Planning a multi-sensors search for a moving target considering traveling costs

Florian Delavernhe, Patrick Jaillet, André Rossi, Marc Sevaux

Version of Record: https://www.sciencedirect.com/science/article/pii/S0377221720309589
Manuscript_57f1d2f8d0db1a6e754bbb3ad4edb296

Planning a Multi-sensors Search for a Moving Target Considering Traveling Costs

Florian Delavernhe[a,b,*], Patrick Jaillet[b], André Rossi[c], Marc Sevaux[d]

[a] *Université d'Angers, LERIA, F-49045 Angers, France*
[b] *Massachusetts Institute of Technology, Cambridge, Massachusetts*
[c] *Université Paris-Dauphine, PSL, LAMSADE, CNRS, UMR 7243, F-75016 Paris, France*
[d] *Université Bretagne Sud, Lab-STICC, CNRS, UMR 6285, F-56321 Lorient, France*

Abstract

This paper addresses the optimization problem of managing the research efforts of a set of sensors in order to minimize the probability of non-detection of a target. A novel formulation of the problem taking into account the traveling costs between the searched areas is proposed; it is more realistic and extends some previous problems addressed in the literature. A greedy heuristic algorithm is devised, it builds a solution gradually, using a linear approximation of the objective function refined at each step. The heuristic algorithm is complemented by a lower bound based on a piecewise linear approximation of the objective function with a parametric error, and extended to the case where the target is moving. Finally, a set of numerical experiments is performed to analyze and evaluate the proposed contributions.

Keywords: combinatorial optimization, multi-sensors search, moving target, non-linear optimization, search theory

1. Introduction

Search theory is a branch of the operations research field, and is devoted to the problem of optimally searching for a missing object or person. It is an important scientific research field that was first addressed during World War II [17] for submarines search and is still relevant today for search and rescue missions [24]. In such missions, an object is missing in a restricted area and multiple agents are sent to locate it with a given limited search capacity. Usually, in the literature of search theory, a distribution of the probabilities for the object to be in various locations is given. The objective is often an exponential function representing the probability of non-detection [4].

One of the problems addressed in this paper is to obtain an optimal distribution of the total search budget (time, energy, etc.) in order to minimize the probability of not detecting a static target. The problem is similar to the one addressed in [20] and [27] where the searched area is partitioned into cells. The search resource of a set of sensors, typically time, is optimally distributed between the cells. The

*Corresponding author
Email addresses: florian.delavernhe@univ-angers.fr (Florian Delavernhe), jaillet@mit.edu (Patrick Jaillet), andre.rossi@lamsade.dauphine.fr (André Rossi), marc.sevaux@univ-ubs.fr (Marc Sevaux)

resulting objective function is computed by summing the probabilities of non-detection in the cells. These probabilities depend on the conditional non-detection probabilities, computed using the search efforts and the visibilities of the cells (*i.e.*, cell-specific coefficients representing how relatively efficient the search effort is in the cells). In [20, 27], the authors are dividing the searched area into zones, which are disjoint sets of cells. Each sensor is allocated to exactly one zone and thus can only search inside the cells of this zone. This results in a hierarchical problem with two levels: find the best allotment of sensors to zones, then find the best resources sharing over the allotted zones. Such a model allows a sensor to freely navigate inside its zone, without restraining, nor penalizing, its movements. In this paper, we address a more realistic and more general aspect of the problem with resources distribution where the sensors movements are penalized. Another questionable consequence of the definition of zones in [20, 27] is that it is impossible for a sensor to move between two cells that are close neighbors without any physical frontier, if they are not in the same zone.

In the problem addressed in this paper, no zones are considered. In addition, the sensors now consume resources to move from a searched cell to another one. Feasible solutions then correspond to an allocation of the total effort available between search activities in the cells and movements needed to travel between them.

The proposed formulation of the static target problem is then extended to the moving target case. In this problem, the activities of the sensors are still the same (*i.e.*, searching cells and traveling between them). However, the target can move over a given time horizon and thus its probability to be in a particular cell is evolving over time. In this problem, the movements of the targets are discretized over the time horizon into time periods. During each time period, the probability of presence of the target in a cell is constant, *i.e.*, we consider that the target is not moving from a cell within a period but only between two consecutive periods. For each period, each sensor has a resource budget to search the target and move. The continuous nature of the sensors activities and the discretization make the problem challenging. Our formulation is a generalization of the previous works with zones as in [20, 21, 27] with static or moving targets.

The contributions of this paper are as follows. First, we introduce a formulation of a new problem, generalizing the static and moving target multi-sensor multi-zone problems to a multi-sensor problem with moving costs. Second, we propose an algorithm to solve the problem that, contrary to previous works in the literature (*e.g.*, [6, 21, 27]), does not rely on a forward-backward method. The solution method uses an approximation of the objective function to compute an utility for each decision variable and constructs a solution step by step. The efficiency of this algorithm is tested and compared to a lower bound also proposed in this work.

This paper is organized as follows. In the very next section, we present the literature and related works in more details. Next, in Section 3 we present the formal definition of our problem. In Section 4 we introduce

the linear approximation used in our algorithm, then we present the algorithm itself in Section 6. In Section 7, we report numerical experiments to assess the efficiency of the proposed method. Finally, in the last section, we conclude the paper and draw some perspectives.

2. Related work

The first work on search theory dates back to 1946, see B.O. Koopman [17] as an extended version published in 1980. Since then, the subject of search theory has drawn a lot of attention from the scientific community as reported by the many surveys published every decades (*e.g.*, [4, 10, 12, 15]). In this section, we give a quick overview of the literature on target search followed by more recent works on search games. Finally, we highlight previous works on resources allocation for target search, for both static and moving targets.

Unmanned Aerial Vehicles (UAVs) are becoming more and more accessible for applications like target search. Many such applications aim at dynamically coordinate a swarm of UAVs to search for one or more targets, possibly followed by a task assignment [8] (*e.g.*, destroy the targets as in [14]). In contrast to our problem, there is no planning ahead of the search activities nor resources allocation. The paths followed by the UAVs are dynamically computed and detection probabilities are directly related to the positions of the targets. For example, in [19], the objective is to minimize time when real-time dynamic decisions are made. These decisions are actions (turn, accelerate, etc) that generate the trajectories followed by the UAVs. The likelihood of a target detection by a UAV is represented by a negative exponential of the distance between the sensor and the prior on the target positions. The priors are assumed known before the mission. In search theory, a prior is a probability of presence of the target on a position. They can be obtained from previous searches, operational considerations, estimations deduced from last position, etc. We will see also (see Section 3.1) how it can be relaxed. In [19], the authors have proposed a novel approach, based on a heuristic future expected reward to reduce the short-sighted aspect of the decision making. The coordination of UAVs for target search has also been based on biologically inspired metaheuristics that mimic swarm or flock behaviors, as shown in [26]. For example, [1] shows a logic emulating biological behavior executed by every UAV. The UAVs are releasing pheromones to locations characterized by some interesting evidences [18]. The movements of the UAVs are impacted by the pheromones near them and their UAV neighbors, similarly to a flock. This strategy provides first a quick survey of the area followed by a better exploration of the interesting areas.

Search games [15] are applications of game theory [30] to search theory. In this setting, the target does not want to be found by the searchers and will react accordingly. Thus, it is hiding while the searchers are looking for it. Search games on networks and two dimensional region are presented in [11]. As shown in this work, the hider (*i.e.*, the target) can be either immobile, or mobile. When immobile, the hider selects an

arbitrary location (a node in the network) and hides there. The objective of the searcher is often to arrive quickly in the same location, or node, than the hider. Many works have been done on the subject and it still interests the scientific community. For example, [13] recently proposed the immobile hider search game with a stochastic network. *i.e.*, at any given time, each edge of the graph can be inactive or active, depending on a known probability law. The authors study the values and strategies available for specific networks, using the knowledge on the deterministic game. Another recent work, [2], treats the problem with an immobile hider, located in a known subset of the network.

The problem of the optimum distribution of a search effort to detect a static target originates from [17]. It has been addressed a few years ago in [20, 27], with multi-zone considerations. In this problem, the searched area is divided into a set of disjoint zones, themselves divided in disjoint cells. The search problem is hierarchical. At an upper level, a best allotment of sensors to zones is sought. Each sensor is allocated to a unique zone, and afterwards is only able to search for the target inside its zone. Note that multiple sensors might be allocated to the same zone, and the authors in [27] consider that an (1:1) injective allotment is easier to solve. At the lower level of the hierarchical problem, the authors are searching for the optimal resources distribution for every sensor. Other characteristics of the problem (the objective function, the target distribution, etc.) are similar to the ones used in our work (see Section 3). In [27], the authors proposed a hierarchical algorithm. The lower level is solved using an iterative method. At each iteration, based on [9], they optimize the resource allocation for one sensor, while the others have their allocation fixed. It iterates until it converges. The upper level is solved using a cross-entropy (CE) method [25]. At each iteration, their CE method draws particular allotments of sensors to zones, then evaluates and selects them using the lower level of their algorithm. Afterwards, their method updates the drawing distribution law and starts the next iteration until convergence. [20] came a few years later than [27] and solves the same problem with a different approach. First, they present a new mathematical formulation of the problem, which eventually leads to a reformulation into a DC (Difference of Convex function) program [29] using an exact penalty technique. Then, they solve it using a DC algorithm (DCA) [29]. Numerical experiments are presented on a similar instance than in [27] and better results are obtained.

The optimum distribution of the search effort is much more complex for a moving target. We consider here the discrete-time problem, where the horizon of time is divided into periods. [6] presents a forward-backward algorithm to solve this problem. It has become the basis of many algorithms in the literature. Theoretically, with a single sensor, the necessary and sufficient conditions for an optimal search has been solved in [28]. It has latter been extended to double-layer constraints (*i.e.*, a resource constraint for the whole horizon and resource constraints for each period) in [16]. This problem, with a moving target, has also been tackled in [27], whereas the authors of [20] have extended their work to this case in [21]. In these works, the authors are searching for an allotment of sensors to zones for each period. Both works are adapting

their own methods to the forward-backward split introduced by [6]. The idea is to do many passes through the horizon of time, and at each pass to compute a resource allocation for each period. The algorithm finds the allocations while considering the target's location distribution obtained after all previous periods, and while considering the distribution expected of the next periods, computed in the previous passes. Every pass updates these distributions. When the number of passes is large enough, the algorithm converges to solutions that are close to the optimal one.

The work of [5] addresses a similar problem, with a set of cells and searches for a planning of the search effort ahead of the mission. However, the search plan produced (*i.e.*, the solution sought) is not a distribution of the resources in the cells. It is a planning of the position of the searchers over a discretized time. That kind of problem has similar characteristics to the UAV searches, where the positions of the searchers are used to compute the probability of detection. However, there is no need to compute the exact path of each individual searcher since they are considered indistinguishable and only the number of searchers in a cell during a time t is useful. Clearly, this assumption makes the problem easier to solve. We can consider this problem as a combination of the optimum distribution of a search effort problem, the UAV search problem, and the problem presented and solved in this paper.

3. Problem Definition

In this section, we introduce the problem definition using multiple subsections to introduce the different components. Table 1 summarizes all the notations that are used in this section. Then, we discuss the relation between the work of [20, 27] and the present problem. Finally, a last subsection introduces an example of application for this problem.

3.1. The area and its discretization

The searched area is denoted by E, it can have heterogeneous search characteristics. For example, some parts of the searched area have landscapes that are harder to search than others, or have sub-zones where the probability to find the target is null (a lake for instance). The area is discretized in such a way that it is divided in a set C of disjoint cells. All the search parameters are constant inside a cell, *i.e.*, cells are homogeneous. The partition is made such that:

$$E = \bigcup_{c \in C} c \quad \textit{and} \quad c_1 \cap c_2 = \emptyset \;\; \forall c_1, c_2 \in C$$

Each pair of cells $(c, c') \in C^2$ induces a traveling cost to join them, denoted by $t_{cc'}$. For simplification, the traveling costs are constant and are not dependent on the order in which the cells will be visited.

Constants	
E	Searched area
C	Set of cells
$t_{cc'}$	Traveling cost between c and c'
H	Time horizon
$T = \{1, \ldots, n\}$	Set of periods
Ω	Set of trajectories
$\alpha(\vec{\omega})$	Prior on the trajectory $\vec{\omega}$
$I = \{1, \ldots, m\}$	Set of sensors
Φ_i^t	Resources available at period t for sensor i
w_c^i	Coefficient that characterizes the efficiency of the search of sensor i inside cell c
$P_c^i(x)$	Conditional non-detection probability for sensor i in cell c depending on the resources x allocated
Indexes	
i	Sensor index
c	Cell index
t	Period index
d	Depot
$\vec{\omega}$	Trajectory index
Decision variables	
R_i^t	The route of sensor i during period t
c_p^{ti}	The p-th cell visited during period t by sensor i
φ_{ic}^t	Resources allocated to cell c at period t by sensor i
$y_{cc'}^{ti}$	Is set to 1 if sensor i at t is traveling from c to c'
h_{ic}^t	Is set to 1 if sensor i at t is visiting c
g_{ti}	Is set to 1 if sensor i is visiting any cell at period t
u_{ic}^t	Dummy variables used for sensor i to avoid sub-tours at t
q_{ti}	Resources used by sensor i for a travel leaving (and not entering) period t
r_{ti}	Resources used by sensor i for a travel entering period t

Table 1: Summary of the notations

3.2. The target

A target is moving through an horizon of time $[0, H]$. This horizon is divided into a set of n time periods $T = \{1, 2, \ldots, n\}$. In each period t, the target is hidden in one cell and we consider that it does not leave the cell during the period. When changing period, the target can either move to any other cell or stay in the same cell. To capture this, a set of trajectories, Ω, and their priors (*i.e.*, the probabilities that the target follows these trajectories) are given. A trajectory, $\vec{\omega} \in \Omega$, is a vector of n cells (one cell per period) with a probability (prior) $\alpha(\vec{\omega})$ that the target follows it. Hence the following equality holds:

$$\sum_{\vec{\omega} \in \Omega} \alpha(\vec{\omega}) = 1$$

Note that this implies that the target is necessarily in the searched space. $\vec{\omega}(t)$ is the cell where the target is at period t if it follows the trajectory $\vec{\omega}$. The trajectories are not constrained, *i.e.* the targets can move between far distanced cells between two successive periods. This makes the problem difficult, with a huge

number of possible trajectories ($|C|^n$). However, it is unlikely that all the trajectories are considered relevant and typical realistic instances will have targets moving to neighboring cells between two consecutive periods.

3.3. The sensor search

For the search mission, a set I of m mobile sensors is provided. This set is heterogeneous and the sensors may not have the same characteristics. A sensor $i \in I$ has for each period t a given amount Φ_i^t of available resource, typically time. Additionally, it has also a visibility coefficient in each cell c, denoted by w_c^i. This coefficient represents the efficiency (or reward) of the search effort put in the cell. Typically, a high visibility means that searching the cell is really efficient and thus the probability to miss the target, if it is in the cell, is quickly approaching zero when increasing the search effort inside it. By contrast, a low visibility sensor-cell means that an important effort (*e.g.*, time spent) inside the cell is needed from this sensor to approach the same value.

The sensor search activities are represented by φ_{ic}^t, the resources put by the sensor i in the cell c during the period t, and R_i^t the route visiting all the cells searched by i during period t. For a given t and i, $R_i^t = \{c_1^{ti}, \ldots, c_{|R_i^t|}^{ti}\}$ represents a route where i will move from cell c_p^{ti} to cell c_{p+1}^{ti} during the period $t, \forall p \in \{1, \ldots, |R_i^t| - 1\}$. It is also important to remember that the route of a sensor over the entire time horizon is continuous. Consequently, a sensor must move from the last cell searched in a period, to the next cell visited in another period. This trip has a cost that should be spread over several periods. Thus, a sensor i will also pass from $c_{|R_i^t|}^{ti}$ to $c_1^{t_2 i}$, $\forall t \in T, t < |T|$ where $t < t_2 \leq |T|$ and such that there is no cell searched for all $t_3 > t, t_3 < t_2$. These movements, between the last cell visited in a period, to the first cell visited in another period, are called inter-period movements. The remaining movements are intra-period ones. Note that in this problem, some moving costs may exceed the resources available during a period. Therefore, such moves may be spread on multiple periods, and the sensor will then travel without sensing on all periods except possibly the first and last ones.

3.4. The objective function

A target is undetected if it has not been detected at any period by any sensor of the search. The aim is to minimize the probability of such an event. We resort to the usual conditional non-detection probability [17, 27], that is a non-increasing and convex function. It represents the effectiveness of the search at the cell level, it is denoted by $P_c^i(\varphi_{ic}^t), \forall i \in I, c \in C, t \in T$ and is defined by:

$$P_c^i(\varphi_{ic}^t) = e^{-w_c^i \varphi_{ic}^t}$$

The sensors act independently at the cell level and on different periods. It means that the search effort put in a cell c by a sensor i at period t is not affecting nor affected by any other search activity. The value

returned by $P_c^i(\varphi_{ic}^t)$ is not depending on $\varphi_{i'c'}^{t'}, \forall i' \in I, \forall t' \in T, \forall c' \in C | (i,t,c) \neq (i',t',c')$. Since the sensor searches are independent, the probability to detect a target inside a cell c at time t is equal to:

$$\prod_{i \in I} P_c^i(\varphi_{ic}^t)$$

The search efforts along a trajectory are also independent, meaning that the effectiveness of a search effort during a period is not impacted by the efforts done on the other periods. Thus, for a given trajectory $\vec{\omega} \in \Omega$, the probability of non-detection of the target on this trajectory is equal to:

$$\prod_{i \in I} \prod_{t \in T} P_c^i(\varphi_{i\vec{\omega}(t)}^t)$$

The objective function, $f(X), X = \{\varphi_{ic}^t\}_{\forall t \in T, i \in I, c \in C}$, is the weighted sum for all trajectories of their probabilities of non-detection. Therefore,

$$\text{Minimize} \quad f(X) = \sum_{\vec{\omega} \in \Omega} \alpha(\vec{\omega}) \prod_{i \in I} \prod_{t \in T} P_c^i(\varphi_{i\vec{\omega}(t)}^t) \qquad (1)$$

It is a non-linear objective function. This objective function shows that using the notion of trajectory is important to produce solutions with a coherent search between periods. Clearly, searching the target along a trajectory in some periods makes this trajectory less interesting to be searched in the others. Indeed, the probability that the target is following the trajectory $\vec{\omega} \in \Omega$ is equal to the initial probability $\alpha(\vec{\omega})$ times the probability that the target has not been detected along this trajectory.

3.5. The mathematical model

The mathematical model of the problem is presented in this section. It has linear constraints only and a non-linear objective function. Many decision variables are needed to fully represent the routes and allocations of the sensors and every possible special case (*e.g.*, no movement, or only moving, during a period). For each sensor and each period, the route R_i^t is represented using similar elements as in the Traveling Salesman Problem [3] by adding a virtual node, the depot d, with null cost to travel to or from. The complexity of our problem and formulation is significant since for each sensor, its route has to be computed between the set of visited cells, that which itself needs to be determined.

The types of decision variables are the following:

- φ_{ic}^t the resources allocated to cell c by sensor i at period t as shown before. It is a continuous nonnegative variable.
- $y_{cc'}^{ti}$, a binary variable, equal to 1 if the sensor i travels from c to c' during the time period t, 0 otherwise.
- y_{cd}^{ti} (respectively y_{dc}^{ti}), a binary variable, equal to 1 if the cell c is the last cell (respectively, first cell) visited by i during period t, 0 otherwise.

- h_{ic}^t, a binary variable, equal to 1 if the cell c is visited by sensor i at period t, 0 otherwise.
- g_{ti}, a binary variable, equal to 1 if the sensor i visits at least one cell at period t, 0 otherwise.
- q_{ti}, a continuous nonnegative variable, the resource effort used during period t by sensor i for the travel that started in period t but finished in another.
- r_{ti}, a nonnegative continuous variable, the resource effort used during period t by sensor i for the travel that started in a period preceding t.
- u_{ic}^t, an integer dummy variable used for the sub-tour elimination constraints as in [22].

The model for the moving target problem is given in Model 1.

$$\text{Minimize } f(X) = \sum_{\vec{\omega}\in\Omega} \alpha(\vec{\omega}) \prod_{t\in T} \prod_{i\in I} P_c^i(\varphi^t_{i\vec{\omega}(t)}) \tag{1}$$

$$\sum_{c\in C} \varphi_{ic}^t + \sum_{c\in C}\sum_{c'\in C} y_{cc'}^{ti} t_{cc'} + q_{ti} + r_{ti} \leq \Phi_i^t \qquad \forall i \in I, \forall t \in T \tag{2}$$

$$q_{ti} + \sum_{t''=t+1}^{t''\leq t'} r_{t''i} \geq \qquad \forall (c,c') \in C^2, c \neq c', \forall i \in I,$$
$$t_{cc'}(1 - 2(1 - (\tfrac{y_{cd}^{ti} + y_{dc'}^{t'i}}{2})) - \sum_{t''=t+1}^{t''<t'} g_{t''i}) \qquad \forall (t,t') \in T^2, t' \geq t+1 \tag{3}$$

$$\sum_{c\in C} h_{ic}^t \leq |C| g_{ti} \qquad \forall i \in I, \forall t \in T \tag{4}$$

$$\varphi_{ic}^t \leq \Phi_i^t h_{ic}^t \qquad \forall c \in C, \forall t \in T, \forall i \in I \tag{5}$$

$$\sum_{c\in C} y_{cd}^{ti} = g_{ti} \qquad \forall i \in I, \forall t \in T \tag{6}$$

$$\sum_{c\in C} y_{dc}^{ti} = g_{ti} \qquad \forall i \in I, \forall t \in T \tag{7}$$

$$\sum_{c'\in C\cup\{d\}} y_{cc'}^{ti} = h_{ic}^t \qquad \forall c \in C, \forall i \in I, \forall t \in T \tag{8}$$

$$\sum_{c'\in C\cup\{d\}} y_{c'c}^{ti} = h_{ic}^t \qquad \forall c \in C, \forall i \in I, \forall t \in T \tag{9}$$

$$u_{ic}^t - u_{ic'}^t + y_{cc'}^{ti} |C| \leq |C| - 1 \qquad \forall (c,c') \in C^2, c \neq c', \forall i \in I, \forall t \in T \tag{10}$$

$$\varphi_{ic}^t \in \mathbb{R}^+ \qquad \forall c \in C, \forall i \in I, \forall t \in T \tag{11}$$

$$h_{ic}^t \in \{0,1\} \qquad \forall c \in C, \forall i \in I, \forall t \in T \tag{12}$$

$$y_{cc'}^{ti} \in \{0,1\} \qquad \forall (c,c') \in C^2, \forall i \in I, \forall t \in T \tag{12}$$

$$0 \leq u_{ic}^t \leq |C| \qquad \forall c \in C, \forall i \in I, \forall t \in T \tag{13}$$

$$q_{ti} \in \mathbb{R}^+ \qquad \forall i \in I, \forall t \in T | t < |T| \tag{14}$$

$$r_{ti} \in \mathbb{R}^+ \qquad \forall i \in I, \forall t \in T | t > 1 \tag{15}$$

$$g_{ti} \in \{0,1\} \qquad \forall i \in I, \forall t \in T \tag{16}$$

Model 1: Mathematical formulation of the problem

More details about the constraints:

- (1) is the objective function as introduced before.
- (2) limit the resources available in each period for each sensor.

- (3) are the constraints sharing the cost of a travel inter-periods between the concerned periods. When moving from a cell c to a cell c' starting at period t and finishing at period t', the cost ($t_{cc'}$) is shared between t, t' and all the empty periods in between (without any cell searched). The constraint is not always active (c is not the last cell visited, there are non-empty periods in between, etc.). Thus, an important part of these constraints is just used to deactivate them if the conditions are not met. *e.g.*, the right-hand side of one constraint is just equal to $t_{cc'}$ when active otherwise it is less or equal to 0. More precisely, the right side of the constraint contains the term $\frac{y_{cd}^{ti}+y_{dc'}^{t'i}}{2}$ which is equal to 1 if c is the last visited cell of period t and c' is the first visited cell of period t', otherwise it is lower than 1; $\sum_{t''=t+1}^{t''<t'} g_{t''i}$ which is equal to 0 if there is no cell searched in the periods between t and t', *i.e.*, the sensor starts its movement in period t and finishes it in period t', otherwise, it is equal to the number of periods with a search activity between t and t', so it is greater than or equal to 1. The left-hand side is $q_{ti} + \sum_{t''=t+1}^{t''\leq t'} r_{t''i}$ with q_{ti} the part of the movement done during period t. $\sum_{t''=t+1}^{t''\leq t'} r_{t''i}$ is the resources spent by the sensor i in the periods between t and t', t' included. Thus, with this constraint, the cost of the inter-period movement is distributed between the period when it starts (t with the variable q_{ti}), the period when it ends (t' with the variable $r_{t'i}$) and all the periods between these two ($t'', \forall t''$ such that $t+1 \leq t'' < t'$, represented with $\sum_{t''=t+1}^{t''<t'} r_{t''i}$).
- (4) are fixing g_{ti} to 1 if at least one cell is visited.
- (5) are fixing h_{ic}^{t} to 1 if resources are allocated.
- (6) & (7) state that only one cell is the last cell visited and only one the first visited.
- (8) & (9) are the flow constraints, *i.e.*, if the cell is visited, the sensor only enters and leaves it once, otherwise it does not enter nor leave it.
- (10) are the usual sub-tour elimination constraints as introduced in [22].
- All others constraints are domain constraints.

The non linear objective function of this problem is not the only aspect that makes the problem very challenging. Indeed, the large numbers of constraints, decision variables and the problem structure also contribute to the problem difficulty. To check this point, we introduce a variant of the problem, where the objective function is to minimize the maximum risk of not detecting the target over all its possible trajectories. This variant is stated in Model 2. Since its objective function can be linearized, it can be stated as a mixed integer linear program (see Model 3) whose constraints are the same as in the original problem. Hence, the variant and the original problem have the same set of feasible solutions. However, it will be shown in Section 7 that no optimal solution to Model 3 can be obtained in a reasonable amount

of time even for modest-size problem instances, which suggests that problem difficulty does not uniquely originates from the non-linear objective function. While this problem variant is also relevant, this paper focuses on the original problem version only.

$$
\begin{aligned}
&\text{Minimize } \alpha \\
&\alpha(\vec{\omega}) \prod_{t \in T} \prod_{i \in I} P_c^i(\varphi^t_{i\vec{\omega}(t)}) \leq \alpha \quad \forall \vec{\omega} \in \Omega \\
&\text{Subject to (2)-(16)} \\
&\alpha \in \mathbb{R}
\end{aligned}
$$

Model 2: Mathematical formulation of the problem variant

$$
\begin{aligned}
&\text{Minimize } \beta \\
&\ln(\alpha(\vec{\omega})) - \sum_{t \in T} \sum_{i \in I} \varphi^t_{i\vec{\omega}(t)} w^i_{\vec{\omega}(t)} \leq \beta \quad \forall \vec{\omega} \in \Omega \\
&\text{Subject to (2)-(16)} \\
&\beta \in \mathbb{R}
\end{aligned}
$$

Model 3: Linearized mathematical formulation of the problem variant

3.6. Multi-zone – a special case

In this section, we show how the problem treated in the previous works on multi-zone [20, 21, 27] can be seen as a particular case of the present problem.

As mentioned before, the multi-zone problem has each sensor allocated to a zone. Once allocated, a sensor can only search the cells associated to this zone with free moves between these cells. The rest of the problem (objective function, visibilities, etc.) is similar to the problem introduced in the paper (see [20, 21, 27]). The traveling costs are important to represent the zone allocations in our formulation. The costs between cells of different zones have to be infinite and thus prohibitive for any of these moves. Also, the movements between any two cells of the same zone are allowed, thus with null costs. With $z(c), \forall c \in C$, the function returning the zone associated to the cell c, we obtain:

$$
\forall (c, c') \in C^2, t_{cc'} = \begin{cases} +\infty & \text{if } z(c) \neq z(c') \\ 0 & \text{otherwise.} \end{cases}
$$

However, in the works on multi-zone, a sensor can be reallocated to a different zone when changing period. To represent this, we divide each sensor of the instance into n sensors. Let us call them sub-sensors. Each one of these sub-sensors is assigned to one period, and has a non-null amount of resources only in this period, *i.e.*, it is only doing its search during one period. Therefore, there is no traveling inter-periods, each sub-sensor is representing the search during one period of its corresponding sensor. Since the costs between two cells of different zones is still infinite, a sub-sensor is also allocated to one zone during its period. Hence, the search of one sensor is represented in each period by one sub-sensor, which is allocated to one zone.

These costs and divisions into sub-sensors make our problem a generalization of the previous works done on multi-zone addressed in [20, 21, 27].

4. Linear Approximation

In this section, we present the linear approximation of the objective function that will be used in the algorithm presented in Section 6, and in the lower bound computation of Section 5. The aim is to get rid of the exponential in the objective function and obtain a linear function that will lead to an easier problem to solve. The closer the linear approximation is, the better the algorithm precision will be. The initial idea is to use the approximation $e^x \approx 1 + x$ of the exponential function. This approximates well when x is close to 0, whereas it gets worst when x increases. A solution to this problem is to refine the approximation using given values, *e.g.*, $e^x \approx exp(y)(1 + (x - y))$ is a better approximation when the value of x is close to the constant y.

We split the decision variables $\varphi_{ic}^t, \forall i \in I, \forall c \in C, \forall t \in T$ in two terms: φ_{ic}^{at} and φ_{ic}^{bt}. The former is the resources already allocated to cell c by the sensor i at period t. It is a decision already made, so it is a constant. The latter is the decision that remains to be made, *i.e.*, how much resources should be added from i to c at t. Hence, $\varphi_{ic}^t = \varphi_{ic}^{at} + \varphi_{ic}^{bt}, \forall i \in I, \forall c \in C, \forall t \in T$. The new objective function, for a current solution with a given set of already made allocation $\vec{X} = \{\varphi_{ic}^{at}\}_{i \in I, t \in T, c \in C}$, is:

$$\text{Minimize} \quad \sum_{\vec{\omega} \in \Omega} \alpha(\vec{\omega}) \prod_{i \in I} \prod_{t \in T} \left(e^{-w^i_{\vec{\omega}(t)} \varphi^{at}_{i\vec{\omega}(t)}} \times e^{-w^i_{\vec{\omega}(t)} \varphi^{bt}_{i\vec{\omega}(t)}} \right)$$

or:

$$\text{Minimize} \quad \sum_{\vec{\omega} \in \Omega} \alpha(\vec{\omega}) \left(e^{\sum_{i \in I} \sum_{t \in T} -w^i_{\vec{\omega}(t)} \varphi^{at}_{i\vec{\omega}(t)}} \times e^{\sum_{i \in I} \sum_{t \in T} -w^i_{\vec{\omega}(t)} \varphi^{bt}_{i\vec{\omega}(t)}} \right)$$

If we apply the approximation:

$$\text{Minimize} \quad \sum_{\vec{\omega} \in \Omega} \alpha(\vec{\omega}) e^{\sum_{i \in I} \sum_{t \in T} -w^i_{\vec{\omega}(t)} \varphi^{at}_{i\vec{\omega}(t)}} (1 - \sum_{i \in I} \sum_{t \in T} w^i_{\vec{\omega}(t)} \varphi^{bt}_{i\vec{\omega}(t)})$$

thus:

$$\text{Minimize} \quad \text{K} - \sum_{\vec{\omega} \in \Omega} \sum_{i \in I} \sum_{t \in T} \alpha(\vec{\omega}) w^i_{\vec{\omega}(t)} e^{\sum_{i' \in I} \sum_{t' \in T} -w^{i'}_{\vec{\omega}(t')} \varphi^{at'}_{i'\vec{\omega}(t')}} \varphi^{bt}_{i\vec{\omega}(t)} \quad \text{(AP)}$$

with K a constant equal to $\sum_{\vec{\omega} \in \Omega} \alpha(\vec{\omega}) e^{\sum_{i \in I} \sum_{t \in T} -w^i_{\vec{\omega}(t)} \varphi^{at}_{i\vec{\omega}(t)}}$.

Undoubtedly, the greater the decision variables φ_{ic}^{bt} are, the further the approximation is from the objective function.

5. Lower bound

We now introduce a method to compute a lower bound to our problem. The idea is to solve a relaxed version of the problem, using the approximation (AP). The objective function of the relaxed problem is a piece-wise linear approximation with a bounded error. It uses many times the approximation (AP), applied to different points $\vec{X}$. The complexity of the problem is highly due to the computation of the routes. We propose to solve a relaxed version of the problem, without the routes but only a computation of a lower bound on the traveling costs. The idea is to consider that with x cells visited during a period, there are $x-1$ intra-period travels between them. Each travel is at least greater than the minimal distance between two cells, *i.e.*, $t_{min} = \min_{(c,c')\in C^2 | c\neq c'} t_{cc'}$.

The proposed relaxed problem is solved using Gurobi [23].

5.1. Relaxed problem model

The relaxed problem has the constraints (4), (6)-(10) removed. Constraints (2) are slightly modified into new constraints named (2'). The new objective function, is presented in more details later and is referred to as $g(X)$ for now. This objective function, for a given resource allocation, is always less than or equal to $f(X)$.

In the relaxed model, we add a new set of variables $k_{ti}, \forall t \in T, \forall i \in I$. It represents the number of intra-period travels needed for sensor i to visit all the cells searched at period t. It is equal to the number of cells visited minus one if there is at least one visited cell, zero otherwise. *i.e.*, $k_{ti} = max\left(0, \sum_{c\in C} h_{ic}^t - 1\right), \forall t \in T, \forall i \in I$. In the model, the k_{ti} are linearized by introducing constraints (17) and (18). The model of the relaxed problem is the Model 4.

$$
\begin{array}{lll}
\text{Minimize } g(X) & & (1') \\
\sum_{c\in C} \varphi_{ic}^t + k_{ti} t_{min} \leq \Phi_i^t & \forall i \in I, \forall t \in T & (2') \\
\varphi_{ic}^t \leq \Phi_i^t h_{ic}^t & \forall c \in C, \forall t \in T, \forall i \in I & (5) \\
\varphi_{ic}^t \in \mathbb{R}^+ & \forall c \in C, \forall i \in I, \forall t \in T & (11) \\
h_{ic}^t \in \{0,1\} & \forall c \in C, \forall i \in I, \forall t \in T & (12) \\
g_{ti} \in \{0,1\} & \forall i \in I, \forall t \in T & (16) \\
k_{ti} \geq (\sum_{c\in C} h_{ic}^t) - 1 & \forall t \in T, \forall i \in I & (17) \\
k_{ti} \in \mathbb{R}^+ & \forall i \in I, \forall t \in T & (18)
\end{array}
$$

Model 4: Mathematical formulation of the relaxed problem

Lemma: Model 4 is a relaxation of the original problem (model 1).

Proof: The constraints of the relaxed problem originate from the initial problem, except for (2'), (17) and (18). (17) and (18) are two types of constraints used to fix the newly introduced variables k_{ti} and do not enforce new constraints to the solution. (2') is a modification of (2). Thus, to prove that we have a

relaxed problem, we need to prove that the constraints (2) are stronger than (2'). It is true if the following inequality holds:

$$\sum_{c\in C}\varphi_{ic}^{t} + k_{ti}t_{min} \leq \sum_{c\in C}\varphi_{ic}^{t} + \sum_{c\in C}\sum_{c'\in C} y_{cc'}^{ti}t_{cc'} + q_{ti} + r_{ti}, \forall i \in I, \forall t \in T$$

$t_{min} \leq t_{cc'}, \forall c, c' \in C$ is straightforward. Then, we need to prove that:

$$k_{ti} \leq \sum_{c\in C}\sum_{c'\in C} y_{cc'}^{ti}, \forall i \in I, \forall t \in T$$

Considering now that:

$$k_{ti} = max\left(0, (\sum_{c\in C} h_{ic}^{t}) - 1\right), \forall i \in I, \forall t \in T$$

Hence, there are two cases for any given $i \in I, t \in T$. Either $k_{ti} = 0$, which is obviously satisfying the inequality. Either $k_{ti} > 0$, for which we need to prove that:

$$\sum_{c\in C} h_{ic}^{t} - 1 \leq \sum_{c\in C}\sum_{c'\in C} y_{cc'}^{ti}$$

Using the constraints (4) and (6), we know that, in this case, $\sum_{c\in C} y_{cd}^{ti} = 1$. Thus, we have:

$$\sum_{c\in C} h_{ic}^{t} - 1 \leq \sum_{c\in C}\sum_{c'\in C} y_{cc'}^{ti} - 1 + \sum_{c\in C} y_{cd}^{ti}$$

$$\sum_{c\in C} h_{ic}^{t} - 1 \leq \sum_{c\in C}\sum_{c'\in C\cup\{d\}} y_{cc'}^{ti} - 1$$

However, with constraints (8), we know that $\sum_{c'\in C\cup\{d\}} y_{cc'}^{ti} = h_{ic}^{t}$. Hence, the inequality is:

$$\sum_{c\in C}\sum_{c'\in C\cup\{d\}} y_{cc'}^{ti} - 1 \leq \sum_{c\in C}\sum_{c'\in C\cup\{d\}} y_{cc'}^{ti} - 1$$

and this proves our point about (2) and (2').□

5.2. The piece-wise linear approximation with bounded error

The objective function used in the relaxed model, $g(X)$, is a piece-wise linear approximation. Its error is ε-bounded, where ε is a strictly positive parameter, *i.e.*, the maximum gap between the values of $g(X)$ and $f(X)$ is ε. The lower ε is, the closer the approximation and the closer the optimal value of the relaxed model will be to the actual optimal value. However, small ε is also increasing the complexity (more pieces in the piece-wise linear function). Hence, a good trade-off between the quality and the complexity is required. The piece-wise linear objective function used in the relaxed problem is based on the approximation (AP) (presented in Section 4) applied to different points. Note that the approximation function is always lower than the actual objective function.

Let's consider, for each trajectory $\vec{\omega}$, the function $A_{\vec{\omega}}(x) = \alpha(\vec{\omega})e^{-x}$ on the interval $[0, L_{\vec{\omega}}]$ such that $L_{\vec{\omega}} = \sum_{i \in I} \sum_{t \in T} \Phi_i^t w^i_{\vec{\omega}(t)}$. The initial objective function can be formulated as $f(X) = \sum_{\vec{\omega} \in \Omega} A_{\vec{\omega}}(x)$. So if we find a ε-bounded approximation of $A_{\vec{\omega}}, \forall \vec{\omega} \in \Omega$, such that it is greater than $(1-\varepsilon)A_{\vec{\omega}}(x)$, then we obtain an ε-bounded approximation by summing these functions.

Let us now work on $A_{\vec{\omega}}$ for any given $\vec{\omega}$. We seek to obtain the piece-wise linear approximation of $A_{\vec{\omega}}$, called $A'_{\vec{\omega}}$. To obtain our ε-bounded error, we divide the interval $[0, L_{\vec{\omega}}]$ into sub-intervals starting at 0, where $A'_{\vec{\omega}}$ is linear in each sub-interval. The line in each sub-intervals joins two points of $(1-\varepsilon)A_{\vec{\omega}}$ and is tangent to $A_{\vec{\omega}}$. The last sub-interval may not consist of a tangent to $A_{\vec{\omega}}$, and all the sub-intervals (expect possibly the last one) are shown to have the same width.

Knowing x_0, the lower bound of the current sub-interval, the first step is to find the abscissa $x_1 \geq x_0$ where the line is tangent to $A_{\vec{\omega}}$. If x_1 is found to be larger than $L_{\vec{\omega}}$, then the sub-interval is $[x_0, L_{\vec{\omega}}]$ and the algorithm terminates. The second step is to determine the abscissa $x_2 \geq x_1$ such that the line intersects $(1-\varepsilon)A_{\vec{\omega}}$ in x_2. Again, if x_2 is found to be larger than $L_{\vec{\omega}}$, it is set to $L_{\vec{\omega}}$ and the algorithm terminates. These two steps are performed by solving an equation of the form $\alpha e^X + \beta X + \gamma = 0$ using the function W of Lambert [7].

We search for the equation of a line l, knowing that it crosses the point $(x_0, (1-\varepsilon)A_{\vec{\omega}}(x_0))$. l is tangent to $A_{\vec{\omega}}$ in $x = x_1$. Using (AP) and $l(x) = ax + b$, the slope is:

$$a = -\alpha(\vec{\omega})e^{-x_1}$$

and b is such that:

$$(1-\varepsilon)A_{\vec{\omega}}(x_0) = ax_0 + b$$

$$b = (1-\varepsilon)\alpha(\vec{\omega})e^{-x_0} - ax_0$$

We replace a with $-\alpha(\vec{\omega})e^{-x_1}$ and find

$$b = (1-\varepsilon)\alpha(\vec{\omega})e^{-x_0} + x_0\alpha(\vec{\omega})e^{-x_1}$$

Since the line l is tangent to $A_{\vec{\omega}}$ in x_1, it satisfies $A_{\vec{\omega}}(x_1) = ax_1 + b$, hence we have

$$\alpha(\vec{\omega})e^{-x_1} = -x_1\alpha(\vec{\omega})e^{-x_1} + (1-\varepsilon)\alpha(\vec{\omega})e^{-x_0} + x_0\alpha(\vec{\omega})e^{-x_1}$$

$$\Leftrightarrow 1 = -x_1 + (1-\varepsilon)e^{-x_0}e^{x_1} + x_0$$

$$\Leftrightarrow (1-\varepsilon)e^{-x_0}e^{x_1} - x_1 + x_0 - 1 = 0$$

By setting $X = x_1$, we have

$$(1-\varepsilon)e^{-x_0}e^X - X + x_0 - 1 = 0$$

So the equation can be written as $\alpha e^X + \beta X + \gamma = 0$ with

$$\alpha = (1-\varepsilon)e^{-x_0}$$
$$\beta = -1$$
$$\gamma = x_0 - 1$$

The number of solutions to this equation is determined by the discriminant $\Delta = \frac{\alpha}{\beta}e^{-\frac{\gamma}{\beta}} = -(1-\varepsilon)e^{-x_0}e^{x_0-1} = -\frac{1}{e}(1-\varepsilon)$. Since $\Delta \in (-\frac{1}{e}, 0)$, this equation has two solutions. We are interested in the largest one, as the smallest one is less than x_0, hence

$$x_1 = \left(-W_{-1}(\Delta) - \frac{\gamma}{\beta}\right) = \left(-W_{-1}(-\frac{1}{e}(1-\varepsilon)) + x_0 - 1\right)$$
$$\Leftrightarrow x_1 = x_0 - \left(W_{-1}(-\frac{1}{e}(1-\varepsilon)) + 1\right)$$

If $x_1 \geq L_{\vec{\omega}}$, then set $x_1 = L_{\vec{\omega}}$ and terminate.

Knowing x_1, we search for $x_2 \geq x_1$ such that the tangent to $A_{\vec{\omega}}$ in x_1 intersects the point $(x_2, (1-\varepsilon)A_{\vec{\omega}}(x_2))$. This tangent has equation $l(x) = ax + b$ with

$$a = -\alpha(\vec{\omega})e^{-x_1}$$

and

$$b = (1+x_1)\,\alpha(\vec{\omega})e^{-x_1}$$

This line also crosses the point $(x_2, (1-\varepsilon)A_{\vec{\omega}}(x_2))$, so

$$(1-\varepsilon)A_{\vec{\omega}}(x_2) = ax_2 + b = (1-\varepsilon)\alpha(\vec{\omega})e^{-x_2}$$

$$\Leftrightarrow -x_2e^{-x_1} + (1+x_1)e^{-x_1} = (1-\varepsilon)e^{-x_2}$$

We set $X = -x_2$, so

$$Xe^{-x_1} + (1+x_1)e^{-x_1} = (1-\varepsilon)e^{X}$$
$$\Leftrightarrow (1-\varepsilon)e^{X} - Xe^{-x_1} - (1+x_1)e^{-x_1} = 0$$
$$\Leftrightarrow \alpha e^{X} + \beta X + \gamma = 0$$

$$\alpha = (1-\varepsilon)$$
$$\beta = -e^{-x_1}$$
$$\gamma = -(1+x_1)e^{-x_1}$$

By the W Lambert function, the number of solutions to this equation is determined by the discriminant $\Delta = \frac{\alpha}{\beta}e^{-\frac{\gamma}{\beta}} = -(1-\varepsilon)e^{x_1}e^{\frac{-(1+x_1)e^{-x_1}}{e^{-x_1}}} = -(1-\varepsilon)e^{x_1}e^{-(1+x_1)} = -\frac{1}{e}(1-\varepsilon)$.

Since $\Delta \in (-\frac{1}{e}, 0)$, $\alpha e^X + \beta X + \gamma = 0$ has two solutions. The smallest one is x_0, the other one is $x_2 = \left(W_0(\Delta) + \frac{\gamma}{\beta}\right) = \left(W_0(-\frac{1}{e}(1-\varepsilon)) + (1+x_1)\right) = x_1 + \left(W_0(-\frac{1}{e}(1-\varepsilon)) + 1\right)$.

We have built a line on the interval $[x_0, x_2]$ that is tangent to $A_{\vec{\omega}}$ in x_1, with:

$$x_1 = x_0 - \left(W_{-1}(-\tfrac{1}{e}(1-\varepsilon)) + 1\right)$$
$$x_2 = x_1 + \left(W_0(-\tfrac{1}{e}(1-\varepsilon)) + 1\right)$$

Since $x_2 - x_0 = \left(W_0(-\frac{1}{e}(1-\varepsilon)) - W_{-1}(-\frac{1}{e}(1-\varepsilon))\right)$ only depends on ε, an ε-piece-wise linear lower bounding function $A'_{\vec{\omega}}$ of $A_{\vec{\omega}}$ on $[0, L_{\vec{\omega}}]$ can be built by splitting this interval into $p_g = \left\lceil \frac{L_{\vec{\omega}}}{x_2 - x_0} \right\rceil$ intervals that all have width $x_2 - x_0$ (expect possibly the last one, whose width is less than $x_2 - x_0$). Hence, for all $i \in \{1, \ldots, p_g\}$, we define $\chi_i = (i-1)(x_2 - x_0)$ as the lower bound of the i-th sub-interval of $[0, L_{\vec{\omega}}]$ and $\chi_{p_g+1} = L_{\vec{\omega}}$, so for all $i \in \{1, \ldots, p_g - 1\}$, the i-th piece of g is the line joining the points $(\chi_i, (1-\varepsilon)A_{\vec{\omega}}(\chi_i))$ and $(\chi_{i+1}, (1-\varepsilon)A_{\vec{\omega}}(\chi_{i+1}))$. The last piece of g is the line that joins the point $(\chi_{p_g}, (1-\varepsilon)A_{\vec{\omega}}(\chi_{p_g}))$ to the point $(L_{\vec{\omega}}, A_{\vec{\omega}}(L_{\vec{\omega}}))$ if $L_{\vec{\omega}} - \chi_{p_g} < \frac{x_2 - x_0}{2}$, and to the point $(L_{\vec{\omega}}, aL_{\vec{\omega}} + b)$ otherwise, where $y = ax + b$ is the equation of the tangent to $A_{\vec{\omega}}$ in $x_1 = \chi_{p_g} + \frac{x_2 - x_0}{2}$.

We know that for all $z \in (-\frac{1}{e}, 0)$, $ez \leq W_0(z) \leq z$, and $-1 \geq W_{-1}(z) \geq \frac{1}{z}$, so $\varepsilon \leq x_2 - x_0$.

So

$$\frac{1}{x_2 - x_0} \leq \frac{1}{\varepsilon} \Rightarrow p_g \leq \left\lceil \frac{L_{\vec{\omega}}}{\varepsilon} \right\rceil$$

Lemma 1 For all $A < B$, if there exists d, an ε-linear lower bounding function of $A_{\vec{\omega}}$ on $[A, B]$, then $d^{A,B}_{\min}$, the line that crosses the points $(A, (1-\varepsilon)A_{\vec{\omega}}(A))$ and $(B, (1-\varepsilon)A_{\vec{\omega}}(B))$ is also an ε-linear lower bounding function of $A_{\vec{\omega}}$ on $[A, B]$, and it satisfies $d^{A,B}_{\min}(x) \leq d(x)$ for all $x \in [A, B]$.

Proof of Lemma 1 If there exists d, an ε-linear lower bounding function of $A_{\vec{\omega}}$ on $[A, B]$, then $d(A) \geq (1-\varepsilon)A_{\vec{\omega}}(A)$, and $d(B) \geq (1-\varepsilon)A_{\vec{\omega}}(B)$, hence $d^{A,B}_{\min}(x) \leq d(x) \leq A_{\vec{\omega}}(x)$ for all $x \in [A, B]$. Moreover, by construction, the line $d^{A,B}_{\min}$ satisfies $(1-\varepsilon)A_{\vec{\omega}}(x) \leq d^{A,B}_{\min}(x)$ for all $x \in [A, B]$, so $d^{A,B}_{\min}$ is an ε-linear lower bounding function of $A_{\vec{\omega}}$ on $[A, B]$.□

Lemma 2 For all A and B such that $0 \leq A < B \leq L_{\vec{\omega}}$ with $B - A = x_2 - x_0$, $d^{A,B}_{\min}$ is the unique ε-linear lower bounding function of $A_{\vec{\omega}}$ over $[A, B]$.

Proof of Lemma 2 (by contradiction) Let $d \neq d^{A,B}_{\min}$ be an ε-linear lower bounding function of $A_{\vec{\omega}}$ over $[A, B]$. Then, $(1-\varepsilon)A_{\vec{\omega}}(A) < d(A)$ or $(1-\varepsilon)A_{\vec{\omega}}(B) < d(B)$, so for all $x \in (A, B)$, $d(x) > d^{A,B}_{\min}(x)$, in particular, $d(x_1) > d^{A,B}_{\min}(x_1) = A_{\vec{\omega}}(x_1)$, so d is not an ε-linear lower bounding function of $A_{\vec{\omega}}$ over $[A, B]$.□

Theorem 1 $p_{A'_{\vec{\omega}}}$, the number of pieces of $A'_{\vec{\omega}}$, the ε-piece-wise linear lower bounding function of $A_{\vec{\omega}}$ over $[0, L_{\vec{\omega}}]$, is minimum.

Proof of Theorem 1 (by contradiction) Let h be an ε-piece-wise linear lower bounding function of $A_{\vec{\omega}}$ over $[0, L_{\vec{\omega}}]$, with $p_h < p_{A'_{\vec{\omega}}}$. Then (by the pigeonhole principle), h has at least one piece which is an ε-linear lower bounding function of $A_{\vec{\omega}}$ over $[A, C]$, with $C - A > x_2 - x_0$. Let $B = A + x_2 - x_0 < C$. By Lemma 2, $d_{\min}^{A,B}$ is the unique ε-piece-wise linear lower bounding function of $A_{\vec{\omega}}$ over $[A, B]$. By construction, $d_{\min}^{A,B}(x) < (1-\varepsilon)A_{\vec{\omega}}(x)$, for all $x \in (B, C]$ so there is no ε-piece-wise linear lower bounding function of $A_{\vec{\omega}}$ over $[A, C]$.□

6. Algorithm

In this section, we present an algorithm to solve our problem. The complexity of a standard instance is such that using an exact method to produce optimal solutions is not possible in an acceptable amount of time. Our method is a greedy heuristic and it is building one solution, step by step, with heuristic estimation of the evolution of the objective function. To build the solution, the algorithm runs many times the three stages presented in this section, that allocate and compute the routes using greedy utilities. Once a solution is produced, a quick and easier version of the algorithm is run to improve the solution locally (without modifying the routes).

In this section, we first present the building of a solution, before introducing the components of the method in more detail.

6.1. The building of a solution

A solution is built by iterating many times on the following stages until no resource is available:

1. Compute the utilities of the decision variables
2. Take a step in a direction, *i.e.*, increase the resources allocated in one cell by one sensor at a period
3. Insert this cell in the route of the sensor if it is not already part of it

The algorithm starts with an empty solution, *i.e.*, without any resource allocated to any cell, nor route. This solution is evolving through these three stages, by greedily increasing the allocated resources, until no more resource is available for each sensor. The first stage is computing a greedy utility for each decision variables $\varphi_{ic}^{bt}, t \in, i \in I, c \in C$. Details on the utilities are presented in Section 6.2. In the second stage, the algorithm performs a step in one direction and increases a decision variable, *i.e.*, it allocates more resources in a cell c at a period t from a sensor i. The way to select the decision variable and to determine the amount of added resources is presented in Section 6.3. The last stage (Section 6.4) adds the cell c in the route of sensor i at period t, with c, i, t corresponding to the decision variable, picked for a step, during the previous stage.

6.2. Utility

The utility is an important part of our algorithm. It has to reflect, for each decision variable φ_{ic}^{bt}, its immediate impact in the objective function but also consider the impact on the available resources. Indeed, allocating resources to a cell c from a sensor i during a period t, means that c has to be added to the route of i at t if it is not already part of it. However, the actual impact of a step is difficult to assess (Section 6.3).

The approximation of the objective function, (AP) presented in Section 4, has the following shape:

$$\text{Minimize} \quad -\sum_{\vec{\omega}\in\Omega}\sum_{i\in I}\sum_{t\in T} D_{i\vec{\omega}}^{t}\varphi_{i\vec{\omega}(t)}^{bt}$$

Where $D_{i\vec{\omega}}^{t} = \alpha(\vec{\omega})w_{\vec{\omega}(t)}^{i}e^{\sum_{i'\in I}\sum_{t'\in T} -w_{\vec{\omega}(t')}^{i'}\varphi_{i\vec{\omega}(t)}^{at}}$, is a constant. It can be reformulated as:

$$\text{Minimize} \quad -\sum_{c\in C}\sum_{i\in I}\sum_{t\in T}\left(\varphi_{i\vec{\omega}(t)}^{bt}\sum_{\vec{\omega}\in\Omega|\vec{\omega}(t)=c} D_{i\vec{\omega}}^{t}\right)$$

This approximation of the objective function is re-computed at each iteration of the algorithm, considering therefore the actual resources allocated of the current solution after the last step. In such a way, the approximation and utilities will fit as best as possible the actual objective function. When computing the utilities, the $D_{i\vec{\omega}}^{t}, \forall t \in T, i \in I, \vec{\omega} \in \Omega$ are constant. Thus, a straightforward utility, without considering the traveling costs, is clearly to use:

$$(U_1^{tic}) = \sum_{\vec{\omega}\in\Omega|\vec{\omega}(t)=c} D_{i\vec{\omega}}^{t}, \forall i \in I, \forall t \in T, \forall c \in C$$

This quantity is the sum of $D_{i\vec{\omega}}^{t}$, for all trajectories such that the target is in c at t. It can naturally serve as a utility since it evaluates the direct impact of every decision variables on the linear approximation (*i.e.*, the steep of its derivative). The greater it is, the better.

Now, we need to penalize the U_1^{tic} in order to take into account the traveling costs. We propose to multiply this utility by the difference between the available resources and a heuristic estimation of the adding cost, *i.e.*, $\rho_i^t - \text{add}_i^t(c)$ where $\text{add}_i^t(c)$ returns an estimation of the cost induced by adding c in the route of i at period t. ρ_i^t is the remaining resources at period t for i. We obtain:

$$U_2^{tic} = \left(\sum_{\vec{\omega}\in\Omega|\vec{\omega}(t)=c} D_{i\vec{\omega}}^{t}\right) \times \left(\rho_i^t - \text{add}_i^t(c)\right), \forall i \in I, \forall t \in T, \forall c \in C$$

The estimation of an extra cost has to be larger than the actual extra cost obtained in Section 6.4. In our algorithm, these estimations are obtained with a best-insertion method. More details are provided in Section 6.4.

This utility will be negative only when the estimation of the extra cost is greater than the available resources. U_2^{tic} is also an estimation of the impact on the current linear approximation if all the resources of i available at t are put in c.

6.3. Setting the step

In this sub-section, we discuss the setting of the step in the process of building a solution. A step corresponds to the increase of the resources already allocated in a cell c, by a sensor i at period t. The triplet picked $(c, i, t), c \in C, i \in I, t \in T$, corresponds to a decision variable φ_{ic}^{bt}. It means that the current solution is modified in such a way that $\varphi_{ic}^{at} \leftarrow \varphi_{ic}^{at} + \varphi_{ic}^{bt}$, where φ_{ic}^{bt} is equal to the minimum between the remaining resources available and the amount z of the step.

In our algorithm, the triplet (c, i, t) is picked randomly between all the triplets with a utility U_2^{tic} equal to the best utility, and greater than 0. In other words, we pick randomly one of the triplet $(c, i, t), c \in C, i \in I, t \in T$ that satisfies:

$$U_2^{tic} = \left(\max_{t' \in T, i' \in I, c' \in C} U_2^{t'i'c'} \right)$$

$$U_2^{tic} > 0$$

If no utility can be selected, *i.e.*, when $\max_{t \in T, i \in I, c \in C} U_2^{tic} \leq 0$, the algorithm stops. This corresponds to a solution where all the sensors have all their resources used in all periods.

The size of the steps, z, is an important parameter. A large step is putting a lot of resources in the cell, and this is a decision that our algorithm will never revoke. However, this decision is based on a greedy utility of the approximation function. When the allocated resources is increasing, the utilities are changing and the decision may no longer be appropriate. Indeed, we have seen in Section 4 that when the decision variables are too large, the linear approximation is far from the real objective function. Thus, large steps are not recommended since this leads to allocating too much resources in a cell at a time, which may be detrimental to the solution quality.

On the other hand, small steps are offering much more careful and precise decisions. Since the approximation and the utilities are updated after each step, each decision is taken considering more information. However, it affects much more the computation time since it means more iterations for the algorithm to fully allocate all the available resources.

In our algorithm, the step size z is a parameter. It is recommended to keep z low compared to the available resources, while controlling the number of iterations. By default, we fix the step size to 1% of the minimum of the available resources (*i.e.*, $z = 0.01 \times \min_{t \in T, i \in I} \Phi_i^t$).

6.4. Addition of cells to the route

In this subsection, we present the last stage of each iteration of the algorithm. In the previous stage, a triplet (c, i, t) was selected for a step. Now, we need to insure that the cell c is in the route of sensor i during period t.

A best insertion method is used. The new cell c is inserted directly in the route of sensor i at period t where its insertion minimizes the additional traveling cost: if c shall be inserted in the route between a and b, then we replace arc (a, b) with (a, c) and (c, b). This induces an insertion cost of $t_{ac} + t_{cb} - t_{ab}$. This cost is removed from the resources available in all the impacted periods. Hence, the insertion cost is removed for all periods where the sensor is traveling along (a, c) and (c, b). For intra-period travels, it only removes resources in this period. For inter-period travels, it removes resources in all the corresponding periods. Such a method produces a final solution where all the periods impacted by some inter-periods moves have enough spare resources to endorse the additional cost. A last reparation algorithm then dispatches the extra-spared resources between the concerned periods.

6.5. The local optimization

This subsection presents the method run to locally optimize a solution produced by the previous algorithm. It is a local optimization since the routes computed are not modified, and only a reallocation of the resources is performed. In this method, the reallocation is done for each pair of periods $(t_1, t_2), \forall (t_1, t_2) \in T^2$ with a common inter-period move starting at t_1 and finishing at t_2. All the resources already allocated between these two periods are reallocated (*i.e.*, $\sum_{i \in I, c \in C} \varphi_{ic}^{t_1} + \varphi_{ic}^{t_2}$). In addition, the cost of the inter-periods moves is also reallocated (since this cost has been spared twice, once in each period). The reallocation method consists in:

1. Computing the utilities $U_1^{tic} \times \rho_i'^t$ of the decision variables
2. Performing a step in the direction of the best utility greater than 0

This is a light version of the previous algorithm, without any consideration to traveling costs since the routes are already fixed. It re-affects the available resources, while satisfying resources limitation and maintaining feasibility of existing routes.

7. Numerical Experiments

In this section, we study the efficiency of our method, in terms of solution quality and computational effort. We designed several experiments, in order to test the impact of many parameters. These important parameters are the numbers of cells, sensors, trajectories and periods, which are all expected to increase the running time of our algorithm but with different impact on objective values. We are not considering a comparison with the results of the previous papers [21, 27]. Indeed, as stated before, their problem is a

special case of ours and the complexity is much less. For example, once a sensor is allotted to a zone, their method only needs to allocate the resources between the cells. Whereas, our method also computes a route between the allocated cells, even if the route always has a null cost. In the case of a mobile target, in order to generalize their instances, we also need to multiply the number of sensors by the number of periods, hence producing much more complex problems to solve. In the instance used in [21], we would need 24 sensors instead of 6 to represent the same instance. Therefore, we consider that a comparison between their results and our method would be meaningless, as the addressed problems differ a lot.

All CPU running times are expressed in seconds. All values are average values between the corresponding set of instances. The software is coded in C++ and all the experiments were run on a computer with Ubuntu 16.04 and Intel Core i7-6700HQ CPU @ 2.60GHz $\times$ 8 cores and 16 GBytes of RAM.

7.1. Description of the instances generator

In order to test our method on a large set of instances, we designed an instance generator. The generator takes different parameters as inputs and creates a discretized instance as output. The area E is represented as several rows of cells, each row having the same number of cells. It takes the form of a grid as presented in the example of Figure A.1. This is obtained by the first two parameters: n_r is the number of rows and n_c is the number of columns. Thus, the number of cells is equal to $n_c \times n_r$. The traveling cost between two cells is the Euclidean distance such that the two farthest cells have a distance equal to $\min_{t \in T, i \in I} \Phi_i^t$. We consider such costs to impact the solutions without restricting the sensors to only a few cells. The numbers of sensors m and periods n are also parameters. For each sensor, a random visibility between 0 and 1 is assigned to each cell. The quantity of available resources for each period, for each sensor, is equal to Φ, a parameter.

The generation of the trajectories is an important part of our generator. First, we want to be able to generate a given number of trajectories. It will allow our experiments to freely study other parameters without any concern on the impact of the number of trajectories. Second, we want to generate realistic trajectories, as in [21, 27], that represents an uncertain movement in a direction between each period. For that purpose, we proposed two ways of generating the trajectories. The first way has a fix number of trajectories, set to 500. These trajectories are computed by first picking a random starting cell, and then picking random transitions between each period, following the probabilities of Figure 1. If one transition is not possible (*e.g.*, the target is in a corner), its probability is equally distributed between the other transitions with non null probability, *i.e.*, we keep an equal distribution of the probability between the possible transitions. The probabilities of the trajectories are uniformly randomly picked such that their sum is equal to 1. The second way to generate trajectories is done by selecting v random starting cells. v is a parameter. All the possible trajectories starting from these cells, following the transition probabilities of Figure 1, are generated. *e.g.*, with two periods from one cell, there are four trajectories with equal

probabilities, one move to the top, one to the left, one to the right and one to the bottom cell. We do not control the number of generated trajectories since there are less possible transitions when the cell is adjacent to the limit of the search space.

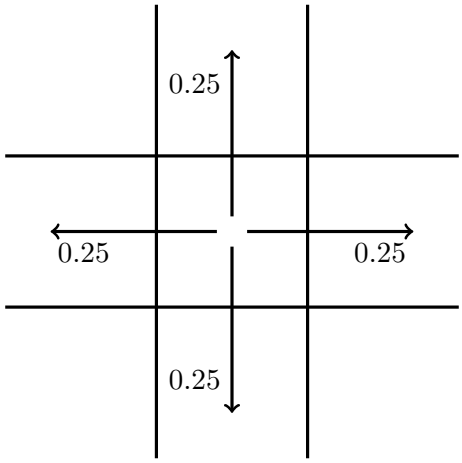

Figure 1: Transition probabilities

The default parameters are given in Table 2.

$n_r \times n_c$	the number of cells	12×10
m	the number of sensors	3
n	the number of periods	4
Φ	the amount of resources per period	5

Table 2: Default values used to generate an instance

7.2. Experiment 1 – Cells

In this experiment, we measure the impact of the number of cells. With more cells, the division of the area is more precise and so is the search. However, adding more cells makes the problem more difficult to solve. The instances of this experiment are generated with all the default parameters and only the number of cells is varying. The considered numbers of cells $|C|$ are $\{5 \times 5, 8 \times 8, 10 \times 10, 15 \times 15, 20 \times 20\}$. This set covers a wide variety of instances with simple (5×5 and 8×8), medium (10×10 and 15×15) and hard (20×20) instances. For each number of tested cells, a set of one hundred instances is generated. The small instances have a high search density, *i.e.* a few cells for a lot of possible trajectories. With more cells, the trajectories have much less parts in common. It means that there are less associations cells-periods that are common to multiple trajectories, where the search effort is much more efficient. Hence, it increases the number of decision variables and the number of utilities to compute. Increasing the number of cells is therefore expected to increase the number of utilities to compute, thus to impact greatly the running time. The average results are presented in Table 3.

These results show the importance of the number of cells in the CPU time. Indeed, the solution time increases quickly, especially for large instances. However, even for these large instances, with up to 400

#cells	Probability of non-detection	CPU time (s)
5×5	0.2061	0.20
8×8	0.4699	0.39
10×10	0.5699	0.73
15×15	0.6962	2.83
20×20	0.7534	7.31

Table 3: Objective value and CPU time when the number of cells varies

cells, the solution time remains acceptable. It shows that our method scales well with the number of cells. Nonetheless, it stays irremediably an important parameter on the solving time, that should not be neglected when discretizing the instances. *i.e.*, when dividing the search area E into cells, the number of cells has to consider the budget of CPU time allowed.

Since the trajectories have the same size, but use more cells to represent the searched area, we can deduce that the common parts between trajectories explain why the objective value is worsening a lot when the number of cells increases. However, those large instances may better represent actual search contexts.

To conclude, our method is scaling well to high numbers of cells, which is an important point as this allows to address realistic problems.

7.3. Experiment 2 – Sensors

This second experiment focuses on the number of sensors. It is expected that more sensors is improving the value of the objective but obviously slowing the solution process. An initial set of one hundred instances with 6 sensors is generated and solved. Afterwards, we remove a sensor and solve again, and iterate in such a way that each number of sensors $m \in \{1, 2, 3, 4, 5, 6\}$ is tested for each instance. Thus, the trajectories, priors and cells are the same for a given instance through each number of tested sensors. This allows the results obtained to show only the impact of the number of sensors. The averaged results are presented in Table 4.

#sensors	Probability of non-detection	CPU time (s)
1	0.8427	0.11
2	0.7099	0.43
3	0.5979	0.99
4	0.5028	1.78
5	0.4219	2.81
6	0.3531	4.10

Table 4: Objective value and CPU time when the number of sensors varies

These results show that, as expected, the problem is more and more difficult to solve but provides better objective values when more sensors are used. First, the CPU time increases continuously along with the number of sensors. More precisely, there is an increase of the solution time whenever a sensor is added to the problem. The increase gets larger when there are more sensors (*e.g.*, an increase of ≈ 0.32s between 1

and 2 sensors, and $\approx$ 1.29s between 5 and 6). However, considering the values as percentage of the CPU time, the increase is lower with more sensors (*e.g.*, an increase of $\approx$ 397% of the CPU time between 1 and 2 sensors, and $\approx$ 146% between 5 and 6). This behavior is observed for all numbers of sensors. On that part, we conclude that adding one sensor to an instance increases the solution time, with an increase depending on the number of sensors (in our results, $m-1$ times a coefficient close to 0.5). The solution time with few sensors is very modest, and is acceptable with many.

Secondly, the objective value is obviously decreasing (hence improving) when more sensors are used. In this set of instances, the decrease seems to be continuous but non linear. Indeed, whenever a sensor is added, the objective value is approximately dropping by 16%.

This experiment shows the impact of the number of sensors, making the problem more complex to solve but providing a better search. However, our solution is still scaling well with the increase of sensors, with more than acceptable CPU times when six sensors are used.

7.4. Experiment 3 – Periods

In this experiment, we focus on the number of periods. As in the previous experiments, we want to test our algorithm on the number of periods varying $n \in \{1, 2, 3, 4, 5, 6\}$. We generate a set of one hundred instances for each value of n, with the other parameters fixed to their default values except for the available resources. Indeed, we aim at measuring the impact on the objective function of more precise trajectories, *i.e.*, more periods used to represent the target moves. Thus, we need to make sure that the overall search budget for each sensor remains the same. With the values of $n \in \{1, 2, 3, 4, 5, 6\}$, the amount of resources in each period for all the sensors is respectively equal to $\{6, 3, 2, 1.5, 1.2, 1\}$. The averaged results are presented in Table 5.

#periods	Probability of non-detection	CPU time (s)
1	0.8605	0.02
2	0.8516	0.14
3	0.8474	0.42
4	0.8452	0.97
5	0.8443	1.86
6	0.8462	3.30

Table 5: Objective value and CPU time when the number of periods varies

These results show that the number of periods has an impact on the solution time. With low value, it is solved very fast, especially in the static case, *i.e.*, with a single period. Whereas, with a lot of periods, it requires more time, with six periods needing 146 times more CPU time than one period, which is a huge ratio. The evolution of the CPU time is similar to Section 7.3 (bigger time increase in seconds with more periods but smaller percentage). These results show that the number of periods has an important impact on the solution time and really marks the difference between easy instances (one or two periods) and medium

to hard instances.

On the objective value side, there are small decreases along with the increase of the number of periods, except for six periods. It is due to the fact that with more periods, there is a higher probability of trajectory intersections, *i.e.*, trajectories are more likely to have common cells. Thus, making a search effort in these cells is more efficient.

To conclude, the number of periods has only a slight effect on the value of the objective value, however it impacts negatively the solution time. Our method is scaling pretty well and has low computational times even when considering six periods.

7.5. Experiment 4 – Trajectories

In this last experiment, we study the impact of the number of trajectories. [27] consider that this number has a significant impact on the combinatorial complexity of their problem, which is a special case of ours. We can easily expect the same impact on the combinatorial complexity of our problem. Here, we aim to measure its impact on the solution time of our method and on solution quality. For that purpose, we consider two sets of instances, each one with a different process to generate the trajectories (see Subsection 7.1). The first set has all the possible trajectories (following the transition probabilities) made from a given number of starting cells. This produces realistic cases, with trajectories that have a lot of similarities. The parameter for this generation is the number of starting cells. The second set has the target trajectories drawn similarly to the previous experiments, where each trajectory is obtained by picking a random starting cell and a random transition between each period. It takes the number of trajectories to draw as a parameter. These instances are likely to have disjoint trajectories, with just a few similarities between them. In order to keep around the same number of trajectories, the first set of instances is generated first, with one hundred instances for each number of starting cells in $\{5, 10, 20, 30, 40, 50\}$. Afterwards, the second set is generated using the average number of trajectories rounded as parameter on each set of a hundred instances. The results on the first set are reported in Table 6 and the second set in Table 7.

#starting cells	#trajectories	Probability of non-detection	CPU time (s)
5	256.25	0.0313	0.48
10	498.07	0.1290	0.73
20	954.94	0.2483	1.01
30	1385.12	0.3058	1.18
40	1793.81	0.3274	1.30
50	2166.32	0.3285	1.39

Table 6: Objective value and CPU time for the first set of instances when the number of starting cells varies

This experiment shows that the number of trajectories has a relatively modest impact on the running time of our method. Indeed, in both sets, the running time increases, but it remains small and even with the highest number of trajectories, the running time is under 1.5 seconds. One trajectory is not necessarily

#trajectories	Probability of non-detection	CPU time (s)
256	0.5235	0.88
498	0.5968	0.99
955	0.6457	1.10
1385	0.6651	1.21
1794	0.6774	1.31
2166	0.6847	1.40

Table 7: Objective value and CPU time for the second set of instances when the number of starting cells varies

adding decision variables thus not adding more utilities to compute. That is why our method is not impacted a lot by the number of trajectories, since there are already a lot of trajectories compared to the size of the searched area. Clearly, the running time of the first set is the most impacted. Indeed, the generation of the trajectories in this set is such that it adds a lot more decision variables than in the second set.

The evolution of the objective function shows that, as expected with more trajectories, the objective function decreases. Also, we can see with Table 6 compared to Table 7, that the non-detection probability is easier to minimize when the trajectories have a lot of similar elements. However, it is easier for our method to solve instances where the trajectories are all randomly generated, thus having less parts in common.

To conclude, the number of trajectories does not impact a lot the solution time of our method, but it impacts negatively solution quality.

7.6. Overall results

We report in this last subsection an overall result on all our instances. It is the average gap with the lower bound. For each generated instance, we compute the lower bound with an epsilon error as a parameter equal to 0.001. The average gap (*i.e.*, $\frac{f}{\ell} - 1$ with f the result of our algorithm and ℓ the lower bound) is equal to 15% of the lower bound. The only exception is for the first set of trajectories, where the lower bounds are very close to 0. Considering that the traveling costs have an impact on the solution (just enough energy to travel between the two more extreme cells), the lower bound is expected to be loose. Hence, we conclude from that average gap with the lower bound on this many instances, that our algorithm has acceptable solutions.

Finally, we try to solve the instances of our experiments using the variant problem introduced in Section 3.5. This problem has a different objective function, but the same constraints as the original problem, plus one new constraint to linearize the objective function. Therefore, a solution obtained with this problem variant is feasible for the original problem. We have used Gurobi to solve Model 3 on the easiest instances from each of our previous experiments: instances with 5×5 cells from subsection 7.2, instances with one sensor from subsection 7.3, etc. We set a time limit of 14 hours for Gurobi to return a solution, but it turns out that none of these instances could be solved to optimality. Additionally, the best solution returned by the solver when time limit is reached is never a good solution for our problem (*i.e.*, when the solution is

applied to our problem, its objective function is worst that the one found heuristically by our algorithm). Therefore, the two problems are distinct and solving the variant problem is both hard and does not help to produce good solution for our main problem. We conclude that the complexity of our problem is not only related to the non-linear objective function but is also highly depending on the constraints and decision variables. The production of an optimal solution using a solver is not possible in an acceptable running time of several hours. On a side note, using a solver like Gurobi on the variant problem with a given budget time is also not returning an interesting solution, that can be exploited for our problem.

8. Conclusions

In this paper, we proposed a new formulation of the problem of planning a multisensor search for a moving target. We generalized the previous works on the literature by adding traveling costs for the sensors between the searched cells. This generalization makes the problem much more difficult to solve since the route of each sensor has to be determined, between a set of searched cells to select. We presented the mathematical formulation of the problem, and a method to compute an efficient lower bound. Afterwards, we presented a novel algorithm to solve our problem that is not relying on a forward-and-backward method as in previous works. We analyzed the efficiency of our algorithm with thousands of instances, and concluded that the running time is scaling well with various parameters. Many extension can be considered for this problem. One is to deal with multiple targets, where the problem could be even more generalized but needs many considerations especially on the objective function and the priors.

References

[1] Alfeo, A. L., Cimino, M. G., and Vaglini, G. (2019). Enhancing biologically inspired swarm behavior: Metaheuristics to foster the optimization of uavs coordination in target search. *Computers & Operations Research*, 110:34–47.

[2] Alpern, S. (2019). Search for an immobile hider in a known subset of a network. *Theoretical Computer Science*, 794:20–26.

[3] Bellmore, M. and Nemhauser, G. L. (1968). The traveling salesman problem: a survey. *Operations Research*, 16(3):538–558.

[4] Benkoski, S. J., Monticino, M. G., and Weisinger, J. R. (1991). A survey of the search theory literature. *Naval Research Logistics (NRL)*, 38(4):469–494.

[5] Bourque, F.-A. (2019). Solving the moving target search problem using indistinguishable searchers. *European Journal of Operational Research*, 275(1):45–52.

[6] Brown, S. S. (1980). Optimal search for a moving target in discrete time and space. *Operations Research*, 28(6):1275–1289.

[7] Corless, R. M., Gonnet, G. H., Hare, D. E., Jeffrey, D. J., and Knuth, D. E. (1996). On the lambertw function. *Advances in Computational mathematics*, 5(1):329–359.

[8] Curtis, J. and Murphey, R. (2003). Simultaneous area search and task assignment for a team of cooperative agents. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 5584.

[9] De Guenin, J. (1961). Optimum distribution of effort: An extension of the koopman basic theory. *Operations Research*, 9(1):1–7.

[10] Dobbie, J. M. (1968). A survey of search theory. *Operations Research*, 16(3):525–537.

[11] Gal, S. (1979). Search games with mobile and immobile hider. *SIAM Journal on Control and Optimization*, 17(1):99–122.

[12] Gal, S. (2010). Search games. *Wiley Encyclopedia of Operations Research and Management Science*.

[13] Garrec, T. and Scarsini, M. (2020). Search for an immobile hider on a stochastic network. *European Journal of Operational Research*, 283(2):783–794.

[14] George, J., Sujit, P., and Sousa, J. B. (2011). Search strategies for multiple uav search and destroy missions. *Journal of Intelligent & Robotic Systems*, 61(1-4):355–367.

[15] Hohzaki, R. (2016). Search games: Literature and survey. *Journal of the Operations Research Society of Japan*, 59(1):1–34.

[16] Hohzaki, R. and Iida, K. (2000). A concave maximization problem with double layers of constraints on the total amount of resources. *Journal of the Operations Research Society of Japan*, 43(1):109–127.

[17] Koopman, B. O. (1980). *Search and screening: general principles with historical applications*. Pergamon Press.

[18] Kuyucu, T., Tanev, I., and Shimohara, K. (2015). Superadditive effect of multi-robot coordination in the exploration of unknown environments via stigmergy. *Neurocomputing*, 148:83–90.

[19] Lanillos, P., Gan, S. K., Besada-Portas, E., Pajares, G., and Sukkarieh, S. (2014). Multi-uav target search using decentralized gradient-based negotiation with expected observation. *Information Sciences*, 282:92–110.

[20] Le Thi, H. A., Nguyen, D. M., and Dinh, T. P. (2014). A dc programming approach for planning a multisensor multizone search for a target. *Computers & Operations Research*, 41:231–239.

[21] Le Thi, H. A., Nguyen, D. M., and Dinh, T. P. (2015). A based-dc programming approach for planning a multisensor multizone search for a moving target. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 107–118. Springer.

[22] Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.

[23] Optimization, G. (2014). Inc.,"gurobi optimizer reference manual," 2015.

[24] Otto, A., Agatz, N., Campbell, J., Golden, B., and Pesch, E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey. *Networks*, 72(4):411–458.

[25] Rubinstein, R. Y. and Kroese, D. P. (2013). *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media.

[26] Senanayake, M., Senthooran, I., Barca, J. C., Chung, H., Kamruzzaman, J., and Murshed, M. (2016). Search and tracking algorithms for swarms of robots: A survey. *Robotics and Autonomous Systems*, 75:422–434.

[27] Simonin, C., Le Cadre, J.-P., and Dambreville, F. (2009). A hierarchical approach for planning a multisensor multizone search for a moving target. *Computers & Operations Research*, 36(7):2179–2192.

[28] Stone, L. D. (1979). Necessary and sufficient conditions for optimal search plans for moving targets. *Mathematics of Operations Research*, 4(4):431–440.

[29] Tao, P. D. and An, L. T. H. (1997). Convex analysis approach to dc programming: theory, algorithms and applications. *Acta mathematica vietnamica*, 22(1):289–355.

[30] Von Neumann, J., Morgenstern, O., and Kuhn, H. W. (2007). *Theory of games and economic behavior (commemorative edition)*. Princeton university press.

Appendix A. Application example

In this appendix, we introduce a problem example for the sake of illustration, taken from naval applications [17]. In the present case, the goal is to find an object lost in the sea. For example, a plane has lost contact with the control tower while flying over the sea, and is presumed to have crashed. The objective is to locate the wreck in a given time limit, before it is too late to rescue potential survivors or before an item, like the black box of the plane, is definitively considered lost. Many trajectories, alongside their probabilities, are drawn from the known sea currents and the different possible locations where the plane may have crashed. As mentioned earlier, the search is done over a given time horizon (*e.g.*, four days to find the target) and thus the trajectories can clearly be discretized into periods (*e.g.*, periods of 12 hours). In such an application, the searchers (*i.e.*, the sensors) can be of different types. In this example, we consider that the searchers are boats, equipped for a week of search. The search carried out by a boat is performed by deploying a device in the sea to search underwater (*e.g.*, a small submarine or a sonar). The boat is stationary while the device is searching around it and thus, we clearly see the discretization of the search into cells. Therefore, the entire problem is discretized into cells. In each searched cell, a boat stops and deploys its sensing device to search for the target. The different visibilities may depend on the various devices used and the topology and weather conditions in the searched zone.

A visual illustration is given in Figure A.1. The mission is 4 days long, discretized into periods of 12 hours, and the searched area is decomposed into square cells that form a 2D-mesh. The two possible locations of the crash site are the red dots and they are both given a probability. They represent the possible locations of the target at period 1. The possible transitions (target movement) between time periods are illustrated with the black arrows centered on one of the initial possible location, and each transition has a given probability, *i.e.* between each period, the sea currents are moving the target in one of the adjacent cell in the south, south-east or east. All the possible trajectories can therefore be deduced from the initial positions, the transitions, the number of periods and the given probabilities. A boat is used to locate the target. It is not available during the first two time periods (the limits on the resources of these periods are equal to 0) to model the amount of time required to reach the searched area. A possible search plan for the boat is to search four cells represented in gray. Table A.1 provides more details about the time spent searching these cells, and the travels between them. Area x is the gray cell with label x. Period 1 is the time interval $[0, 12]$ hours, Period 2 is time interval $[12, 24]$ and so on.

The searching activities of Table A.1 are all deduced from φ_{ic}^{t} decision variables. For example, the search of Area 2 is represented by two nonzero decision variables as it is searched during two periods. These variables are $\varphi_{ix}^{4} = 8$ and $\varphi_{ix}^{5} = 7$, with x the cell with label 2 and i the sensor. It means that the cell is searched during 8 hours in period 4 and 7 hours in period 5. Since there is no movement between these two periods, we have a single search activity in this area. The moves of the sensor are deduced from q_{ti}, r_{ti}

and $y_{cc'}^{ti}$ variables. For instance, moving from Area 3 to Area 4 takes 10 hours, and spans over two periods, which can be deduced from $q_{7i} = 8$ (8 hours of this inter-period move is done during period 7) and $r_{8i} = 2$ (2 hours done during period 8).

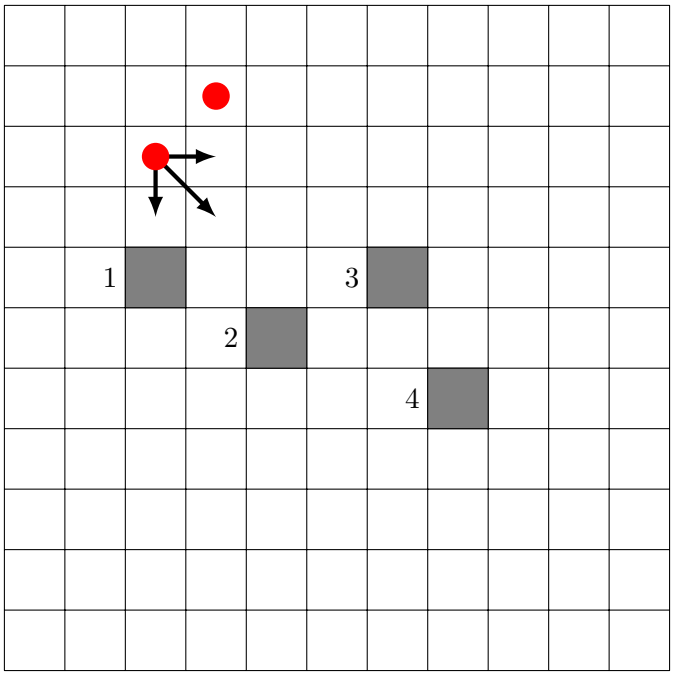

Figure A.1: Search of a lost object in the sea

Activity	Time window	Corresponding Periods
Searching Area 1	[24,30]	Period 3
Moving from Area 1 to Area 2	[30,40]	Periods 3 and 4
Searching Area 2	[40,55]	Periods 4 and 5
Moving from Area 2 to Area 3	[55,65]	Periods 5 and 6
Searching Area 3	[65,76]	Periods 6 and 7
Moving from Area 3 to Area 4	[76,86]	Periods 7 and 8
Searching Area 4	[86,96]	Period 8

Table A.1: A possible plan of search for the boat