Spatial and temporal robustness for scheduling a target tracking mission using wireless sensor networks

Florian Delavernhe, André Rossi, Marc Sevaux

Version of Record: https://www.sciencedirect.com/science/article/pii/S0305054821001076
Manuscript_9eef51493cf1e3f1ede1e37e1a998bc2

Spatial and temporal robustness for scheduling a target tracking mission using wireless sensor networks

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

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

Abstract

Robust scheduling for target tracking with a wireless sensor network (WSN), focuses on the deployment of a WSN in a remote area to monitor a set of moving targets. Each sensor operates on a battery and is able to communicate with reachable sensors in the network. The targets are typically moving vehicles (planes, trains, cars,...) passing through the area. In order to monitor the targets, an activation schedule is sought such that the sensor network is continuously collecting data about the targets. Additionally, the transfer of the data collected to a base station deployed near the network also has to be planned. In this work, we consider that the trajectories of the targets are estimated. *i.e.*, during the mission, at each time instant t, there is a given position where the target is expected. However, such estimations are inaccurate and deviations can occur. In this work, we formulate the problem of spatial robust scheduling. The aim is to produce an activation schedule for the sensors such that the targets are covered as long as they remain no farther from their estimated positions than a maximized value, called the spatial stability radius of the schedule. Afterwards, we formulate the spatio-temporal robustness problem. It is a bi-objective problem, with a spatial stability radius and a temporal stability radius for covering delays and advances. Two algorithms are proposed to solve these problems, and we show their efficiency through several numerical experiments.

Keywords: Linear Programming, Sensor Network, Robust Optimization, Target Tracking, Spatio-temporal uncertainties

1. Introduction

Wireless Sensor Networks (WSN) are a growing technology that can be found in many fields and many applications [2, 22]. They rely on many cheap and easy-to-configure sensors, collecting data about their environment and gathering them for an overall process. The workload is then dispatched on many sensors such that the failure

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

of a few of them does not compromise the mission of the network. The flexibility of such a technology allows it to be deployed in infrastructure-less or dangerous areas, where human resources or expansive technologies cannot be deployed. For example, it is the case with military applications [21] when a WSN is deployed in an hostile territory, without any allied infrastructures, where the sensors might be destroyed or compromised. In such applications, the batteries of the sensors cannot be refilled or replaced, thus the network lifetime (*i.e.*, the time period during which the network can fulfill its mission) is limited. For this reason, energy preservation mechanisms are often considered when designing a solution method for WSN.

In this work, we consider the target tracking mission for a WSN [16]. In this application, one ore more targets are moving inside an area where a WSN is deployed. The sensors have to monitor the targets (record data about them) and the data collected is sent to a base station using hop-communication in the network. In order to preserve energy, there should not be more than one active sensor at a time to monitor a target. Such mechanism greatly extends the lifetime of the network compared to a continuous activation of the sensors [3]. In the case where the trajectories of the targets are known ahead of the mission (*e.g*, a train follows tracks with a time schedule), a schedule can be used to program the activities of the sensors and to guarantee a continuous monitoring of the targets all along their trajectories. However, the trajectories are often estimated and derived from previous collected information, handmade estimations, approximated data and so on. Because these predictions are not always reliable [11] some previous works (*e.g.*, [8, 14]) have been proposed to build a robust schedule based upon estimated trajectories, but that can stand deviations due to uncertain events. In this work, we present a method to produce robust schedules that can stand spatial deviations from the targets trajectory, in the case where temporal estimations are accurate. Indeed, the spatial trajectory of a target is often only estimated, from previous observations (known positions). It is already a difficult problem. Additionally the data may also be inaccurate (due to the accuracy of GPS devices) and the target may not behave as expected (*e.g.*, it could take a parallel route). In [8], a small spatial deviation may lead to lose a target. A robust spatial schedule is a possible response to these issues as it aim at covering the greatest possible deviations. Thereafter, we extend the proposed approach to build robust schedules that can stand both spatial and temporal deviations.

The contributions of this paper are fourfold. (i) We introduce the spatial robustness problem for the target tracking mission with a wireless sensor network. (ii) We propose a method to solve this problem, that computes a solution that is λ-close to the optimal solution, where λ is a parameter. (iii) We extend the previous problem to the spatio-temporal robustness problem, with a combination of two stability radii. (iv) For this problem, we present a method that produces a set of weakly non-dominated solutions that approximate the Pareto front. It is based on an epsilon-constraint method [13] and a dichotomy algorithm [8, 14, 15].

This paper is organized as follows. First, Section 2 introduces the related work on the topic. More precisely, it presents, with many details, the temporal robustness problem since many features of the present work are based on it. Section 3 presents the problem definition, and the stability radii used to represent the robustness of a solution. Next, Section 4 presents the discretization phase. Its features are quite similar to the ones used in previous works but also has novel elements to deal with spatial

robustness. Afterwards, in Section 5, an upper bound for the spatial stability radius is proposed, and used to improve the solution method presented in the next section. Section 6 presents a method to address the spatial robustness problem only, then an extension of this method for the spatio-temporal problem but with a lexicographical order of the two objectives (spatial robustness is maximized first, then temporal robustness) and finally the classical biobjective approach is addressed to approximate the Pareto front. Section 7 presents a set of numerical experiments for assessing the quality of the proposed solutions and the scalability of the results, and Section 8 concludes this work.

2. Related Works

Many works in the literature address the subject of tracking an object in a spatio-temporal context using data collected by sensors (application examples: video surveillance or autonomous driving). In such problems, the objective is to find an algorithm to infer the spatial position of a target over time with the collected data. Typically, the position of an object, person or vehicle, has to be determined on each frame from a recorded video. One particular problem in this research field, is robust target tracking in a crowded environment where the targets have complicated interactions, occlusions (*i.e.* a target hides another one), same appearances,.... Following a target becomes a true challenge in such environments. For example, [12] address this problem and propose a machine-learning method, that tracks the targets by learning a discriminative appearance model for different targets. There are many works addressing this problem, using different algorithms often from the artificial intelligence field. Among other approaches, [23] use a data driven Markov chain-Monte Carlo association, and [7] resort to convolutional neural networks. In this part of the literature, the objective is to track targets that are known to be part of the collected data and are not focused on how to collect this data.

On the other hand, there are also works done in the literature on finding optimal ways to collect data about the targets. It is the case for the temporal robustness problem, that has been treated in [14], and later extended in [8] with communication, multi-target and energy considerations. In this problem, the objective is to find an activation schedule for the sensors such that they capture data about a set of targets all along their trajectories. For that purpose, the trajectories are spatially known and temporally estimated. Each target follows a known trajectory but can be early or late to the points of passage. Thus, to follow a target, the solution sought only has to activate a sensor guaranteed to be in monitoring range of the target. The sensors are limited by their batteries and only one sensor can be activated per target at a time. These works [8, 14] use the stability radius [17] adapted to the target tracking problem and propose a solution method to produce a schedule maximizing this stability radius, and thus the robustness to deviations. The method is based on the notion of *faces*, a set of spatial points covered by the same set of sensors. Thereafter, the trajectory of a target is expressed as a series of time windows with set of candidate sensors that can monitor the target in all the faces where it can be. The difficulty is that the time windows, and the sets of candidate sensors, are depending on the value of the temporal stability radius. Therefore, the authors develop a dichotomy method that finds an interval of values where the optimal stability radius is guaranteed to be located and with fixed association sets of candidate

sensors to time windows. Once this interval is found, a final linear program can be solved to find the optimal solution. In [8], the solution is furthermore optimized to also include energy consumption considerations. The problem of spatio-temporal robustness presented in Section 3 is clearly extending this problem and the method presented in Section 6 exploits the one introduced in [8].

The topic of spatial uncertainties in target trajectory has been addressed for the problem of Moving Object Database [19]. The objective of such a problem is to extend the database technologies to moving objects, with location continuously changing over time (see [9]). However, querying on the location of a target is difficult. Indeed, its position at time t is prone to uncertainties that can be inherent to the imprecision of the devices locating the object or the discrete time updates of the locations. There are two approaches to model uncertainty on the location of a target. The first approach is to resort to the probability density function that associates a possible location for the target with a probability. The second approach is more related to our work since the possible locations are defined by an area of the plane. The idea is that for instants t and $t+1$, the locations are known. However, it is not the case in the time interval between these two instants. The most natural idea is to consider a straight line trajectory between the two points but it is unrealistic and more complex motion model are sought [18]. Hence, bounding the possible locations of the target inside areas has been studied with various figures: *cones*, *beads* [10] or *two inverted half cones*. In [20], the authors bound the trajectory inside sheared cylinders. Such figures arise as the intersection of the discs of radius r (r is a given value) centered on the estimated positions of the target for every time instant t in the considered time interval. The obtained result is very similar to the uncertain area obtained in our spatio-temporal robustness problem (see Section 3), with as difference, the time interval depending on a varying temporal radius. When the motion of a target is constrained by a road network such models are inadequate. Alternative models can be considered [19], however the context of a road network is out of the scope of the present work, where targets can move in any direction.

3. Problem Definition

The purpose of this work is to use a wireless sensor network (WSN) to monitor a set of moving targets, whose trajectories are subject to uncertainties. The network is composed of a set I of m homogeneous sensors, which are all randomly deployed inside an area where a set J of n targets are estimated moving in a given time horizon T. Each target has a spatio-temporal trajectory estimated during T such that the function $\tau_j(t)$ returns the estimated position of the target j at instant $t \in T$. We consider that a trajectory is a set of straight segments. The data collected by a sensor always have to be transferred to the base station, which is inside the area and is not subject to power limitation. The data can be transferred directly or with multi-hop-communication, *i.e.*, a sensor transmits its data to another sensor which forwards it to the base station or to another sensor. Each sensor $i \in I$ is limited by the energy available in its battery, equal to E_i and nonrefillable. We do not explicitly minimize energy consumption as in [8], but we add a constraint limiting the number of sensors activated at the same time and thus limiting energy consumption. This constraint enforces that at any time, no more than one sensor is used to monitor any target. Thus, at time instant t, there is no more

than n sensors activated to monitor the n targets. However, a sensor can be activated to monitor multiple targets at the same time. A sensor can monitor only the targets inside its sensing range R_s and is able to transfer or receive data from another sensor or the base station if it is inside its communication range R_c. $N(i)$ defines the neighborhood of a sensor $i \in I$, *i.e.* all the sensors in communication range of i (distance less than or equals to R_c). We consider stable connections between the sensors, and thus the neighborhood of a sensor does not vary over time. The sensors are multi-role [6] and thus are able to monitor the targets, receive and transfer data. A sensor can execute multiple activities (monitoring, receiving, transferring) at the same time. There are therefore three non-exclusive consumption modes:

- p^S Watts are consumed when monitoring a target.
- p^T Watts are consumed when emitting data.
- p^R Watts are consumed when receiving data.

β is a parameter representing the amount of data (in bytes) collected per target, and per unit of time. At time instant t, a sensor $i \in I$ monitoring $\widetilde{x_1}$ targets is collecting $\widetilde{x_1}\beta$ bytes per unit of time, which implies a power consumption of $\widetilde{x_1}p^S$ Watts for monitoring. If the same sensor receives B bytes per unit of time from other sensors, where $B = \widetilde{x_2}\beta$ corresponds to data collected by the monitoring of $\widetilde{x_2}$ targets by other sensors, then i consumes $p^R\frac{1}{\beta}B = \widetilde{x_2}p^R$ Watts for receiving incoming data. This implies that the re-transmission of these data toward the base station incurs a power consumption of $\widetilde{x_2}p^R$ Watts for sensor i. Sensor i is therefore transmitting $C = \widetilde{x_1}\beta + B$ bytes per unit of time to other sensors or to the base station. Hence its power consumption is $\widetilde{x_1}p^S + \widetilde{x_2}p^R + (\widetilde{x_1} + \widetilde{x_2})p^T$ Watts.

We seek to produce a continuous coverage of the targets (monitoring and transfer of the data to the base station) over the horizon of time T. It is done by implementing, in the network, an activation schedule with all the sensors' activities planned before the start of the mission. This schedule is computed in such a way that it covers the estimated trajectories of the targets. A robust schedule is sought to achieve this goal. Such a schedule also guarantees to cover some deviations from the estimated trajectories, and the offered robustness is measured by two stability radii. As defined by [17], and [14] for the target tracking mission, a stability radius is a robustness indicator. A schedule remains feasible as long as some uncertain parameters remain below the stability radius centered at the estimated values of these parameters. With two stability radii, the robustness of the schedule proposed in this work is twofold. The first indicator is the spatial stability radius denoted by R which also denotes its numerical value. It aims to provide robustness against spatial deviation from the estimated trajectories, *e.g.* the target is passing through a parallel route. Then at every instant $t \in T$, the network covers the target $j, \forall j \in J$ wherever it is inside the disc (or the ball, in a three dimensional context) of radius R and centered on $\tau_j(t)$. Hence, the schedule remains feasible if the distance between the target and its estimated position is always less than or equal to R. The second stability radius, ρ which also denotes its numerical value, is temporal (see [8]). In our problem, this means that the schedule is guaranteed to monitor the targets, as long as their temporal trajectories are in the stability radius ρ

centered around their estimated times. In other words, it means that a target is covered as long as it has no delay or advance greater than ρ. Thus, each point p of the estimated trajectory of a target $j \in J$, such that $\exists t, \tau_j(t) = p$, is covered during the time window $[t - \rho, t + \rho]$.

Combining the two stability radii gives a more complex problem than just considering the two separately. Indeed, a non null spatial stability radius means that the schedule, at each instant, has to cover several points in space where the target can be. By contrast, with a non null temporal stability radius, the schedule has to cover delays and advances to the many possible locations where a target can be. More formally, having a schedule with a spatial stability radius R and a temporal ρ means that the target $j \in J$ at time t is guaranteed to be covered if it is inside a disc (or ball) centered on $\tau_j(t')$ such that $t' \in [t-\rho, t+\rho]$. In other words, the target is guaranteed to be covered if it is always not farther than R meters from a point of its estimated trajectory, estimated to be reached less than ρ seconds earlier or later. The uncertain area formed by these stability radii, at time t, is the intersection of all the discs of radius R centered on all the estimated points of the trajectory in $[t - \rho, t + \rho]$. The shape of the area is sheared cylinders. Such figure can be found in the literature of uncertain spatial trajectory as in [20] (see Section 2).

We want to maximize both the spatial stability radius R and the temporal stability radius ρ. The problem is bi-objective with two conflicting objectives. A quick example to asses this conflict is the coverage of a single point trajectory, *i.e.*, a stationary target to cover during one unit of time. In this example, the optimal solution for the spatial stability seeks for the closest sensor to this point with enough energy for the one time coverage of this point. Whereas, the optimal temporal stability radius seeks for one sensor in range of this point with the maximal amount of energy (to cover it as long as possible). The method sought needs to be able to produce a set of solutions representing well the Pareto front. In this work, we present a method that computes a set of weakly nondominated solutions widely and evenly spread on the Pareto front.

All notations related to our problem are summarized in Table 1.

4. Discretization

Discretization is the process of transforming a part of the geometric data of the problem into a set of discretized data. Thereafter, these data can be used for modeling and solving the problem. The discretization used in this work reuses the principles presented in [8] for the temporal stability radius. These principles have been extended in the present work for the spatial stability radius and afterwards to the bi-objective problem. Note that in this work, we make the assumption that the given estimated trajectories are sets of segments, *i.e.*, the targets have straight trajectories between different waypoints. It is also assumed that the target move at constant speed on each segment, but speed may change from a segment to the next one.

In this section, we use Figure 1 to illustrate discretization. It is very similar to the one used in [8], with however an added segment in the trajectory for a better illustration of our new contributions. There are three sensors, 1, 2 and 3, with yellow discs showing their monitoring ranges. A single target is moving alongside the black arrow, its starting position is marked by a $\times$ in the figure. It starts its trajectory in a part of the area only

Sets	
$I = \{1, \ldots, m\}$	Set of sensors
$J = \{1, \ldots, n\}$	Set of targets
Indexes	
$i \in I$	Index of sensor
$j \in J$	Index of target
Parameters	
E_i	Energy of the battery of sensor i
p^S	Power consumption in watts for sensing
p^R	Power consumption in watts for receiving data
p^T	Power consumption in watts for transmitting data
R_c	Communication range of a sensor
R_s	Sensing range of a sensor
T	Time horizon
$\tau_j(t)$	Estimated position of target $j \in J$ at time $t \in [0, T]$
$N(i)$	Set of all the sensors in range of communication of sensor i
β	Units of data collected during on unit of monitoring time
Decision variables	
ρ	The temporal stability radius, also denoting its value
R	The spatial stability radius, also denoting its value

Table 1: Summary of the notations

covered by sensor 1, gets under the range of sensor 2 under the range of sensor 3 before leaving the range of sensor 1 and finally finishing its trajectory there. The time horizon is $T = [0, 20]$ and B is the base station.

4.1. Discretizing the estimated trajectories and possible communications

In our problem, we want to represent the trajectories of the targets as sets of candidate sensors available during time windows. A face f is a set of spatial points that are all covered by the same set of sensors. Therefore, for a schedule, being able to monitor a target j at time t corresponds to have a sensor covering all the faces where j can possibly be at time t (*i.e.*, all the points inside the uncertain area obtained with the stability radii). Hence, any given pair (R, ρ) defines at each instant t for a target $j \in J$, a set of faces where the target has to be covered. More precisely, the WSN has to cover all the faces having a nonempty intersection with at least one disc of radius R centered at $\tau_j(t')$, such that t' is in $[t - \rho, t + \rho] \cap T$. The intersection of all these faces defines a set of candidate sensors (*i.e.*, a face) that can cover the target (if the intersection is empty, then the target cannot be covered). With the example of Figure 1, if at time instant t, the face $\{1\}$ and the face $\{1, 2\}$ need to be covered, the set of candidate sensors is $\{1\} \cap \{1, 2\} = \{1\}$.

We delimit the time windows, where a target can be monitored by a same set of candidate sensors, using the notion of *tick*. Our definition of a tick is an extension of the one used in [8], adapted now to the spatial stability radius. In the present work, a tick is a tuple (tick ID, sensor ID, target ID, date, position) such that the date and the

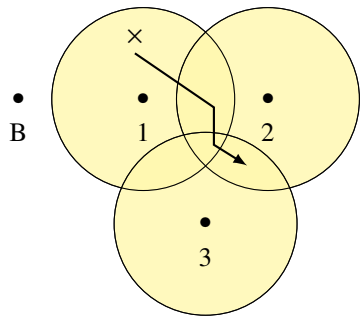

Figure 1: Example with three sensors

position are variables depending of R and ρ. Note that the sensor ID can be empty to represent the beginning or the end of the target trajectory as shown later. It represents the first or last point of a part of the trajectory such that a sensor covers entirely the uncertain area defined by the spatial stability radius. The associated date is the earliest or latest date when the target is guaranteed to be covered on this point (*i.e.*, inside the temporal stability radius). Thus, the elements associated to a tick (position and date) depend of the values of the stability radii. For a given R, the position associated to a tick is a point on the trajectory of the target such that the disc of radius R centered on this point is tangent to the sensing range. The date is the estimated date for this point plus or minus ρ depending if the sensor becomes or is no longer candidate.

If a sensor becomes candidate, it is an *entering* tick; and it is a *leaving* tick if a sensor is no longer candidate. The k-th tick of target $j \in J$ is denoted by t_k^j. It is associated to a date $d_{t_k^j}$ and a spatial position $p_{t_k^j}$ on the estimated trajectory. $d_{t_k^j}$ is the earliest (if leaving) or latest (if entering) date when the target j is expected to be anywhere inside the disc centered on position $p_{t_k^j}$ and with a radius R. The tick t_k^j is also associated to a sensor s_k^j and σ_k^j, an integer value which is $+1$ if the tick is entering, and -1 if it is leaving. By convention, the start and the end of a trajectory are represented with a first and a last tick, which are respectively leaving and entering [14]. These ticks are used to represent a target starting or finishing its journey earlier or later, they are not associated to a sensor and only the temporal stability radius is affecting them. These ticks can be understood as representations of the instants when the target leaves or enters the empty set of candidate sensors. For a target $j \in J$, a time window Δ_k^j is the duration between two successive ticks t_k^j and t_{k+1}^j. The k-th time window is such that $k \in K_j = \{1, 2, \ldots, H_j\}$, where H_j is the number of time windows for target j. The set of candidate sensors for a time window $k \in K_j$ is $S_j(k)$. The set of ticks for target $j, \forall j \in J$ is denoted by T_j such that $T_j = \{1, \ldots, H_j + 1\}$.

The possible direct communications in the network between the sensors and the base station, are represented by the communication digraph $\vec{G}_c = (I \cup \{B\}, A)$, where A is all the pairs of elements of $I \cup \{B\}$ that are close enough for direct communication. Since

the sensors are not supposed to move, the graph remains unchanged for the entire time horizon. To represent the data transfer to the base station, we use a flow representation. The flow is computed from the monitoring sensor to the base station, in $\vec{G}_c$. With the example of Figure 1, we obtain the digraph $\vec{G}_c$ of Figure 2.

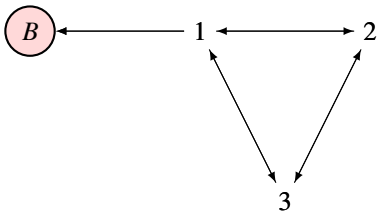

Figure 2: The communication digraph $\vec{G}_c$ for the example of Figure 1

In the example of Figure 1, for $R = 0$ and $\rho = 0$, the estimated trajectory of the target can be discretized with only the four faces that the target should visit: {1}, {1,2}, {1,2,3} and {2,3}. The problem has therefore five ticks and four time windows presented in Table 2.

Face	{1}	{1,2}	
Time window length (given)	$\Delta_0^1 = 5$	$\Delta_1^1 = 10$	
Tick	Tick 0 $t_0^1 = 0$ $\sigma_0^1 = -1$ $s_0^1 = \emptyset$	Tick 1 $t_1^1 = 5$ $\sigma_1^1 = +1$ $s_1^1 = 2$	Tick 2 $t_2^1 = 15$ $\sigma_2^1 = +1$ $s_2^1 = 3$

Face	{1,2,3}	{2,3}	
Time window length (given)	$\Delta_2^1 = 2.5$	$\Delta_3^1 = 3$	
Tick	Tick 3 $t_3^1 = 17.5$ $\sigma_3^1 = -1$ $s_3^1 = 1$	Tick 4 $t_4^1 = 20.5$ $\sigma_4^1 = +1$ $s_4^1 = \emptyset$	

Table 2: The five ticks of the example of Figure 1

Tick 1 represents the beginning of the trajectory of the target, thus it is not associated to any sensor. In fact, this tick is useful for representing cases where a target starts moving along its trajectory earlier than estimated. Tick 2 represents the moment when the target is entering the range of sensor 2. For example here, $\sigma_1^1 = +1$ since tick 2 is an entering tick, thus the date t_1^1 is increasing and with increasing stability radii the instant when sensor 2 becomes candidate is delayed. Tick 3 models the time when the target is entering the range of sensor 3. Tick 4 models the target leaving the range of sensor 1, it is a leaving tick thus $\sigma_3^1 = -1$. Finally, the last tick represents the end of the trajectory, to deal with those situations where the target reaches its destination later than estimated. Once again, by convention, this tick is not associated to a sensor. The value of Δ_0^1, the length of the first estimated time window (with the target only

in range of sensor 1) is equal to $t_1^1 - t_0^1 = 5 - 0 = 5$, *i.e.*, the instant when sensor 2 becomes candidate minus the instant the target starts its trajectory. The other Δ_k^1 values are obtained similarly.

All the notations introduced for discretization are summarized in Table 3.

t_k^j	Tick ID of the k-th tick of target j
$d_{t_k^j}$	The date of the tick t_k^j
$p_{t_k^j}$	The spatial position (on the estimated trajectory of j) corresponding to the tick t_k^j
s_k^j	Sensor ID of the tick t_k^j (becoming, or no longer, candidate)
σ_k^j	$+1$ if t_k^j is an entering tick, -1 otherwise
K_j	The set of time windows for the target j
H_j	The number of time windows for the target j
Δ_k^j	The k-th time window of target j
$S_j(k)$	The set of candidate sensors for the time window k
T_j	Set of all ticks of target j

Table 3: Summary of the notations for discretization

4.2. Dates and points associated to the ticks

In the previous section, we introduced the notion of tick. As mentioned, the actual position and date associated to a tick depends on the values of the stability radii. The ticks are first computed considering the trajectory estimation as shown in Table 2. Afterwards, with given values of R and ρ, the ticks are spatially and temporally moved and some are added. As presented in the definition of a tick, it always represents the first instant covered by the stability radii when a sensor is no longer candidate or the first instant when a sensor is guaranteed to cover the target wherever it is inside the stability radii. An entering tick is always postponed when one of the stability radii increases as this delays the instant when a sensor is candidate. By contrast, a leaving tick is advanced when a stability radius value increases. It is necessary to compute the tick dates and positions according to the values of the radii. Changing the dates of the ticks means also that the time windows and sets of candidate sensors are changing with different values of R and ρ. However, they can easily be computed once the tick dates are updated, using the initial set of candidate sensors (guaranteed to cover the start of the trajectory) and using the ticks ordered by date.

To illustrate the ticks moving or the ticks added, we re-use the example of Figure 1. With different given values of $R > 0$, the situation changes, and it is illustrated by Figures 3a, 3b and 3c, where for more clarity only the sensor 2 is represented. In each figure, the value R is equal to the radius of the red circle. Each red dot on the estimated trajectory is the center of a circle of radius R and tangent to the sensing range of sensor 2. Hence, these dots all represent ticks that delimit the part of the trajectory (in black) that can be covered by sensor 2 for the corresponding value of R. The red part of the trajectory is not guaranteed to be covered by sensor 2 when considering

a spatial stability radius R. The red dots, on which the red circle is centered, always represent the same tick, with a varying position depending of R. As we can see in this example, the position of this tick is always computed such that its distance with the closest point on the sensing range is equal to R.

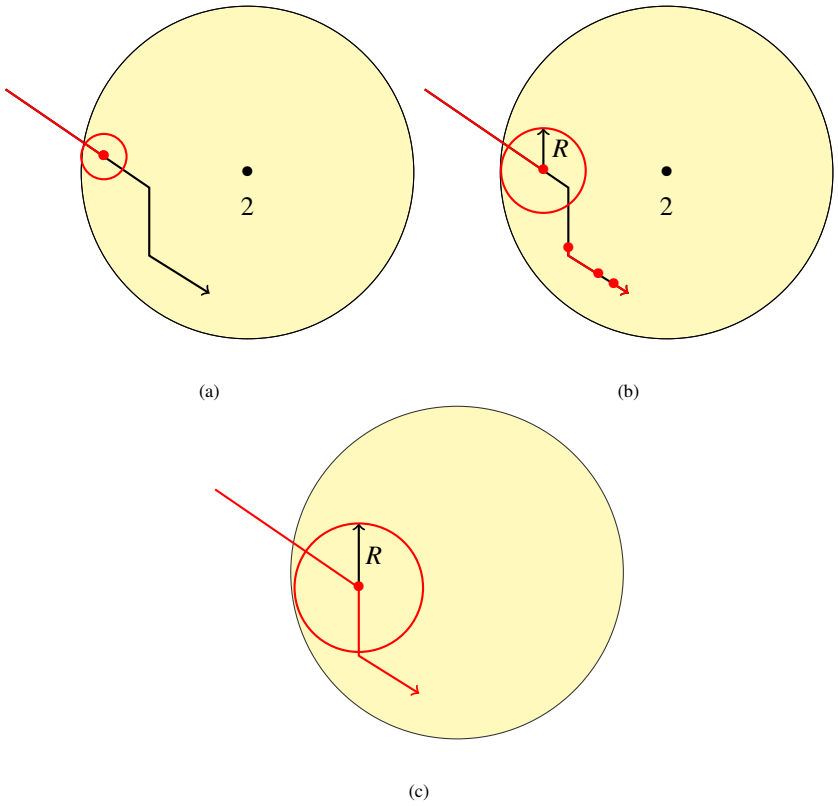

Figure 3: Moving ticks of the example of Figure 1

4.3. Computation of the date and position of a tick

As seen in [8], the date of a tick changes with the value of ρ to represent delays and advances. In the present work, for a given temporal stability radius ρ, the date of the tick t_k^j is:

$$d_{t_k^j} = \tau_j(p_{t_k^j}) + \rho\sigma_k^j, \forall j \in J, \forall k \in T_j \quad (1)$$

Thus, the date of a tick depends on both R (for $p_{t_k^j}$, the corresponding estimated position) and ρ.

When R increases, the k-th tick of target $j \in J$, t_k^j, is moved such that $p_{t_k^j}$ is always on the estimated trajectory of j and is R meters away from the limit of the range of the sensor s_k^j. *i.e.*, the disc of radius R centered at $p_{t_k^j}$ has to be inscribed in the sensing disk

of sensor s_k^j, and is tangent to this disc. This is shown by the red circles of Figures 3a, 3b, 3c. Moving the spatial positions of the ticks accordingly to the definition of a tick always makes them delimiting the first or the last point of a section of the estimated trajectory where a sensor is candidate. Note that when R increases, an entering tick $t_k^j, j \in J, k \in T_j$ can reach the end of the trajectory of j and is therefore removed: the sensor will not become candidate. The same applies for a leaving tick that can reach the start of the trajectory, thus the tick is removed and the sensor is no longer candidate to monitor the target at the beginning of its trajectory. For a target $j \in J$, if a leaving tick and an entering tick of the same sensor are passing through the same date, and exchanging their date order, they are both removed. In that case, the sensor is no longer candidate to monitor a nonempty section of the estimated trajectory passing through its sensing range. In the example of Figure 3b, each remaining part of the trajectory where sensor 2 is still candidate is delimited by the two ticks (represented by red dots). With a greater value R such that we obtain Figure 3c, these ticks have moved and eventually have reached the same date. When this happened, they have been removed as the sensor 2 is never candidate to cover the target for this section of the trajectory.

With a given value of R, the position $p_{t_k^j}, \forall j \in J, k \in T_j$ can be computed with a non linear function using the coordinates of s_k^j and the coordinates of the line segments of the estimated trajectory of j. The value of ρ does not change $p_{t_k^j}$. We consider, in the rest of this paper, the function $\gamma_k^j(R)$ that geometrically computes the position $p_{t_k^j}$ for the value R. This function solves a second order polynomial function, in a constant time. It just finds a point in the plane, as the intersection between a circle and a segment. More precisely, the function finds a point on the trajectory of the target, with a distance to the corresponding sensor equal to $R_s - R$. Hence:

$$p_{t_k^j} = \gamma_k^j(R), \forall j \in J, \forall k \in T_j \quad (2)$$

To summarize, for any given value of R and ρ, the position of a tick is computed using the linear function (2) and its date is calculated with (1).

4.4. Adding ticks

Moreover, there are ticks that appear when R reaches certain values. For example in Figure 1, when R is large enough, it is possible to have multiple time windows where sensor 2 is not longer candidate as it is illustrated in Figure 3b. Each one of the red dots in this figure is the center of a disc of radius R and tangent to the sensing range. In that particular case, the status of sensor 2 alternates between *not candidate*, and *candidate*, whereas for $R = 0$ or for R as in Figure 3a, it was simply first not candidate, then candidate until the end. To represent such events, new ticks are needed and thus new time windows appear. In the example of Figure 3b, the ticks appear in the last way point and at the end of the estimated trajectory.

A tick is added at a spatial point p if p is on the estimated trajectory of a target $j \in J$, under the sensing range of a sensor s and is further from s than any other neighbor point on the estimated trajectory. In other words, there is no ξ such that all the points on the estimated trajectory and in the disc of radius ξ centered at p, are further from s than

from p. This means that p will be the first point of its neighborhood no longer covered by s when R increases. It is also not corresponding to an other tick that moved to this point. Since we only consider the estimated trajectories of the targets as sets of consecutive line segments, such points are only the extremities of the segments. A tick corresponding to a sensor s appears to one extremity e if and only if the value of R is greater than the distance between e and s minus R_s.

If the extremity e is the start or the end of the trajectory, only one tick is added, otherwise two ticks are added (one entering and one leaving). The introduction of new ticks leads to define new time windows with less candidate sensors.

Figure 3b illustrates these principles. On the chronological order of the estimated trajectory, the ticks are the following. The first tick is the initial tick, the target arrives in range of 2 when $R = 0$. The two following ticks appeared on the last waypoint of the target when R is large enough so that the uncertainty disc centered at this extremity is no longer entirely inscribed in the range of sensor 2. The last tick appeared alone at the end of the trajectory.

To summarize, discretization transforms the estimated trajectories of the targets into sets of ticks and with given values of ρ and R, the ticks are associated to different positions and dates, some ticks are removed and some are added. The ticks delimit time windows and moving them adds or removes sensors from the set of candidates sensors to monitor the target.

5. Upper bounds on the spatial stability radius of a schedule

Two upper bounds are proposed for the spatial stability radius, with the final upper bound returned being the lowest of both. The bounds are computed considering $\rho = 0$ and used only when maximizing R as shown in Algorithm 3. The computation of this upper bound is used to reduce the running time of the solution method by reducing the search space.

The first upper bound searches for the first value of R that creates a face with no candidate sensor, thus the stability radius cannot exceed this value; that is, the lowest value of R with which there is an empty set of candidate sensors associated to a non-null time window. It always corresponds to either the intersection between two ticks (see [8]), or the appearance of a new time window. To find this value of R, we need to compute, for each extremity of each segment of the estimated trajectories, the last time window that will appear. Because each of the time windows corresponds to one sensor no longer candidate for this extremity, the last time window corresponds to a face with no candidate sensor. These values are computed as the maximum distance between the extremity and any sensor whose distance to this extremity is less than or equal to R. The second part in the computation of this bound is, for each segment, to find the lowest value of R that corresponds to an intersection between two ticks such that the intersection point is no longer covered by any other sensor when the intersection occurs. So, for each intersection of ticks, we need to compute the position of others ticks. These values are computed using the position of the sensors and the segments' coordinates. The complexity of this bound is $O(MN^3)$ with N the number of possible ticks and M the number of faces. Both values are pseudo-polynomial (see Section 6.4) and upper-bounded by $2qm$ with q the number of segments in the trajectories of the

targets and m the number of sensors. However, this upper bound removes a lot of possible values for R.

The second upper bound computes the values of R such that there is not enough energy in some set of candidate sensors to cover the target during the total length of the initial time window. To this end, we consider the problem with $R = 0$, and for all targets and all time windows, we compute the total energy in the batteries of the candidate sensors, and the total energy needed in the corresponding time windows, *i.e.*, $\forall j \in J, \forall k \in K_j$, we compute $\sum_{i \in S_j(k)} E_i$, and $(t^j_{k+1} - t^j_k) \times (p^S + p^T)$. The former is the energy available and the latter the energy needed. Then, for each candidate sensor ($\forall i \in S_j(k)$), we compute the minimal value of R for which the sensor is no longer guaranteed to cover the target at any time during this time window. This values uses the closest point from the sensor which is on the estimated trajectory and inside the time window. It is obtained using the segments' coordinates and the sensors' coordinates. When such a value is reached, the battery of the sensor can be removed from the available energy. Thus to find a lower bound, we need to find the lowest value of R for which the energy available is less than the energy needed. The complexity of this bound is $O(NM)$ with N the number of possible ticks and M the number of faces (both pseudo-polynomial, see Section 6.4).

Algorithm 1 synthesizes the computation of the first upper bound, denoted by UB. Its value is initialized to R_s (a trivial upper bound). Compute_All_Ticks(*problem*,(j,k)) computes and returns the set of ticks of *problem* that will pass through the k-th time window of j. Intersection(*problem*,$tick_1$,$tick_1$,(j,k)) returns a pair (p,R) such that R is the greatest value of the spatial stability radius with the two ticks having the same associated position p, and p is estimated to be visited by target j in its k-th time window. Is_In_Range_Of_No_Sensor(*problem*,p,R) returns True provided that with a spatial stability radius value R, the position p is not cover by any sensor, and False otherwise.

Algorithm 1: First Upper Bound on R

Data: *problem* is the discretized problem data, with the ticks, time windows, candidate sensors,...
Result: the value of the first upper bound on R
UB$\leftarrow R_s$
for $j \in J, k \in K_j$ **do**
  $TICK \leftarrow$ Compute_All_Ticks(*problem*,(j,k))
  for $tick_1$ *in* $TICK$ **do**
    for $tick_2$ *in* $TICK$ **do**
      $(p,R) \leftarrow$ Intersection(*problem*,$tick_1$,$tick_1$,(j,k))
      if *Is_In_Range_Of_No_Sensor(problem,p,R)* **then**
        UB$\leftarrow$min(UB,R)
return UB

Algorithm 2 synthesizes the computation of the second upper bound. This upper is initialized to R_s as in Algorithm 1. Next_Non_Covering_Sensor(*problem*,(j,k), R)

returns (i, R') such that R' is the lowest value greater than or equal to R such that sensor $i \in S_j(k)$ cannot be used for any instant of the k-th time window of target j when considering a stability radius greater than R'.

Algorithm 2: Second Upper Bound on R

Data: *problem* is the discretized problem data, with the ticks, time windows, candidate sensors,...
Result: the value of the second upper bound on R
UB$\leftarrow R_s$
for $j \in J, k \in K_j$ **do**
  $E_{\text{available}} \leftarrow \sum_{i \in S_j(k)} E_i$
  $E_{\text{needed}} \leftarrow (t^j_{k+1} - t^j_k) \times (p^S + p^T)$
  while $E_{needed} \leq E_{available}$ **do**
    $(i, R') \leftarrow$ Next_Non_Covering_Sensor(*problem*,(j, k), R)
    $E_{\text{available}} \leftarrow E_{\text{available}} - E_i$
    $R \leftarrow R'$
  UB$\leftarrow$ min(UB, R)
return *UB*

6. Solution Method

We present in this work a solution method for the biobjective problem. This method relies on two algorithms, each one optimizes a different objective while the other one is fixed. For the temporal stability radius, we use the algorithm introduced in [8]. Note that we use only the first step of the algorithm presented in [8], maximizing ρ, and do not consider the energy minimization afterwards. For the spatial stability radius, we use a novel algorithm, presented in Section 6.1. The solution method combines the two algorithms and has two steps. First, Section 6.2 presents the solution method for the hierarchical case (R is the primary objective), and secondly this method is extended for the biobjective problem (both objectives matter as much). Section 6.4 analyses the complexity of the overall method and thus the method for the spatial robustness.

6.1. Solving the spatial robustness problem

The solution process for the maximization of the spatial stability radius is very similar to the maximization of the temporal stability radius presented in [8].

The difficulty when solving such a problem is that the set of time windows and the sets of candidate sensors depend on the value of the stability radius. Moreover, the sizes of the time windows depend on the dates of the ticks and thus of their positions, that are computed by applying a non linear function on R. Therefore, to maximize the spatial robustness, we cannot solve one linear program where R is a decision variable. With a linear program, we can only test the feasibility of a given value of R.

The solution process builds feasible schedules for different given values of R until a schedule with a value of R guaranteed to be close enough of the optimal solution is

found. *i.e.*, we iterate until two values of R, a and b, have been found such that there exists a feasible solution for $R = a$ but not for $R = b$, and $0 \leq b - a \leq \lambda$ with λ a given parameter that controls the optimality gap. At the end of the solution process, the optimal solution value is guaranteed to be in $[a, b)$ and thus the method returns the schedule obtained with $R = a$. The smaller λ is, the more precise the method is but the more running time it requires.

In order to decide if the problem is feasible for a given value of R, we just need to compute the corresponding time windows and candidate sensors, by moving and adding the ticks corresponding to this value. These ticks and positions are obtained as presented in Section 4 when considering $\rho = 0$ and without considering a tick for the start of the trajectory and one for the end. Afterwards, the linear program (LP) referred to as Model 1 is solved in order to find a feasible schedule. The model is linear since the value of R is a given constant. This LP is in fact a satisfaction problem as it has no objective function, only a feasible solution is sought. The model solved is the following:

$$\sum_{j \in J} \sum_{k \in K_j | i \in S_j(k)} x_{jik} \quad p^S \quad + p^R \quad \frac{1}{\beta} \sum_{i' \in N(i)} f_{i'i} \quad + p^T \quad \frac{1}{\beta} \sum_{i' \in N(i)} f_{ii'} \quad \leq E_i \quad \forall i \in I \tag{3}$$

$$\sum_{j \in J} \sum_{k \in K_j | i \in S_j(k)} x_{jik} \beta \quad + \quad \sum_{i' \in N(i)} f_{i'i} \quad - \quad \sum_{i' \in N(i)} f_{ii'} \quad = 0 \quad \forall i \in I \tag{4}$$

$$\sum_{i \in S_j(k)} x_{jik} \quad = \Delta_k^j \quad \forall j \in J, \forall k \in K_j \tag{5}$$

$$x_{jik} \geq 0 \quad \forall j \in J, \forall k \in K_j, \forall i \in S_j(k) \tag{6}$$

$$f_{ii'} \geq 0 \quad \forall i \in I, \forall i' \in N(i) \tag{7}$$

Model 1: Model solved for the spatial robustness problem

The decision variables $x_{jik}, \forall j \in J, \forall i \in I, \forall k \in K_j$ are the monitoring times of the target j by the sensor i during its k-th time window. Constraints (3) correspond to the limitations of the batteries. Constraints (4) are flow constraints, where the data collected and received by a sensor is transmitted. Constraints (5) states that the sum of the activities of the sensors in a time window is equal to its length. The following table (Table 4) summarize the added notations.

x_{jik}	The duration of the monitoring activity by the sensor $i \in I$ of the target $j \in J$ during the time window $k \in K_j$
$f_{ii'}$	The total amount of data transferred during the horizon of time from $i \in I$ to $i' \in N(i)$

Table 4: Summary of the notations for the model

We use two dichotomy methods to find the values of R to test. First, we use a dichotomy on a discrete set of values of R that create time windows, or change the sets of candidate sensors. These values correspond to ticks added or ticks meeting (*i.e.*, two ticks are moved to the same date). Note that when a value of R corresponds to the addition of ticks, it has a time window with a null length which does not need to be considered. It means that this time window should only be considered when R is greater than this value. This set of values is reduced by removing all the values greater

than the upper bound. This first dichotomy returns an interval $[a, b)$ where the optimal solution is located. Secondly, we use a continuous dichotomy on this interval to find a feasible schedule such that its value of R is no farther than λ from the optimal solution. This second dichotomy returns the interval $[c, d)$ such that $0 \leq d - c \leq \lambda$. The schedule found with its spatial stability radius equal to c is the returned solution. The linear program presented in model 1 is solved at each stage of the dichotomy to reduce the aforementioned intervals.

Algorithm 3 summarizes the overall method. Move_Spatial(*problem*, x) is the function returning *problem* discretized with its time windows, candidate sensors, etc, obtained when $R = x$. It computes the dates and positions associated to a tick with (1) and (2), using R as an input. Let us remind that it simply uses a geometric function, computing intersection points between line segments and circles in constant time. NextValueDichotomy(*List_of_values*, a, b) returns a value x of *List_of_values* obtained by a dichotomy in open interval (a, b), if such a value exists, and $x = a$ otherwise. Compute_Upper_Bound returns the best upper bound (lowest) from Section 5 for *problem*.

6.2. Solving the hierarchical problem

To solve the hierarchical problem, where R is optimized first and then ρ, we just need to consecutively use the solution methods for spatial then temporal stability radii. Both methods rely on the same notion of *tick*, and the optimization of ρ can use the resulting ticks of the maximization of R. Indeed, in both cases, a tick is an instant separating two time windows and it is associated to a physical position. This position is the last or the first position guaranteed to be covered by a sensor. The optimization of the spatial stability radius is computing points on the estimated trajectories of the targets corresponding to ticks, and maximizing ρ is considering that the targets can arrive earlier or later to these points thus moving the ticks. The only difference is that the temporal stability radius is using two more ticks per target to represent the start and the end of the trajectory. Thus, after maximizing R, we need to add these two ticks for each target. It is a leaving tick for the beginning of the trajectory and an entering tick for the end, as presented in [8].

The only particularity is that if the optimal value R_{opt} of R returned corresponds to the addition of one or two ticks, such ticks should not be included for the temporal solution process. Indeed, as mentioned before (Section 6.1), the resulting time window has only to be considered when R is greater than R_{opt} which can never happen.

To conclude, the hierarchical problem can easily be solved by first maximizing R using the method presented in Section 6.1. Afterwards, we quickly add or remove a few ticks from the tick pool obtained with the optimal value of R and finally we maximize ρ using the method introduced in [8].

6.3. Computing solutions for the spatio-temporal problem

In this section, we focus on the bi-objective formulation of the problem, where there is no preference defined on the objectives. Thus, the method sought needs to produce a set of solutions representing trade-offs between the objectives. This set has to approximate as best as possible the Pareto front of the problem. We use an ϵ-constraint

Algorithm 3: Maximization of R

Data: *problem* is the discretized problem data, with the ticks, time windows, candidate sensors,... obtained when $R = 0$. *List_of_values* is the set of values of R corresponding to ticks added or ticks meeting. λ is the precision sought.

Result: -1 if the problem is unfeasible, otherwise a value of $R \geq 0$ λ-close from the optimal solution

```
R_ub ← Compute_Upper_Bound(problem) ;
R_lb ← 0 ;
problem ← Move_Spatial(problem,R_ub) ;
if Solve_Model_1(problem) = feasible then
    return R_ub
problem ← Move_Spatial(problem,R_lb) ;
if Solve_Model_1(problem) ≠ feasible then
    return −1
/* R_ub unfeasible upper bound, R_lb feasible lower bound     */
/* Start of the discrete dichotomy                            */
R ← Next_Value_Dichotomy(List_of_values,R_lb,R_ub) ;
while R_lb ≠ R do
    problem ← Move_Spatial(problem,R) ;
    if Solve_Model_1(problem) = feasible then
        R_lb ← R ;
    else
        R_ub ← R ;
    R ← Next_Value_Dichotomy(List_of_values,R_lb,R_ub) ;
/* Start of the continuous dichotomy                          */
while R_ub − R_lb ≥ λ do
    R ← (R_ub + R_lb)/2 ;
    problem ← Move_Spatial(problem,R) ;
    if Solve_Model_1(problem) = feasible then
        R_lb ← R ;
    else
        R_ub ← R ;
return R_lb
```

method [13] that produces a set of weakly non-dominated solutions. With our method, the solutions returned can only have their spatial stability radius improved and by at most ϵ (a parameter) only. Each solution is generated using a hierarchical solution process similar to the one introduced in Section 6.2. Each generated solution has its value of R bounded by R_+, where R_+ is lower than the R value of the last solution found. It also has a lower bound ρ_- on ρ, also depending on the value of ρ of the last solution found. R_+ is decreases in the process while ρ_- increases.

The global idea is to start from an extreme point of the Pareto front, and to generate a series of weakly nondominated points by bounding one objective with decreasing values and increasing the other. The computation of one solution is reusing the last obtained solution as a starting point and is expected to be faster than if all computations would be performed from scratch. Indeed, the proposed approach starts directly with set of time windows and candidate sensors known to be feasible for $R = R_+$ and $\rho = \rho_-$. Only ρ can be increased.

The overall method is presented in Algorithm 4. *Solve_Maximization_R* maximizes the value of R in the given problem using the method presented in Section 6.1. It returns -1 if the problem is unfeasible. The function *Solve_Maximization_ρ* maximizes ρ in the given problem with the method presented in [8]. Its last parameter is a lower bound on the optimal value of ρ. *Move_spatial* returns the problem with the ticks, time windows and candidate sensors obtained with the given value of R as mentioned for Algortihm 3.

Algorithm 4: Approximating the Pareto front

Data: *problem* is the discretized problem data, with the ticks, time windows, candidate sensors,... obtained when $R = 0$ and $\rho = 0$. ϵ is the minimal value of the decrease of R between two different returned solutions

Result: Set of solutions approximating the Pareto front

```
Solutions ← ∅ ;
R_ub ← Solve_Maximization_R(problem) ;
if R_ub < 0 then
    return Solutions;
R_+ ← R_ub;
ρ_- ← 0;
ρ_ub ← Solve_Maximization_ρ(problem,ρ_-);
do
    problem ← Move_Spatial(problem,R_+);
    ρ_- ← Solve_Maximization_ρ(problem,ρ_-);
    Add(Solutions, (R_+,ρ_-)) ;
    R_+ ← R_+ − ε ;
while ρ_- < ρ_ub;
return Solutions ;
```

The first step is to find the upper bounds R_{ub} and ρ_{ub}, for respectively R and ρ. To obtain these bounds, the method just needs to solve independently the maximization of

ρ and the maximization of R. Note that if no value has been found for R_{ub}, the problem is not feasible even for the estimated trajectories ($R = 0, \rho = 0$) and the algorithm stops. For ρ_{ub}, we use the exact same method as in [8], with a lower bound equal to 0 (the problem is known to be feasible). R_{ub} is the starting point of the method, *i.e.*, the value of R of the first computed solution. ρ_{ub} is the ending point, *i.e.*, the method stops when a solution with a temporal stability radius equal to ρ_{ub} is found.

The method generates solutions with decreasing values of R, denoted by R_+, and increasing values of ρ, denoted by ρ_-. For each iteration of the method, it starts by creating the problem corresponding to the spatial stability radius R_+. Afterwards, it optimizes the value of ρ for this given R. The method searches for a solution with $\rho \geq \rho_-$, and such a lower bound is useful in reducing the running time. The solutions found with $R \leq R_+$ and $\rho = \rho_-$ are weakly dominated by a solution already found, and possibly dominated by the Pareto front, thus they are not kept. The other solutions found are added to the set of returned solutions. The value of R_+ is decreased by a constant ϵ, a parameter that controls the number of solutions that will be returned. Finally, the lower bound ρ_- for the next solution is updated in such a way that it is equal to the ρ value of the last solution found.

Theorem: The method generates a set of weakly non-dominated solutions. For each solution, only R can be increased and no more than ϵ.

Proof (weakly non-dominated): Each solution is obtained with the method used to optimize the value of ρ that produces optimal value of ρ for a given R, and ρ is always decreasing if R increases.

Proof (no more increase than ϵ): Let us consider a solution S_1 returned by the method with a temporal stability radius ρ_1 and a spatial stability radius R_1. If S_1 is the first solution found, with R_1 the maximal value of R, clearly there is no solution s_2 with a radius $R_2 > R_1$. Otherwise, the method has found a solution s_2 with a spatial radius $R_2 = R_1 + \epsilon$ and an optimal temporal radius $\rho_2 \leq \rho_1$ for this given value of R. Since S_1 has been returned (*i.e.*, non dominated by any solution found yet), we know that $\rho_2 < \rho_1$. We conclude that there is no solution s_3 with a radius $R_3 > R_1 + \epsilon$ and $\rho_3 = \rho_1$.

6.4. Complexity analysis

In this section, we study the complexity of our methods. First, it is important to determine the complexity of the discretization. The discretization algorithms are polynomial in the number of faces, ticks and time windows. Fortunately, these numbers are bounded by the same polynomials as for the temporal problem presented in [8, 14]. Thus, the discretization phase has a pseudo-polynomial complexity.

For the spatial maximization complexity, we need to determine how many linear programs are solved in the dichotomies. The first dichotomy is performed on a discrete set of values. The size of this set in the worst case is equal to the number of intersections between the ticks plus the number of ticks (they can all possibly be added). There are at most $2qm$ ticks, with m the number of sensors and q the number of line segments in the estimated trajectories. The second dichotomy stops when the interval width is smaller than λ. The width of this interval is divided by 2 at each iteration, where a linear program is solved. In the worst case, the initial interval is $[a, b]$ such that $a = 0$ and b is

the greatest value of the set used in the first dichotomy, *i.e.*, b is the upper bound on R. This upper bound is guaranteed to be bounded by R_s. Indeed, greater values correspond to empty sets of candidate sensors since the uncertainty disc to cover at any instant t is larger than the range of any sensor. Therefore, in the worst case, the dichotomy iterates until $\frac{R_s}{2^\gamma} < \lambda$ where γ the number of iterations. Hence, at most $\left\lceil \lg(\frac{R_s}{\lambda}) \right\rceil$ linear programs are solved in the worst case in this dichotomy.

The time complexity of the spatio-temporal method is the following. The complexity for the computation of the upper bounds R_{ub} and ρ_{ub} is the addition of the complexity of the method to maximize ρ (given in [8]) and the method for R presented in Section 6.1. The number of generated solutions is the integer value $\left\lceil \frac{R_s}{\epsilon} \right\rceil$, since R_s is the largest possible value of R of a feasible solution. For each solution, the method maximizes ρ as shown in [8], where linear programs are solved for different values of ρ taken in a discrete set of $(2qm)^2$ values (possible intersections between ticks). Therefore, the number of solved linear programs is at most $\left\lceil \frac{R_s}{\epsilon} \right\rceil \times \left\lceil \lg\left((2qm)^2\right) \right\rceil$.

7. Numerical experiments

In this section, we present several experiments. The objective is to study the effect of several parameters on the running time of both the spatial robustness problem and the spatio-temporal problem. Indeed, the complexity of the problem is heavily depending on the number of sensors, targets, desired precision (both ϵ and λ), etc. The running time of our method is expected to be long when the complexity increases and we want to study the scalability of the method. We would like to know when the running time is still acceptable. We also study the efficiency of the method and more particularly the quality of the solutions returned for different values of ϵ. Indeed, when ϵ decreases, the running time increases significantly but more solutions are returned, and a close approximation of the Pareto front is found. Therefore, with these experiments, we study the potential trade-off between the running time and the quality of the solutions.

The experiments aim to answer the following research questions:

- **RQ 1**: How fast is the method with different numbers of sensors?
- **RQ 2**: How fast is the method with different numbers of targets?
- **RQ 3**: How good are the solutions produced with different values of ϵ?

To achieve this, we resort to an instance generator presented in the next subsection. For each experimentation, the generator produces a set of 50 instances with varying parameters. The average CPU running times of corresponding instances are reported in this section, always expressed in seconds. The method is implemented in C++, with IBM CPLEX 12.7 [1] and all the experiments were run on a computer with Ubuntu 20.04 powered by an Intel Core i7-10850H CPU processor at 2.70 GHz × 12 cores and 16 GBytes of RAM. The method is called with $\lambda = 0.0001$ and $\epsilon = 1$. This default value of λ produces solutions that are considered of acceptable quality and as it will be show in the experiments, the value of ϵ achieves a good trade-off between quality and running time.

7.1. Instance generator

The experiments presented in the next sections are run on instances generated with the method presented here. Each instance is generated randomly with different parameters as inputs. These parameters and their default values are the following:

- The number of sensors m is set to 100 by default
- The number of targets n is set to 2 by default
- The interval from which a random level of battery is picked for each sensor is $[200, 300]$

All the other parameters (power consumption, sensing and communication ranges, etc) are the same for all our instances and are reported in Table 5. The generator starts by randomly deploying the sensors in the 300×300 area. The estimated trajectories of the targets are also randomly drawn in the zone with a straight trajectory such that it starts at instant $t = 0$ and finishes at time $t = 1000$. Finally, the coordinates of the base station are randomly picked inside the range of a randomly selected sensor. β is equal to 1 in all our experiments.

Parameter	Value
Size of the area	300×300
$p^S = p^R = p^T$	1
Sensing range R_s	50
Communication range R_c	100
β	1
Parameter	Default value
n	2
m	100
Energy of the sensors	$[200, 300]$

Table 5: Value of the parameters in the instance generator

7.2. Impact of the number of sensors on execution time

In this experiment, we generate 50 instances for different numbers of sensors. The number of sensors is well known to impact the complexity of the problem and thus is expected to heavily impact the running time of our method. The different numbers of sensors tested are $\{100, 150, 200, 400\}$. These values represent realistic ($\{100, 150, 200\}$) and difficult ($\{400\}$) instances with dense and challenging networks. Lower values are not tested since they may generate infeasible instances due to network disconnectivity. The other parameters for instance generation are the default values given in Table 5. The average results are reported in Table 6 with the running time of the discretization process and the number of returned solutions. We also report the running time only for the spatial optimization and the running time for the entire spatio-temporal optimization, both excluding the discretization. Note that therefore,

#Sensors	Discretization time (s)	Spatial time (s)	Spatio-temporal time (s)	#Solutions
100	0.004	0.054	0.396	10.26
200	0.036	0.617	10.138	21.60
300	0.153	1.779	45.492	29.00
400	0.487	3.579	104.053	32.98

Table 6: Impact of the number of sensors

the overall process' running time for one instance is the addition of its discretization's time and its spatio-temporal's time.

As expected, the discretization process needs more running time when the number of sensors increases. However, it remains low even with 400 sensors. For both the spatial robustness and the spatio-temporal robustness, the running time increases quickly with the number of sensors. However, in both cases it always stays acceptable since in average, in the worst case, the running time is less than two minutes. Increasing the number of sensors also increases the number of generated solutions, which means that the optimal value of R increases with the number of sensors since it offers more opportunity to cover the targets.

7.3. Impact of the number of targets on execution time

We now study how the running time is scaling with the number of targets. For that purpose, we generate 50 instances for different number of targets $n \in \{1, 2, 3, 4, 5\}$. The other parameter for instance generation are the default values given in Table 5. For each tested value, we report in Table 7 the average running time of both the spatial method and the spatio-temporal method, and the average running time of the dicretization process.

#Targets	Discretization time (s)	Spatial time (s)	Spatio-temporal time (s)	#Solutions
1	0.003	0.06	0.46	16.62
2	0.003	0.05	0.40	10.26
3	0.004	0.04	0.18	2.78
4	0.006	0.03	0.07	0.58
5	0.007	0.02	0.03	0.06

Table 7: Impact of the number of targets

First of all, the number of generated solutions decreases very fast with the number of targets. Indeed, the value of R decreases when the number of targets to monitor increases, and since the number of generated solutions directly depends on R, there are less solutions when the number of targets increases. With 5 targets, there are only a few generated solutions and a lot of instances are infeasible. Actually, there is only one feasible instance with three returned solutions..

0nly the running time of the discretization process increases with the number of targets, whereas the maximization of ρ and R is achieved faster with more targets in this experiment. At first sight, the fact that the running time decreases when the number of targets increases may sound counter-intuitive. Indeed, the problem becomes more

complex (more decision variables and more constraints) when targets are added. However, this evolution can be explained by the lower number of generated solutions. For example, with 1 target, in average the spatio-temporal method requires approximately 0.03s of CPU time per solution. With 5 targets, it is around 0.63s per solution. This behavior is seen for each tested value, *i.e.*, with more targets, the method always needs more time to find a solution, and the increase is significant. The average running time per solution for the overall process is presented in Table 8.

#Targets	Running time per solution (s)
1	0.03
2	0.04
3	0.06
4	0.13
5	0.63

Table 8: Average running time required by the overall method to generate a single solution

Additionally, we study the solution process with high number of targets. However, we also vary the available energy in the batteries of the sensors, in order to avoid having numerous unfeasible instances. The considered numbers of targets are $n \in \{15, 75, 100, 150, 200\}$, with respectively $\{[6000, 8000], [30000, 40000], [40000, 53333], [60000, 80000], [80000, 106666]\}$ energy available for the sensors. *i.e.*, with 15 targets, the energy available for each sensor in randomly picked between 6000 and 8000, with 75 it is between 30000 and 40000 and so on. For each tested value, a new set of 50 instances is generated. The objective here is to assess the scalability of our method on instances with a large number of targets when the number of feasible solutions is not too low. The average running times are presented in Table 9, similarly to the previous tables. Additionally, we report the average number of models solved.

#Targets	Discretization time (s)	Spatial time (s)	Spatio-temporal time (s)	#Model solved
15	0.05	0.08	2.67	128.5
75	1.31	0.23	8.44	75.98
100	2.33	0.30	11.92	75.62
150	5.45	0.45	19.12	68.24
200	10.74	0.61	27.92	70.42

Table 9: Impact of a very large number of targets

As can be seen, each part of the overall process needs more time when the number of targets increases. It is an important increase and with 200 targets, the running time of the overall process is around 40 seconds per instances. This increase affects the discretization process, which is very sensitive to the number of targets. The method proposed also needs more time to optimize the stability radii. However, as we can see in this experiment, the number of linear programs solved is not increasing, and thus the increase on the running time of the spatio-temporal method is only explained by the complexity of these linear programs.

Despite these increases, the overall running time stays acceptable and we conclude that the method is scaling well with the number of targets. From these two first ex-

periments, we deduce that the number of sensors is a more critical parameter than the number of targets.

7.4. Impact of ϵ on solution quality and execution time

In this experiment, we study the impact of ϵ on both the running time of our method and on the quality of the set of returned solutions. In order to evaluate this quality, we use the hypervolume metric [4]. It is one of the most popular metrics to assess the quality of the sets of solutions with a multi-objective problem. It computes the hypervolume between a reference point and the set of returned solutions. The higher the hypervolume, the better it is. The hypervolume is only used to assess the quality of the approximation of the Pareto front returned and is not considered in the running time. It is computed on the set of returned solutions using Pygmo [5], a library of Python programming language. The different values tested for ϵ are $\{10, 5, 1, 0.1, 0.01\}$. The first values ($\{10, 5\}$) are expected to generate only a few solutions with poor hypervolumes, but they are quickly generated. By contrast, the smallest values ($\{0.1, 0.01\}$) produce a lot of good solutions but are expensive to compute. We generate a set of 10 instances and apply our method with each value of ϵ on each instance. The results are reported in Table 10 with the running time of the overall method in seconds, the number of linear programs solved, the number of weakly non-dominated solutions returned and the hypervolume. All the values returned are average values over the 50 instances.

ϵ	Spatio-temporal (s)	#Models solved	#Returned solutions	Hypervolume
10	0.15	46.6	2.7	311.69
5	0.18	56.6	4.0	591.16
1	0.40	138.3	14.7	810.41
0.1	2.84	1044.8	133.1	859.52
0.01	27.35	10106.5	1317.8	864.42

Table 10: The method with different values of ϵ

As expected, the running time is heavily impacted by the value of ϵ. Indeed, the method always needs more time with smaller ϵ values. The running times are small with the first three values ($\{10, 5, 1\}$) but greater with $\epsilon = 0.01$. It can be easily explained by the number of returned solutions, and thus the number of models solved, which depends on the value of ϵ. However, the running time remains acceptable even with small ϵ values, since in that case, the instances are solved in about one minute.

In the case where $\epsilon = 0.01$, there are approximately 1318 returned solutions in average, and these solutions are all really close from the Pareto front (only R can be increased, and by no more than 0.01). With such a value, we can expect the set of solutions to produce a very good hypervolume since it approaches closely the Pareto front. This value is used as a reference value for the hypervolume and to study the efficiency of the sets of solutions returned by the other values of ϵ. With $\epsilon \in \{1, 0.1\}$, the hypervolumes are really close from the reference value. It means that the returned solutions are evenly spread and are likely to be on the Pareto front. However, these solutions are obtained way faster than with $\epsilon = 0.01$. The hypervolumes returned with $\{10, 5\}$ are much worst and this can easily be explained by the low number of returned solution, which is only 2.7 and 4.0 in average.

To conclude on this experiment, the value of ϵ is very important and defines a trade-off between the quality of the solutions and the computational effort. The value $\epsilon = 1$ seems to be the best trade-off since it produces a good hypervolume, that is not increasing much with smaller ϵ, and requires a modest running time.

8. Conclusion

This work proposes a novel formulation of the target tracking problem under spatio-temporal deviations. It extends the previous work on temporal uncertainties ([8]) by adding the spatial uncertainties. The problem is therefore to find robust schedules against both types of uncertainty, using two stability radii. The proposed method to solve this problem generates a set of solutions that approximate the Pareto front. This method has been tested through extensive experimentation, testing its scalability against an increase of the number of sensors and the number of targets. The experimentation concluded that the running times stay relatively low for all the tested values. Moreover, an additional experiment has been performed to investigate the impact of the parameter ϵ which controls the number of generated solutions, their quality and the computational effort. It shows that good sets of solutions may be returned when ϵ is around 1, as this setting keeps the running time reasonably low with acceptable solution qualities compared to what is obtained with lower ϵ values.

References

[1] Cplex optimiser. `https://www.ibm.com/analytics/cplex-optimizer`. Accessed: 13-07-2020.

[2] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communications magazine*, 40(8):102–114, 2002.

[3] Luca Benini, Giuliano Castelli, Alberto Macii, Enrico Macii, Massimo Poncino, and Riccardo Scarsi. A discrete-time battery model for high-level power estimation. In *Proceedings of the conference on Design, automation and test in Europe*, pages 35–41. ACM, 2000.

[4] Nicola Beume, Carlos M Fonseca, Manuel López-Ibáñez, Luís Paquete, and Jan Vahrenhold. On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, 13(5):1075–1082, 2009.

[5] Francesco Biscani and Dario Izzo. A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software*, 5(53):2338, 2020. doi: 10.21105/joss.02338. URL `https://doi.org/10.21105/joss.02338`.

[6] Fabian Castaño, Éric Bourreau, André Rossi, Marc Sevaux, and Nubia Velasco. Partial target coverage to extend the lifetime in wireless multi-role sensor networks. *Networks*, 68(1):34–53, 2016. doi: 10.1002/net.21682. URL `https://hal.archives-ouvertes.fr/hal-01328424`.

[7] Qi Chu, Wanli Ouyang, Hongsheng Li, Xiaogang Wang, Bin Liu, and Nenghai Yu. Online multi-object tracking using cnn-based single object tracker with spatial-temporal attention mechanism. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4836–4845, 2017.

[8] Florian Delavernhe, Charly Lersteau, André Rossi, and Marc Sevaux. Robust scheduling for target tracking using wireless sensor networks. *Computers & Operations Research*, page 104873, 2020.

[9] Ralf Hartmut Güting and Markus Schneider. *Moving objects databases*. Elsevier, 2005.

[10] Kathleen Hornsby and Max J Egenhofer. Modeling moving objects over multiple granularities. *Annals of Mathematics and Artificial Intelligence*, 36(1-2):177–194, 2002.

[11] Hoyoung Jeung, Qing Liu, Heng Tao Shen, and Xiaofang Zhou. A hybrid prediction model for moving objects. In *2008 IEEE 24th international conference on data engineering*, pages 70–79. IEEE, 2008.

[12] Cheng-Hao Kuo, Chang Huang, and Ramakant Nevatia. Multi-target tracking by on-line learned discriminative appearance models. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 685–692. IEEE, 2010.

[13] Marco Laumanns, Lothar Thiele, and Eckart Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3):932–942, 2006.

[14] Charly Lersteau, André Rossi, and Marc Sevaux. Robust scheduling of wireless sensor networks for target tracking under uncertainty. *European Journal of Operational Research*, 252(2):407–417, 2016.

[15] Haitao Lin and Xiaopeng Yang. Dichotomy algorithm for solving weighted min-max programming problem with addition-min fuzzy relation inequalities constraint. *Computers & Industrial Engineering*, 146:106537, 2020.

[16] Yuzhen Liu and Weifa Liang. Approximate coverage in wireless sensor networks. In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pages 68–75. IEEE, 2005.

[17] Yuri N Sotskov, Vyacheslav S Tanaev, and Frank Werner. Stability radius of an optimal schedule: A survey and recent developments. In *Industrial applications of combinatorial optimization*, pages 72–108. Springer, 1998.

[18] Yufei Tao, Christos Faloutsos, Dimitris Papadias, and Bin Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 611–622, 2004.

[19] Goce Trajcevski. Uncertainty in spatial trajectories. In *Computing with Spatial Trajectories*, pages 63–107. Springer, 2011.

[20] Goce Trajcevski, Ouri Wolfson, Klaus Hinrichs, and Sam Chamberlain. Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems (TODS)*, 29 (3):463–507, 2004.

[21] Milica Pejanović Đurišić, Zhilbert Tafa, Goran Dimić, and Veljko Milutinović. A survey of military applications of wireless sensor networks. In *Embedded Computing (MECO), 2012 Mediterranean Conference on*, pages 196–199. IEEE, 2012.

[22] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer networks*, 52(12):2292–2330, 2008.

[23] Qian Yu, Gérard Medioni, and Isaac Cohen. Multiple target tracking using spatio-temporal markov chain monte carlo data association. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.