63–77.
Sona, G., Passoni, D., Pinto, L., Pagliari, D., Masseroni, D., Ortuani, B., Facchi, A., 2016. UAV Multispectral survey to map soil and crop for precision farming applications. Int. Arch. Photogram. Remote Sens. Spat. Inform. Sci. 41, 1023–1029.
Torres-Sánchez, J., Peña, J.M., de Castro, A.I., López-Granados, F., 2014. Multi-temporal mapping of the vegetation fraction in early-season wheat fields using images from UAV. Comput. Electron. Agricult. 103, 104–113.
Vansteenwegen, P., Souffriau, W., Sörensen, K., 2012. The travelling salesperson problem with hotel selection. J. Opera. Res. Soc. 63, 207–217.
Vasudevan, A., Kumar, D.A., Bhuvaneswari, N., 2016. Precision farming using unmanned aerial and ground vehicles. In: Technological Innovations in ICT for Agriculture and Rural Development (TIAR). IEEE, pp. 146–150.
Vu, Q., Raković, M., Delic, V., Ronzhin, A., 2018. Trends in development of UAV-UGV cooperation approaches in precision agriculture. In: Ronzhin, A., Rigoll, G., Meshcheryakov, R. (Eds.), Interactive Collaborative Robotics. Springer International Publishing, Cham, pp. 213–221.
Wilhelm, W.E., 2001. A technical review of column generation in integer programming. Optim. Eng. 2 (2), 159–200.
Zecha, C., Link, J., Claupein, W., 2013. Mobile sensor platforms: categorisation and research applications in precision farming. J. Sens. Sens. Syst. 2 (1), 51–72.
Zhang, H., Tong, W., Xu, Y., Lin, G., 2016. The steiner traveling salesman problem with online advanced edge blockages. Comput. Opera. Res. 70, 26–38.
Zhao, Y., Larsson, T., Rönnberg, E., Pardalos, P.M., 2018. The fixed charge transportation problem: a strong formulation based on lagrangian decomposition and column generation. J. Glob. Optim. 72 (3), 517–538.
Zhou, K., Jensen, A.L., Sørensen, C., Busato, P., Bothtis, D., 2014. Agricultural operations planning in fields with multiple obstacle areas. Comput. Electron. Agricult. 109, 12–22.