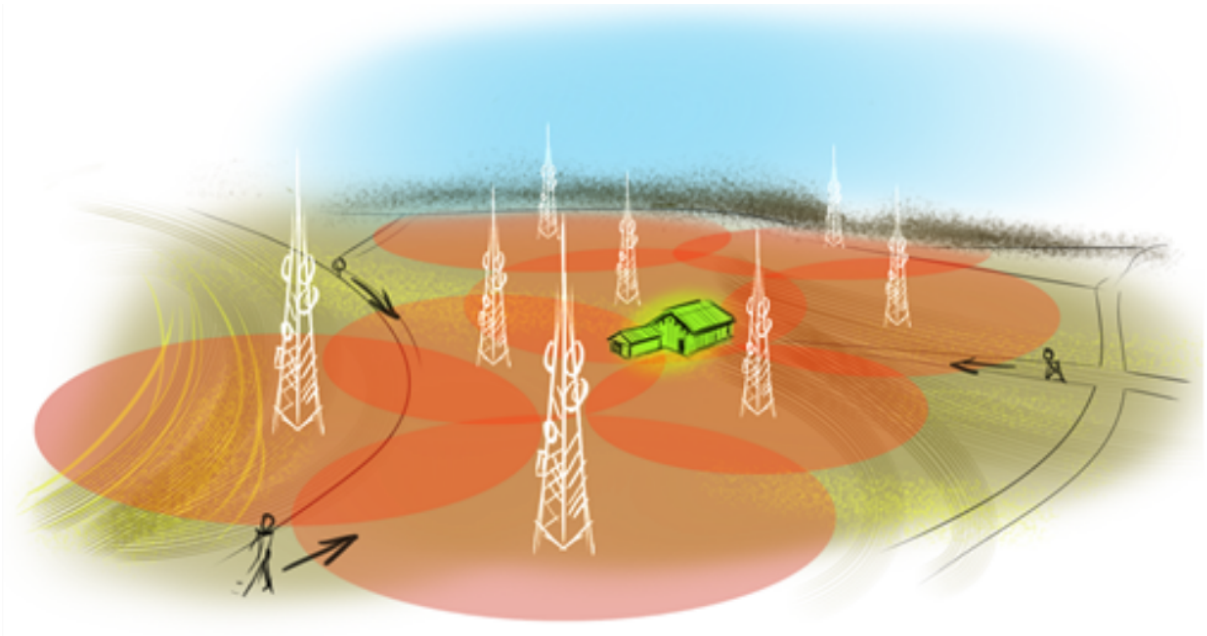
Challenge 2: Escape the Sensor

Mathematical Challenges in the Electromagnetic Environment

Report January 27th, 2021

University of Cambridge, Newton Gateway to Mathematics
A PA Consulting & DSTL Project



Report Authors and Contributors

Anvarbek Atayev (University of Warwick, anvar.atayev@warwick.ac.uk)
Connor Delaosa (University of Strathclyde, connor.delaosa@strath.ac.uk)

Contributors

John Beasley (Brunel University, john.beasley@brunel.ac.uk)
Joerg Fliege (University of Southampton, j.fliege@soton.ac.uk)
Kurt Langfeld (University of Leeds, k.langfeld@leeds.ac.uk)
Bill Lionheart (University of Manchester, bill.lionheart@manchester.ac.uk)
William Liles (lilesw@gmail.com)
Hamza Alawiye (University of Bristol, h.alawiye@bristol.ac.uk)
Jaroslav Fowkes (STFC RAL, jaroslav.fowkes@stfc.ac.uk)

Challenge Owners

Emily Russell (DSTL, erussell@dstl.gov.uk)
Olly Gage (DSTL, ogage@dstl.gov.uk)
Richard Claridge (PA Consulting, richard.claridge@paconsulting.com)

Contents

Chapter 1

Introduction

In many real world situations, to restrict access to sites of particular importance, a series of sensors of various types (be it motion detectors, cameras or personnel) are used to surround the site to detect incoming devices or personnel. One can reverse this scenario and instead be faced with a situation where they wish to gain access to a site, but encounter sensors which they must bypass without being detected. This was the problem that was posed by DSTL and PA Consulting at the Mathematical Challenges in the Electromagnetic Environment Study Group hosted by Newton Gateway held between January 6-8th 2021.

1.1 Problem Description

Consider a network of sensors surrounding a site of interest, called the *target*, and a collection of *agents* which wish to reach said target (of which only one is required to reach the target).

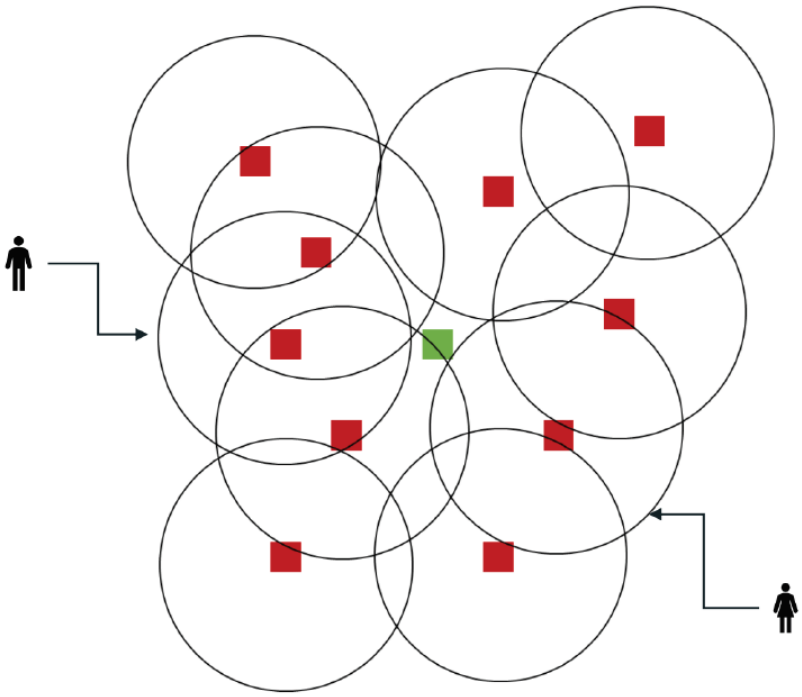


Figure 1.1: Depiction of the Escape the Sensor problem. Red squares denote the sensor positions (where the circles denote the sensor's detection range), green square denotes the objective position, and persons denote agent positions.

Assume that each sensor has a range and probability of detection and the identification of the agent is contingent on a sufficient level of detection by one or more sensors. Furthermore, assume that agents are able to perform disabling/disrupting interventions to sensors along their path to the target, whereby an agent is able to disrupt or disable all sensors within a given range for a fixed period of time, and performing such an intervention incurs an equipment and time cost to the agent concerned.

Through the use of the study group, DSTL and PA Consulting looked to answer for of the following questions:

1. What is the optimal path, which minimises the equipment cost, time and probability of detection, that an agent can take to a target (in collaboration with other agents on the field) given (a) full knowledge of sensor arrangement (b) partial/no knowledge of sensor arrangement?

2. How is the optimal path affected to changes in the objective/cost function?

3. How would a solution approach change for a 2D scenario, a fully 3D scenario and a partial 3D scenario accounting for topology?

4. Does the route optimisation need to be dynamic (what gains can be made versus an initial case?)

1.2 Report Structure

Throughout this study group, many mathematical avenues were considered. Two distinct routes were apparent, an approach via continuous and discrete space models, both of which have been outlined in this document. In Part I, the continuum approach to this problem is described: Chapter 2 outlines the general continuum model used and Chapter 3 and 4 outline the results of computational experiments carried out using the described model. In Part II, the discrete approach to this model is described and can be treated completely separately to the discussion in Part I. Chapter 5 discusses the theory of the discrete model, the contents of which are expanded upon and experimented with in Chapter 6. The backmatter of the document contains a number of appendices, providing further discussions on the theory and code listings for the computational studies described in the document.

Part I

Continuum Approach

Chapter 2

Continuum Model

In this chapter, we outline a continuum mathematical model that will be used to obtain a solution to the optimal path problem described in Chapter 1.

Definitions

Throughout this part, we assume that the working domain is two dimensional, which corresponds to all objects of interest lying on a flat plane, although it is thought that an extension to a three dimensional model, where landscape and physical obstacles are considered, is relatively straightforward.[1]

- Denote by $X \in \mathbb{R}^2$ the location of the target;
- Denote by $x_i \in \mathbb{R}^2$ for $i = 1, \dots, N$ the locations of a collection of N sensors;
- Corresponding with each sensor i an effective *agent registration* probability function given by $p_i : \mathbb{R}^2 \to [0, 1]$;
- Denote by $P : \mathbb{R}^2 \to [0, 1]$ the detection probability for an agent, i.e. $P(z)$ denotes the detection probability of an agent at location $z \in \mathbb{R}^2$. The definition of P will be dependent on sensor registration probabilities $(p_i)_{i=1}^N$ and their locations $(x_i)_{i=1}^N$. Although, the precise definition of P is strictly dependent on detection criteria, e.g. if one or more registrations on sensors are required for an agent to be detected.[2] This can be chosen freely by the stakeholder and the proceeding implementation is expected to hold for a large class of choices.

2.1 Detection Probability

For the sake of experimentation, assume that at least two sensors must register an agent for a detection event to occur. Then, introducing a binary detection variable defined as

$$k_l := \begin{cases} 1 & \text{sensor } l \text{ detects agent,} \\ 0 & \text{otherwise,} \end{cases} \tag{2.1}$$

[1] One way to do this would be to alter the associated cost function, used to obtain an optimal path, to account for such features. For example by making it more *expensive* to have paths that go up steep hills, or *infinitely expensive* to have paths that go through obstacles.

[2] One can imagine that if a sensor is a camera, then a visual registration on one device may be sufficient for an agent to be detected. Although if you have a motion sensor, to reduce the likelihood of false-positives, sensors may only detect a presence on an agent if registrations on at least two devices are made.

the probability of detection of an agent at position $z \in \mathbb{R}^2$ can be assumed to be

$$P(z) = \sum_{\{k_1,\dots,k_N\}} \prod_{i=1}^{N} (p_i(z))^{k_i} (1 - p_i(z))^{1-k_i}, \tag{2.2}$$

where the sum is over all sets $\{k_1, \dots, k_N\}$ with at least two 1's (i.e. with at least two sensors registering the agent's position). Without loss of generality, if one assumes that only sensors i_1, i_2 register an agent at position z, we find that

$$P(z) = p_{i_1}(z) p_{i_2}(z) \prod_{\substack{j=1 \\ j \neq i_1, i_2}}^{N} (1 - p_j(z)). \tag{2.3}$$

If we further assume that the detection probabilities are rapidly decaying with respect to their position, we find that, for sufficiently well spaced sensors, the dominant contributions to $P(z)$ are

$$P(z) = p_{i_1}(z) p_{i_2}(z) + \mathcal{O}\left(\max_{\substack{j=1,\dots,N \\ j \neq i_1, i_2}} p_j(z) \right) \tag{2.4}$$

as $\min\{|z - x_{i_1}|, |z - x_{i_2}|\} \to 0$. Therefore, for brevity, one can assume that the contribution to the detection probability is dominated by the contributions from the two sensors i_1, i_2 closest to an agents position z. Thus define the *approximated detection probability* of an agent at position $z \in \mathbb{R}^2$ by

$$P_{\text{approx}}(z) = p_{i_1}(z) p_{i_2}(z), \tag{2.5}$$

where sensors i_1, i_2 are the ones closes to z, i.e. $|x_{i_1} - z|, |x_{i_2} - z| < |x_j - z|$ for $j = 1, \dots, N$ such that $j \neq i_1, i_2$. In the event that an agent z is equidistant to more than two sensors, for simplicity, choose at random any two closest to z.

Remark 2.1.1. The approximation in Equation 2.5 yields an over approximation to the probability of detection since any further contribution to the product from other sensors will either keep the same or decrease the probability. Therefore the approximation does not hinder the aim of the challenge. Indeed, since for $j = 1, \dots, N$, $p_j(z) \leq 1$, and thus $0 \leq 1 - p_j(z) \leq 1$ for all $z \in \mathbb{R}^2$, it follows that

$$0 \leq \prod_{\substack{j=1 \\ j \neq i_1, i_2}}^{N} (1 - p_j(z)) \leq 1, \tag{2.6}$$

hence for all $z \in \mathbb{R}^2$

$$0 < P(z) = p_{i_1}(z) p_{i_2}(z) \underbrace{\prod_{\substack{j=1 \\ j \neq i_1, i_2}}^{N} (1 - p_j(z))}_{\leq 1} \leq p_{i_1}(z) p_{i_2}(z) = P_{\text{approx}}(z). \tag{2.7}$$

For the remainder of this part (Cotinuum Approach) of the report we shall use P_{approx} as our probability of detection at a point. For brevity, we lose the subscript and denote by P the approximated probability of detection at a point.

2.2 Objective/Cost Function

Given a target location at $X \in \mathbb{R}^2$ and an initial position $A \in \mathbb{R}^2$, the aim of this challenge is to obtain an optimal path from a chosen initial position to the target position whilst minimising together the total probability of detection, the cost of equipment usage and the time of flight. Although, the main priority is minimise the probability of detection along the path, and thus we shall construct our cost function according to this. Mathematically, we aim to determine an optimal path $\gamma : [0, T] \to \mathbb{R}^2$ such that $\gamma(0) = A$ and $\gamma(T) = X$, where $T > 0$ is the time that is taken for the agent to reach the target. Now, there are two approaches that we can take here: we can assume that T is unknown, or select a value of T from which we would like an optimal path. Clearly, if one is allowed to choose T, then depending on velocity constraints, this choice may result in a path that is unnecessarily long, or too short whilst making the velocity too large. For the current moment, assume that T is unknown.

Define by $\mathcal{A}$ the space of all paths that go from A to X, i.e.

$$\mathcal{A} := \left\{ \gamma : [0, T] \to \mathbb{R}^2 \;\middle|\; \gamma(0) = A,\ \gamma(T) = X,\ T > 0 \right\}. \tag{2.8}$$

Then, given the aim, the next step is to construct an objective/cost function, F, dependent on the parameters outlined in the introduction to obtain an optimal path from A to X.

2.2.1 Construction of Cost Function

Let $\gamma \in \mathcal{A}$ denote an admissible path and assume a time of flight of $T > 0$. The obvious choice of cost function is the cumulative probability of detection along the path γ with the assumption that detection is also proportional to the amount of time spent at a certain position, a minimisation of which would yield a path from an initial position to the target that minimises the total probability of detection.

Let $t \in [0, T)$ and let $\delta t > 0$ be a small time parameter. Then between t and $t + \delta t$, the agent moves from position $\gamma(t)$ to $\gamma(t + \delta t)$. Assuming δt is small enough, we can approximate the probability of detection from moving from $\gamma(t)$ to $\gamma(t + \delta t)$ to be $P(\gamma(t))\delta t$. Now, split $[0, T)$ into M uniformly long time intervals $[t_{i-1}, t_i]$, with $t_i = \frac{iT}{M}$ for $i = 0, \ldots, M$ and choose $\delta t := t_{i+1} - t_i = \frac{T}{M}$. Then the cumulative probability of detection along the whole journey (approximated by the time steps (t_i)) is

$$\sum_{i=1}^{M} P(\gamma(t_i))\delta t. \tag{2.9}$$

To obtain the continuous counterpart so that we are able to perform calculus on it, we take $M \to \infty$ (to consider a model with smaller and smaller time step intervals) and obtain that the cumulative probability of detection along the whole journey in the limit as $M \to \infty$ converges to

$$\sum_{i=1}^{M} P(\gamma(t_i))\delta t \xrightarrow{M\to\infty} \int_0^T P(\gamma(t))\ \mathrm{d}t. \tag{2.10}$$

Therefore, the cumulative probability of detection over the whole journey is given by

$$F_{\text{proto}}[\gamma; T] = \int_0^T P(\gamma(t))\ \mathrm{d}t. \tag{2.11}$$

Initially one may think that F_{proto} can be used as the cost function. Although, a quick analysis shows that this cost function is ill-defined since F_{proto} can be minimised (to a value of 0) with

respect to $\gamma \in \mathcal{A}$ by a straight line from A to X which moves infinitely fast. Clearly this is not feasible, and thus, when minimising the cost function, we must avoid solutions that break the rules of reality/that are unreasonable. Thus, in the cost function, we must penalise velocity if it is too high. We do this by adding to F_{proto} a velocity term which grows as the speed of traversal increases (as to make the cost of very fast traversals expensive, and thus not a candidate for a minimiser). This can be done by adding to F_{proto} the *average* velocity of traversal $\int_0^T |\dot{\gamma}(t)|^2 \, \mathrm{d}t$ where $\dot{\gamma}(t) = \frac{\mathrm{d}\gamma(t)}{\mathrm{d}t}$ denotes the velocity of the agent at time t. We can therefore construct an initial cost function to be defined as

$$F[\gamma; T] := \int_0^T P(\gamma(t)) \, \mathrm{d}t + \lambda_1 \int_0^T |\dot{\gamma}(t)|^2 \, \mathrm{d}t + \lambda_2 T, \tag{2.12}$$

where we have also included a time cost which aims to minimise the time of travel (assuming $\lambda_2 > 0$). The parameters λ_1, λ_2 have been introduced to allow for the stakeholder to vary the importance of certain parameters (e.g. traversal speed and time). The aim is now to minimise the cost function F with respect to paths $\gamma \in \mathcal{A}$ and a travel time $T > 0$, for some chosen $\lambda_1 > 0$ and $\lambda_2 \geq 0$.

Remark 2.2.1. The cost function defined in Equation 2.12 is not the only feasible choice and can be chosen according to stakeholder needs. Indeed, one can alternatively define the cost function F as a path integral from A to X, i.e.

$$F[\gamma, T] := \int_\gamma P(z) \, \mathrm{d}z = \int_0^T P(\gamma(t))|\dot{\gamma}(t)| \, \mathrm{d}t, \tag{2.13}$$

to which one can add additional cost parameters as needed.

Remark 2.2.2. One should note that the cost function described in Equation 2.12 does not yet account for the use of interventions. The effect of interventions and their associated cost is introduced in Chapters 3 and 4.

2.2.2 Technical Remark for Numerical Implementation

Note that the travel time T discussed above is a parameter subjected to the minimisation problem. It is convenient to remove this parameter from the integration boundaries of the cost function to allow for easier numerical implementation. This can be achieved by 3D implicit parametrisation as follows. Assume that $\gamma \in \mathcal{A}$ is a path and $t : [0, S] \to \mathbb{R}$ is a parametrisation of the time variable for some chosen $S > 0$. Then the path-time pair can be defined with respect to the new time parametrisation as follows: for $s \in [0, S]$, define $\gamma(t(s)), t(s)$ such that

$$\gamma(0) = A, \; t(0) = 0 \quad \text{and} \quad \gamma(t(s)) = X, \; t(S) = T. \tag{2.14}$$

For brevity, denote by $\gamma(s) \equiv \gamma(t(s))$. Then, the cost function for optimisation can then be restated as

$$F[\gamma; T] := \int_0^S P(\gamma(s)) \frac{\mathrm{d}t}{\mathrm{d}s}(s) \, \mathrm{d}s + \lambda_1 \int_0^S |\dot{\gamma}(s)|^2 \frac{\mathrm{d}t}{\mathrm{d}s}(s) \, \mathrm{d}s + \lambda_2 t(S). \tag{2.15}$$

This new formulation is now more suitable for numerical implementation, in particular, it is suitable for discretisation in the following sense: given a fixed $S > 0$, partition $[0, S]$ into N_S discretisation steps such as $[0, S] = \bigcup_{i=1}^{N_S} [s_{i-1}, s_i]$, where $s_0 = 0$ and $s_{N_S} = S$. Then one can define the kth discretisation of the path-time pair as

$$(\gamma, t)_k = (\gamma(s_k), t(s_k)), \quad k = 1, \ldots, N_S.$$

Without loss of generality, we can choose $S = N_S$ in which case $s_k = k$.

Remark 2.2.3. A simple choice of $t : [0, S] \to \mathbb{R}$ would be $t(s) = \frac{T}{S} s$, in which case

$$F[\gamma; T] := \frac{T}{S} \left(\int_0^S P(\gamma(s)) \, \mathrm{d}s + \lambda_1 \int_0^S |\dot{\gamma}(s)|^2 \, \mathrm{d}s + \lambda_2 S \right). \tag{2.16}$$

2.3 Sensor Detection Probability

The choice of sensor detection probability is left to the stakeholder. In reality one would expect the detection probabilities to be directional, decay with distance from the sensor, and vary for each sensor, accounting for the type of sensor used. Although, for experimentation purposes, one can safely assume that the detection probability is radial, all detection probabilities are equal and are of the form

$$p_i(z) := \frac{1}{1 + \left(\frac{|z - x_i|}{\delta}\right)^2}, \quad i = 1, \ldots, N \tag{2.17}$$

for some detection intensity $\delta > 0$. Other choices for experimentation can also be made, such as $p_i(z) = e^{-\left(\frac{|z-x_i|}{\delta}\right)^2}$.

2.4 Possible Numerical Approaches

There are numerous methods that can be used to minimise the cost function F. Given the time-limited nature of the study group, only three methods were investigated: a) A *simulated annealing* approach (see Chapter 3) for which a number of useful texts exist, e.g. Appendix A of [3] and references therein. b) A *nonlinear integer programming* approach (see Chapter 4). c) A brief look into *finite element method* (FEM) based solvers (see below).

FEM Solver

For the following, assume that T is a known parameter implicit in the choice of admissible path and $\lambda_2 = 0$ in the definition of cost function as given in Equation 2.12. The aim of the challenge is then to obtain an optimal path $\gamma \in \mathcal{A}$ from an initial position A to a target position X under a randomly generated (but assumed known) probability landscape. In particular, the aim is to obtain a $\Gamma \in \mathcal{A}$ such that

$$\Gamma = \arg\min_{\gamma \in \mathcal{A}} F[\gamma].$$

Letting $\mathcal{L}(t, \gamma(t), \dot{\gamma}(t)) := P(\gamma(t)) + \lambda_1 |\dot{\gamma}(t)|^2$, then

$$\Gamma = \arg\min_{\gamma \in \mathcal{A}} F[\gamma] = \arg\min_{\gamma \in \mathcal{A}} \int_0^T \mathcal{L}(t, \gamma(t), \dot{\gamma}(t)) \, \mathrm{d}t, \tag{2.18}$$

which is a standard optimisation problem found in the field of calculus of variations. This minimisation problem can often be solved using Euler-Lagrange equations, where determining the optimal path $\Gamma \in \mathcal{A}$ is equivalent to solving a PDE of the form

$$\frac{\partial \mathcal{L}}{\partial \Gamma_i}(t, \Gamma(t), \dot{\Gamma}(t)) - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial \mathcal{L}}{\partial \dot{\Gamma}_i}(t, \Gamma(t), \dot{\Gamma}(t)) = 0 \quad \text{for } i = 1, 2. \tag{2.19}$$

The weak form of this equation can be solved by the finite element method using any number of commercial or open source packages (e.g, FEniCS, Firedrake, FreeFEM). Note that the minima found using this method are only local minima. The nonlinearities present in the problem may mean that there are in fact many such local minima, but there exist methods such as deflation that allow us to systematically identify multiple solutions of nonlinear equations where they exist, see [4].

The formulation described by Equations 2.18 and 2.19 also lends itself to the case where we

might want to model the usage of disruptive equipment as a continuous function of time. In addition to a path γ, we would solve for another function $q : [0, T] \to \mathbb{R}$ and modify the cost function to include the disruptive effect $D(q)$:

$$F[\gamma] = \int_0^T D(q(t))P(\gamma(t)) \, \mathrm{d}t + \lambda_1 \int_0^T |\dot{\gamma(t)}|^2 \, \mathrm{d}t. \tag{2.20}$$

Perhaps D might take the form $D(q) = (1 - q)^2$ or something of a similar nature. Appropriate boundary conditions would also need to be considered. Furthermore, q would need to satisfy some sort of constraint such as

$$\int_0^T q(t)^2 \, \mathrm{d}t = Q, \tag{2.21}$$

for some known value Q to take into account limitations in equipment usage. This can be achieved through the use of Lagrange multipliers in the cost function. It is also possible to incorporate inequality constraints (e.g. $\int_0^T q(t)^2 \, \mathrm{d}t \leq Q$) in this framework to better model equipment usage constraints, though they make the problem more computationally difficult. A further benefit of this approach is that it is naturally suited to combination with continuation methods if one wishes to consider a range of parameters (e.g. $\lambda_1 \in [\lambda_1^-, \lambda_1^+]$) and see how the solution space evolves as the parameters change.

Chapter 3

Computational Study 1 - Simulated Annealing: Kurt Langfeld

In this chapter, we perform a computational experiment on the optimisation problem discussed in Chapter 2 and extend upon it to take into account the use of sensor disrupting interventions. In this chapter, the numerical method that is used to obtain optimal solutions to the cost functions is called simulated annealing, see Appendix A of [3] for an introduction. In Section 3.1, a path that minimises the cost function defined in Equation 2.12 is obtained. In addition, in Section 3.2, optimal path solutions are obtained for a cost function with additional constraints which were added to account interventions and their associated cost We note that the following experiment assumes that we have only one agent, and furthermore that similar experiments can be done for variations of the cost function, according to stakeholder needs or the scenario.

Let us assume that we have $N = 10$ sensors distributed on an integer grid $[0, 10]^2 \cap \mathbb{Z}^2$ with the objective assumed to be in position $X = (5, 5)$ and an initial position $A = (0, 0)$.[1] As shown in Figure 3.1a the positions of the sensors are at x_i, $i = 1 \dots N$. Assume furthermore that each sensor has a probability $p_i(z)$ to *register* an agent given by

$$p_i(z) = \frac{1}{1 + (r/\delta)^2}, \quad i = 1, \dots, N, \tag{3.1}$$

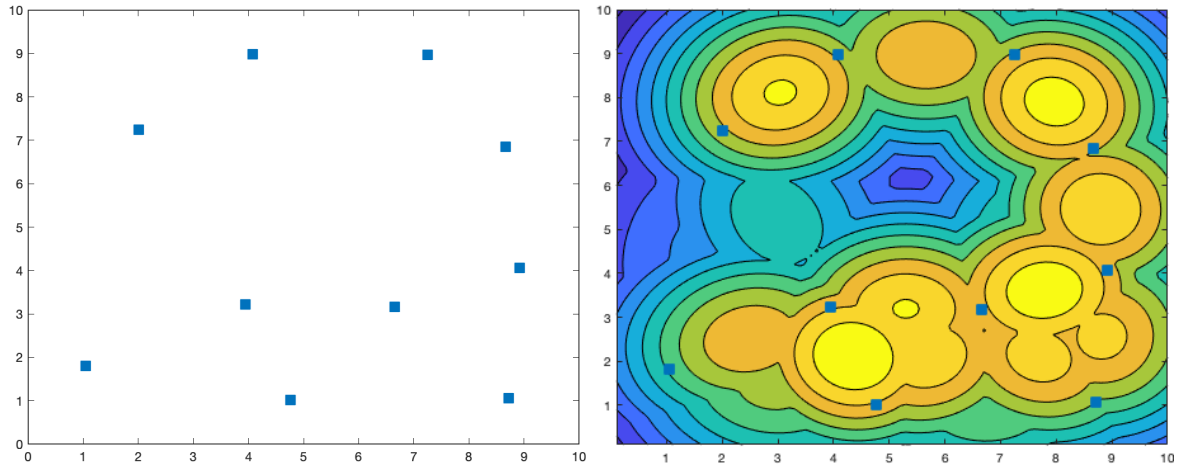
where $r = |z - x_i|$ and δ denotes the range for detection. Througout this section, we take $\delta = 4$, although this choice is up to the stakeholder.

As described in the Chapter 2, we approximate the probability of detection of an agent in $z \in [0, 10]^2$ by

$$P(z) = p_{i_1}(z)p_{i_2}(z), \tag{3.2}$$

where sensors i_1, i_2 are the closest to position z. For the described sensor configuration and $\delta = 4$, the probability of detection on $[0, 10]^2$ is depicted in Figure 3.1b. In this figure, the more *yellow* the contour, the higher the probability of detection. Given a probability of detection described by P, we expect that a path that minimises the probability of detection would avoid these *yellow* regions. Due to the sensor positions being chosen at random, for the given configuration, there is a large concentration of valleys in the contour plot which has a relatively low probability of detection. In reality, this would not be the design of a chosen system and would have a much more complicated array design but for the purposes of experimentation this is the toy problem we choose to simulate.

[1] The fact that the sensors, initial, and target positions are located on a grid is an experimental choice, chosen for simplicity. This choice does not hinder the analysis performed in this chapter, and indeed, any point in $\mathbb{R}^2$ can be used to represent initial, target and sensor positions.



(a) (b)

Figure 3.1: (a) Depiction of sensor positions at x_i, $i = 1, \dots, N$ with $N = 10$, represented by blue squares. (b) Contour plot of detection probability P, where blue represents a low probability and orange a high probability.

3.1 Single Agent - No Intervention and No Time Penalty

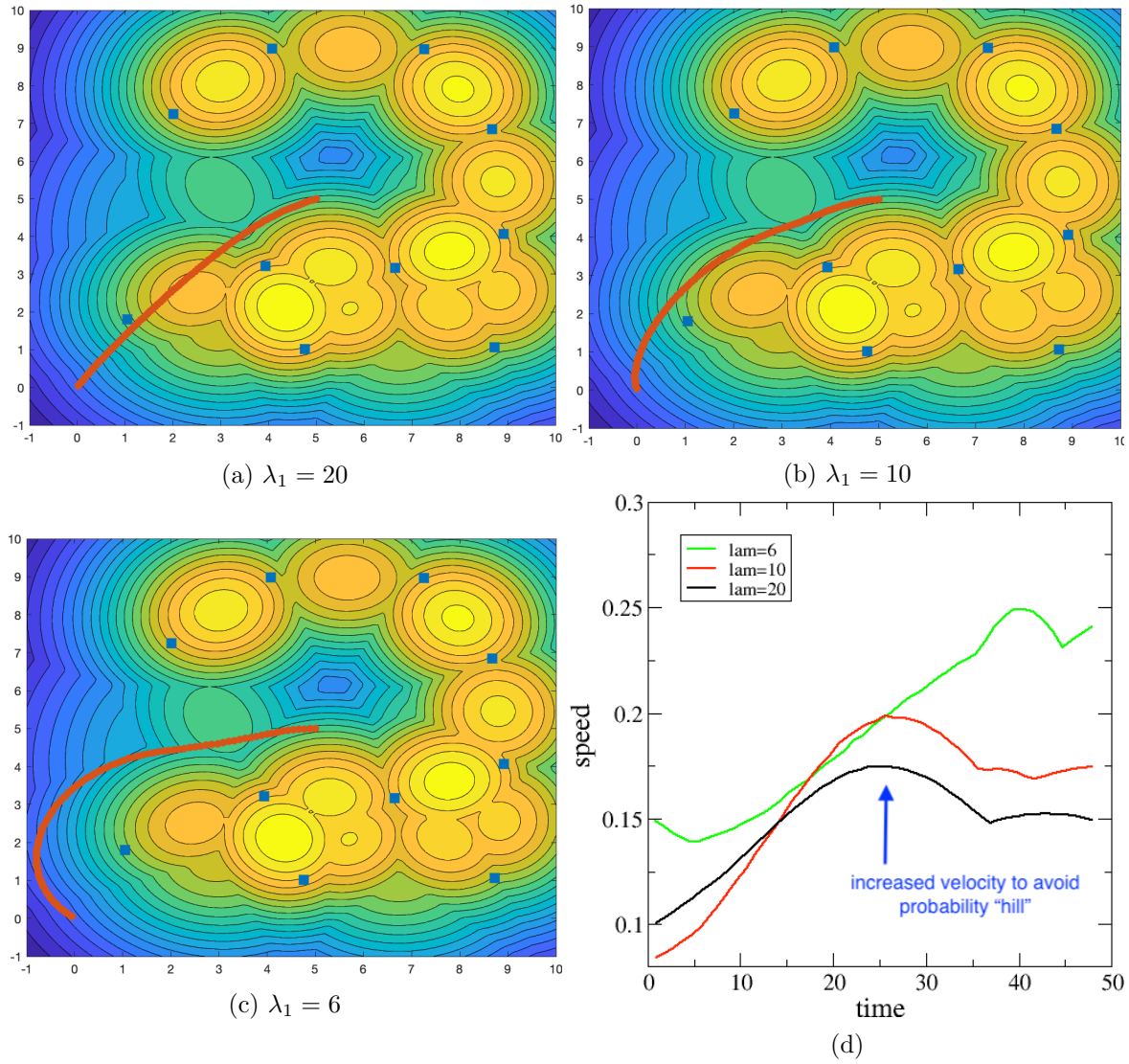
For the case of a single agent and no interventions to stop detection, we seek a path $\gamma(t) := (x(t), y(t))$ for $t \in [0, T]$ with time parameter t, that starts at the origin, i.e., $(x(0), y(0)) = (0, 0) =: A$ and that ends at the object of interest at some yet unknown time $T > 0$, i.e. $(x(T), y(T)) = (5, 5) =: X$.

By Equation 2.12, we define the cost function, when no interventions are used, by

$$F[\gamma; T] = \int_0^T P(\gamma(t)) \, \mathrm{d}t + \lambda_1 \int_0^T |\dot{\gamma}(t)|^2 \, \mathrm{d}t + \lambda_2 T. \tag{3.3}$$

where $\lambda_1 > 0$ denotes a penalty for velocity and $\lambda_2 \geq 0$ the penalty for time. We seek to minimise the cost function with respect to the path and travel time. To perform the necessary discretisation for numerical implementation, we follow the remark made in Subsection 2.2.2, where we denote by S the length of the time parametrisation as defined in Equation 2.14 and by N_S the number of discretisation steps. Throughout the following, to simplify the algebra, assume that $N_S = S$ and use $N_S = 50$. For the problem with no intervention, we study only $\lambda_2 = 0$ and do not optimise with respect to time and solve the minimisation problem using simulated annealing.

Large Velocity Cost ($\lambda_1 = 20$) Let us start with a large value for $\lambda_1 = 20$. Since λ_1 is large, the contributions to the cost function F due to cumulative probability of detection is expected to be small with respect to the velocity contribution, and thus matters much less. Therefore, under a large λ_1, we expect that the optimal path will largely ignore the necessity for minimising the probability of detection, but instead aim to minimise speed of traversal. Thus, we expect the solution to be an almost straight line that connects the initial position A with X, largely ignoring the *probability term* of the cost functional F. Indeed, from Figure 3.2a we can see that the optimal path is close to a straight line, but still the *high probability regions* are avoided to some extent. The black curve in 3.2d shows the speed of the agent as a function of time. At around $t = 25$, the agent reaches the *orange region* of moderately high detection probability, where we notice that the speed of the agent is higher to reduce the time spent in this region and thus the risk of detection.

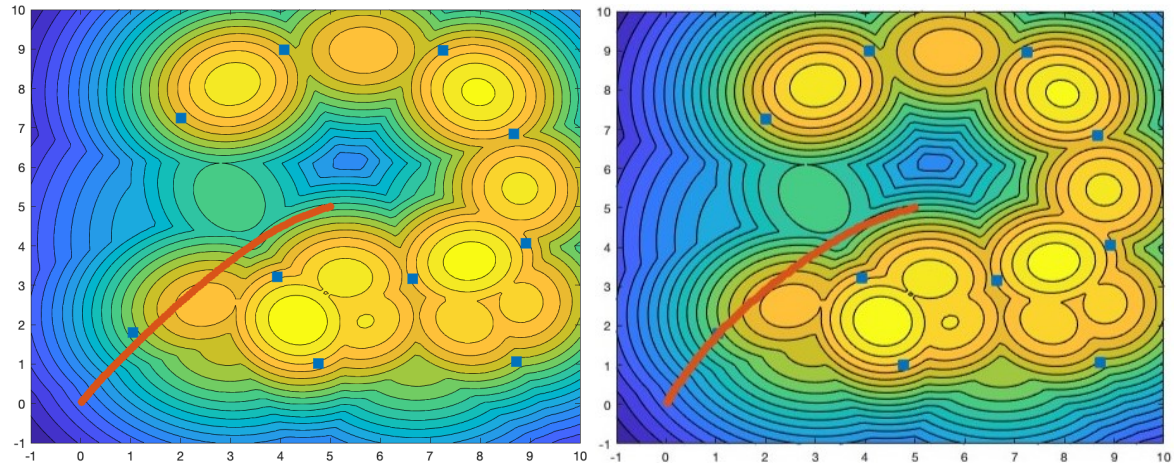


(a) $\lambda_1 = 20$ (b) $\lambda_1 = 10$ (c) $\lambda_1 = 6$ (d)

Figure 3.2: (a), (b), (c) 2D probability map showing optimum paths (in red) for various values of velocity penalty λ_1. (d) Speed-time plot of optimal paths obtained for various values of velocity penalty λ_1.

Low Velocity Cost ($\lambda_1 = 10, 6$): Now, we lower λ_1 to 10 and 6 as to reduce the effect of the velocity penalty in F and further involve the importance of the cumulative probability of detection. The corresponding speed functions are depicted in Figure 3.2d as red ($\lambda_1 = 10$) and green ($\lambda_1 = 6$) curves with their corresponding optimal paths given in Figure 3.2b and Figure 3.2c, respectively. While the speed for $\lambda_1 = 10$ is qualitatively the same (though at higher values), the speed curve for $\lambda_1 = 6$ is qualitatively different. When $\lambda_1 = 10$, the curve now has the liberty to avoid more of the high probability regions, and is depicted in Figure 3.2b. For $\lambda_1 = 6$, this avoidance is more extreme and the path trajectory now has a higher curvature, but also features a higher average speed to arrive at the target.

Remark 3.1.1. Which curve *applies* for a realistic scenario is not a mathematical question, but a stakeholder decision. The freedom to choose is encoded in the choice of cost function and penalty parameter values.



(a) Without intervention.

(b) With intervention.

Figure 3.3: Comparison in optimal path for $\lambda_1 = 20$ with and without an intervention applied to sensor 5 at time $t = 10$.

3.2 Single Agent - Dynamic Intervention

Assume now that the agent has the ability to reduce the capacity of any sensor to register an agent. If $(\gamma, t)_k$ now represents the trajectory with s_k the discritised implicit parameter, at any point the agent has the possibility to active this capacity. Mathematically, we implement this by a binary function (or a set of binary variables for the discritised version):

$$b(s) = \begin{cases} 1 & \text{suppression activated at time } s, \\ 0 & \text{else.} \end{cases} \tag{3.4}$$

At any point of the trajectory specified by $s \in [0, S]$ with $b(s) = 1$, we seek the sensor i_0 closest to the agent's location $\gamma(s)$. We then replace, for a limited time, Δt, the registration probability for this sensor by a rescaled probability given by

$$p_{i_0} \mapsto \kappa p_{i_0}, \tag{3.5}$$

for $\kappa \in [0, 1]$, where $\kappa = 1$ would not disrupt the sensor, and $\kappa = 0$ would completely disable the sensor. In this experiment, we choose $\kappa = \frac{1}{10}$.

The suppression (or *confusion of the sensor*) comes at a price which we take into account in the cost function through the use of $\lambda_3 \geq 0$:

$$F[\gamma; T, \mathbf{b}] = \int_0^T P(\gamma(t))\ \mathrm{d}t + \lambda_1 \int_0^T |\dot{\gamma(t)}|^2\ \mathrm{d}t + \lambda_2 T + \lambda_3 \sum_k \Delta t_k \tag{3.6}$$

where Δt_k denotes the duration of the kth interference. Note that we have added to the pool of variables earmarked for minimisation the values $\mathbf{b} = (b_k)_{k=1}^{N_S}$ which are such that $b_k = b(s_k) \in \{0, 1\}$ for $k = 1, \ldots, N_S$. This is done by simulated annealing once again, which is well suited to the discrete nature of the binary variables.

Furthermore, given a path, we define by

$$D = \frac{1}{T} \int_0^T P(\gamma(t))\ \mathrm{d}t \tag{3.7}$$

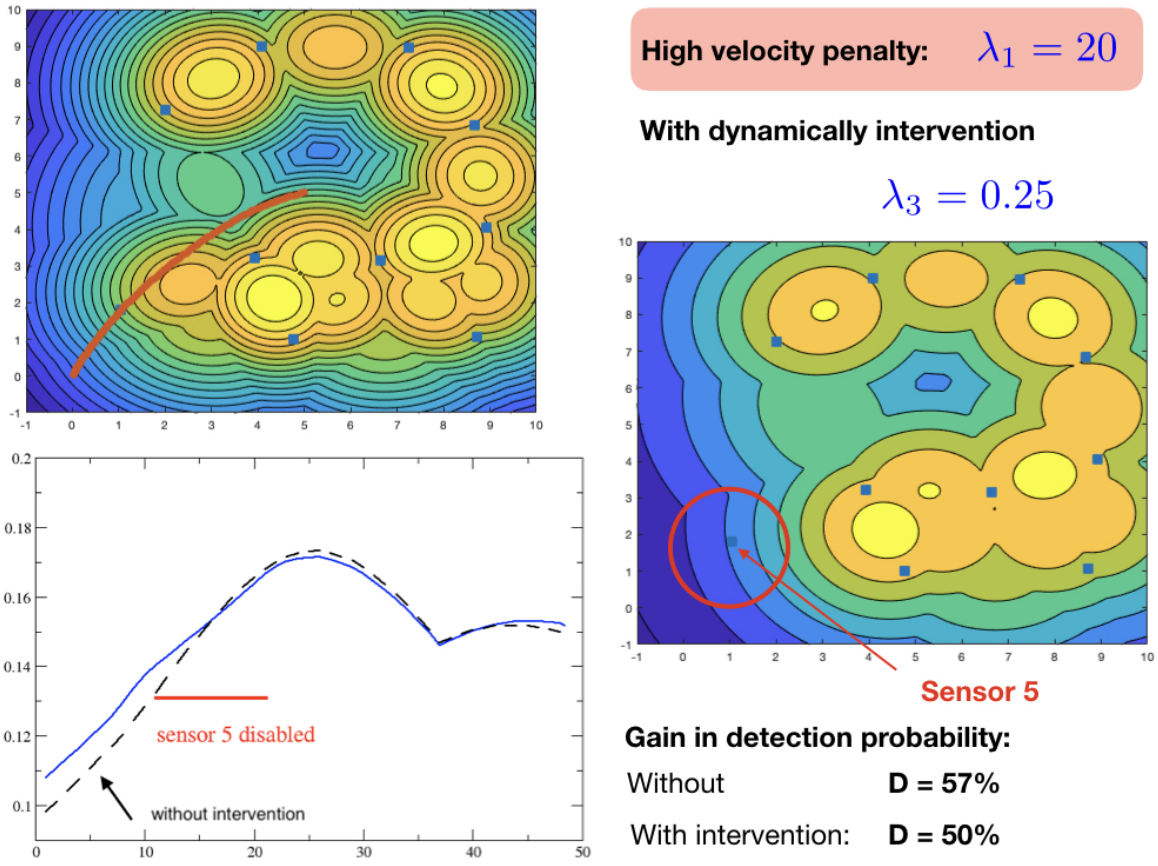


Figure 3.4: Analysis of change in optimal path for $\lambda_1 = 20$ with an intervention applied to sensor 5 at time $t = 10$. For the speed graph, the blue line represents the speed of path with intervention, whilst the dashed line represents the speed of the path with no intervention.

the overall detection probability along the path, where in this definition, we assume that the probability of detection P is adjusted to take into account the change in sensitivity of the sensors.

To study the effect of intervention, we again investigate the change in the optimal path and total detection probability for the case when $\lambda_1 = 20$ and 6 and throughout the following assume also that $\lambda_2 = 0$, i.e. do not consider penalty on traversal time.

Large Velocity Cost ($\lambda_1 = 20$): Consider the case $\lambda_1 = 20$ and compare the change in optimal path and detection probability when an intervention is introduced. Assume that the initial position is $A = (0, 0)$ and the target position is $X = (5, 5)$. Furthermore assume that $\lambda_3 = 0.25$, e.g. a low but nonzero cost to the use of equipment. Further assume that at time $t = 10$ (between 0 and $S = 50$) the agent travelling between A and X initiates an intervention, the results of which can be seen in Figure 3.4. One can observe from Figure 3.3, under the prescribed sensor and target configuration, for $\lambda_1 = 20$ the optimal path with intervention does not change significantly from the path when no intervention is initiated. Similarly, from Figure 3.4, the velocity graph for when an intervention is used or not is also very similar, although a slight increase in speed of traversal is observed at the beginning of the intervention journey, when the agent was close to the disabled sensor. This can be explained by the fact that near the disabled sensor, the probability of detection was low, and thus more cost can be invested in the velocity of the agent whilst still maintaining a *cheaper* journey. In addition, as one would expect, the cumulative probability of detection, D, does decrease with the use of an intervention.

Low Velocity Cost ($\lambda_1 = 6$): Consider now instead the scenario when $\lambda_1 = 6$, $A = (10, 0)$, $X = (5, 5)$. Assume also that the agent is able to choose whether or not to disable a sensor dynamically along its path, depending on whether it is *cheaper* (with respect to the cost function) to do so or not. If the cost of the intervention is too large, and the task is to minimise F, even through an intervention will reduce the total probability of detection D, the drop in probability may not be worth the cost of the use of equipment, and thus it may be *cheaper* not using any interventions at all. Indeed, this can be observed in Figure 3.5. If $\lambda_3 = 0.5$, with the ability to dynamically disable sensors along its path, in the setting provided, the cost of equipment usage is too high for it to make sense to use them to reduce the detection of probability. Although if we drop λ_3 to 0.22, the agent now sees that it is overall *cheaper* to use an intervention to disable sensor 10, than to not at all, which therefore reduces its total probability of detection. Furthermore, if we drop λ_3 to 0.16, then the agent determines that it is even *cheaper* to disable three sensors along its path to reach its objective, which results in a significant drop in the probability of detection when compared with other values of λ_3.

3.3 Conclusion

In conclusion, a method for determining the optimal path of an agent through a probability landscape has been obtained, in both the case when intervention is and is not possible. The method used to obtain the optimal path of agents to reach a target in this section is known as *simulated annealing*, a well studied and developed tool. Furthermore, at its current stage, optimal path computations take in the order of tens of seconds, which is likely to increase with the introduction of more complex and realistic model features, or increase in number of sensors. Although, given the time frame of the study group, an efficient algorithm was not the main priority, and thus there is still lots of scope for improvement. In addition, various similar studies have been performed in obtaining optimal paths using simulated annealing, e.g. for ship routing, see [5]. Other methods are also common in route planning of robots, see [1].

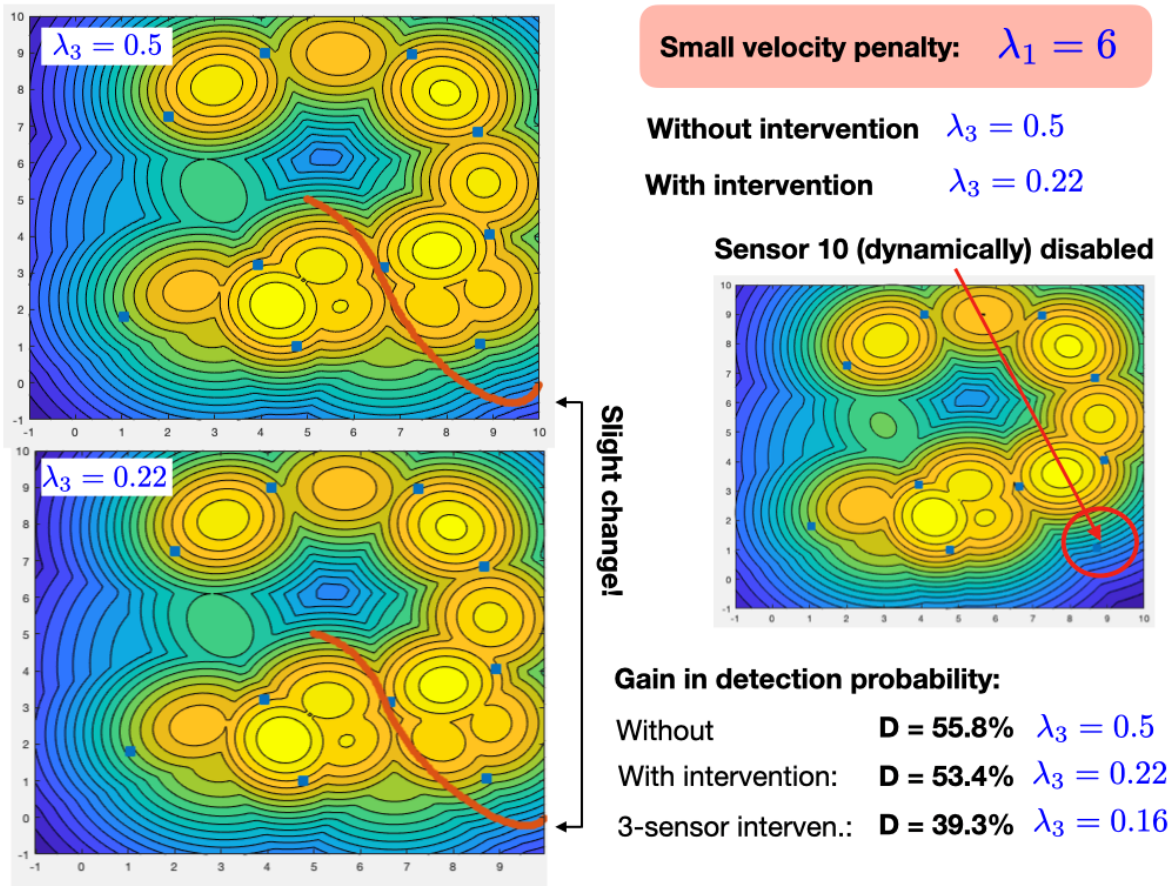


Figure 3.5: Analysis of change in optimal path for $\lambda_1 = 6$ and a new starting position at $(10, 0)$ with an intervention applied to sensor 10.

Chapter 4

Computational Study 2 - Nonlinear Integer Programming: Joerg Fliege

We now provide an alternative computational study of the minimisation problem, with a different choice in probability of detection and method of minimisation, that being nonlinear integer programming. The motivation behind the choice of such a minimisation procedure is due to the wide availability of commercial solvers for mixed-integer nonlinear problems, which over the last decades have made big improvements. The experiments conducted in this section have been performed using AMPL to allow for rapid prototyping, for which the code listing can be found in Appendix B.

Following Chapter 2, assume that N denotes the number of sensors and $P : \mathbb{R}^2 \to [0, 1]$ denotes the probability of detection of an agent at position $z \in \mathbb{R}^2$. Furthermore, alternatively to the definition given in Equation 2.12, to an admissible path $\gamma \in \mathcal{A}$ we can associate with it a cost function proportional to the cumulative probability of detection of an agent traversing the path by the path integral

$$F[\gamma; T] := \int_\gamma P(z)\ \mathrm{d}z = \int_0^T P(\gamma(t))|\dot{\gamma}(t)|\ \mathrm{d}t, \tag{4.1}$$

for some unknown traversal time $T > 0$. Then, to determine the optimal path under no interventions, i.e. where agents decide not to disable or confuse sensors, the problem would be to minimise the cost function F with respect to γ which has been treated previously and is a standard nonlinear optimisation problem.

4.1 Set-up of Multiple Agents with Intervention Model

Assume now that agents have the ability to disrupt or disable sensors, and denote by N_A the number of agents in the field, whom we'll assign a numeric label $i = 1, \ldots, N_A$. If agent i decides to disable sensor k at time t to account for this in the optimisation problem, we can introduce a binary decision variables $u_{i,k,t} \in \{0, 1\}$, where

$$u_{i,k,t} = \begin{cases} 1 & \text{agent } i \text{ disables sensor } k \text{ at time } t, \\ 0 & \text{agent } i \text{ does not disable sensor } k \text{ at time } t. \end{cases}$$

Constraints will assume $u_{i,k,t} = 0$ if agent i is too far away from sensor k. Assuming discrete time steps t, we now need to introduce an additional term in the objective function of the form

$$\sum_{i,k,t} c_{i,k,t} u_{i,k,t}$$

where $c_{i,k,t}$ denotes the cost for agent i disabling sensor k at time t.[1] Alternatively, we can treat this cost term as an additional objective function in a multi-objective setting. While this appears to be a large number of additional variables, note that the number of agents as well as the number of sensors is not that large, and that we can ignore the index t if we assume that the agent disables a sensor at the earliest possible time.

Now, given the disruption capabilities, denote by $P(z,u) : \mathbb{R}^2 \times \{0,1\}^{N_A} \to [0,1]$ the probability of detection at point z given the decisions to disable encoded in u. The objective function which now takes into account intervention capabilities has the form

$$F[\gamma, u] = \int_\gamma P(z,u)dz + \lambda c^\top u \tag{4.2}$$

where $\lambda \geq 0$ is a weighting factor and acts in the same way to λ_i as defined in Chapters 2 and 3. The choice of this weight λ is a stakeholder decision and can be changed at will, depending on how much effect a stakeholder would like interventions to have on the cost function F, indeed a high value of λ would invoke a high cost on equipment usage, and a low value of λ would invoke a low cost. Note that the objective function now depends on the path variable γ and vector of binaries u. Obviously, for fixed u this new objective is again a standard *shortest geodesic* problem. It is also interesting to note that for fractional values $0 < u_{i,k,t} < 1$ the above formulation still makes sense where the fraction can be interpreted as a damping factor applied to sensor k, i.e. agent i *confuses/disrupts* sensor k.

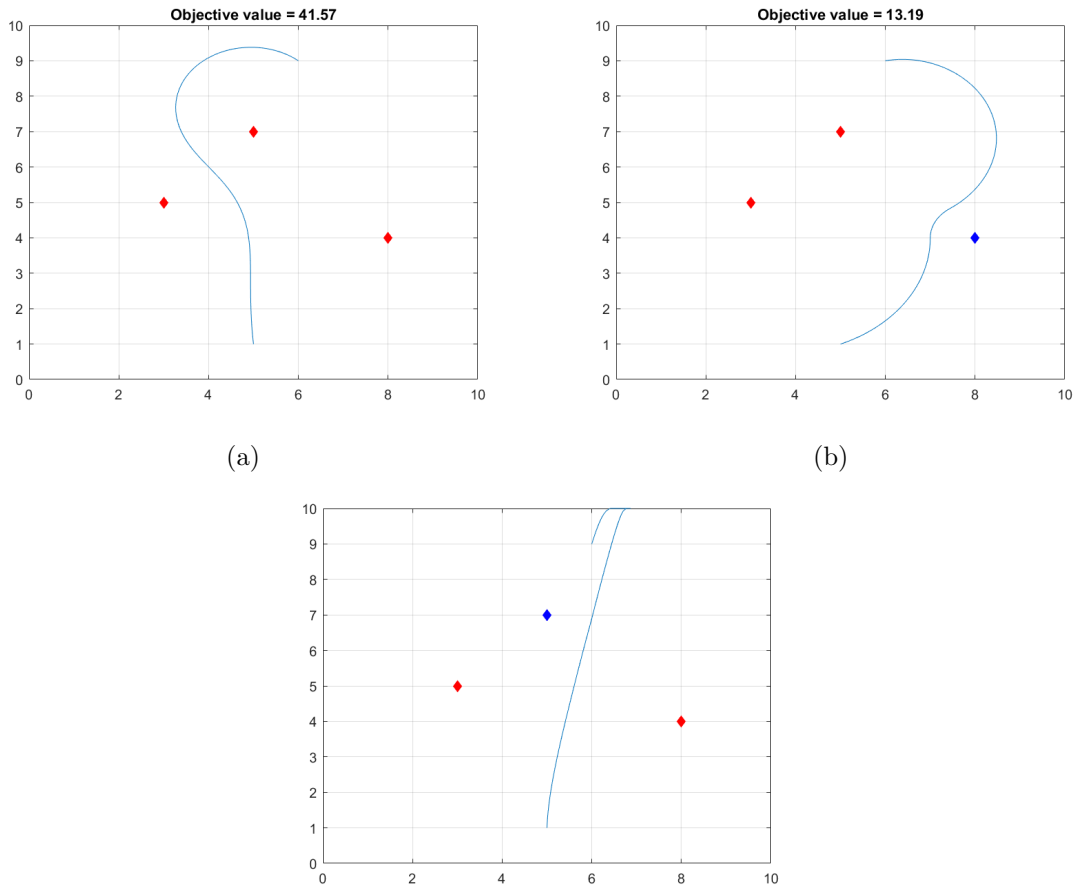
There are a number of software systems available that allow for the tackling of such mixed-integer nonlinear problems. A list of solvers available on the NEOS server can be found at `https://neos-server.org/neos/solvers/index.html#minco`. Among these, KNITRO (`https://www.artelys.com/solvers/knitro/`) and MINLP are generally seen as the most advanced systems. Other robust solvers include ANTIGONE (`https://www.gams.com/latest/docs/S_ANTIGONE.html`) and BARON (`https://minlp.com/baron`). All underlying algorithms in these solvers rely on sophisticated branch-and-bound techniques to implicitly enumerate the *discrete part* of the variable space in order to eliminate as many non-optimal variable choices as possible.

The efficiency of the approach outlines above rests crucially on the ability to solve, for a given choice of discrete variables u, a nonlinear problem with objective function $F[\cdot, u]$, or provide good lower bounds to it (this is often relies on the function $F[\cdot, u]$ being *not too non-convex*). Multi objective mixed-integer problems have recently attracted attention, see [2] and the Matlab prototype described therein.

4.2 Example Scenario

Let us consider a simple example with three sensors, i.e. $N = 3$, in two-dimensional space, at coordinates $(3,5)$, $(5,7)$, and $(8,4)$ respectively. Assume further that the target is located at $X = (6,9)$, while the agent starts at coordinate $A = (5,1)$. We have added a constraint for overall flight time, but do not penalize long flight times. This approach is equivalent to penalizing flight time, i.e. each minimum of a problem where flight time is penalized in the objective can be recovered as a minimum of a problem with a constraint on flight time, and vice versa. Figure 4.1a shows the optimal trajectory of the drone, computed by KNITRO within 1.7 seconds.

[1] It is not known whether we can precompute/estimate these coefficients? Although this information would be useful.



(a) (b)

(c)

Figure 4.1: (a) Optimal pathing with no intervention and no penalty to flight time. (b) Optimal pathing whilst turning off one sensor (8,4). (c) Optimal pathing whilst turning off one sensor (5,7).

Remark 4.2.1. Note that the objective function value is a path integral over probabilities, and so would need to be divided by flight time to arrive at an average probability of detection.

We now add a binary variable that indicates if the sensor at $(8, 3)$ should be disabled. Solving the modified problem results in a solution depicted in Figure 4.1b. The optimisation code successfully identifies that the sensor should be disabled, as indicated by the colour blue.

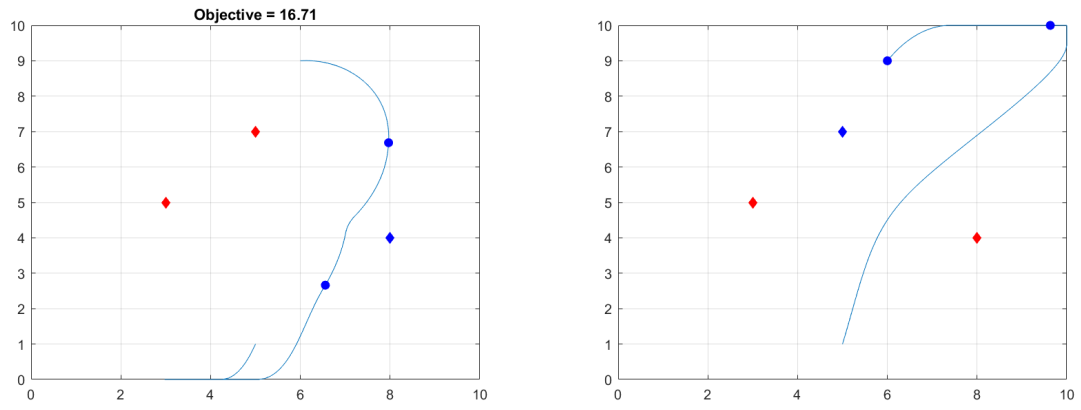
Next, we introduce three binary variables u_s to decide which sensor $s = 1, 2, 3$ should be disabled, that is

$$u_s = \begin{cases} 1 & \text{sensor } s \text{ is disabled,} \\ 0 & \text{sensor } s \text{ is not disabled,} \end{cases} \tag{4.3}$$

and allow only one sensor to be disabled by adding the constraint

$$\sum_s u_s = 1. \tag{4.4}$$

The result, achieved after 0.5s of computation time with KNITRO, is depicted in Figure 4.1c. Note that the drone first passes by the target to achieve greater distance to the two remaining sensors, to return later to the target by approaching from the North. This indicates that the problem setup allows for too much flight time, or that the flight time is not appropriately penalized.



(a) (b)

Figure 4.2: (a) Optimal pathing whilst turning off one sensor (8,4) twice during the total flight time. (b) Optimal pathing whilst turning off one sensor (5,7) twice during flight time.

4.2.1 Time-limited Interventions

As the next step, we consider a time-limited disruption: deciding again if we should or should not disable the sensor at (8,4), we now also have to decide when to disable it. Once disabled, the sensor will stay disabled for 30 time steps (out of 110). Figure 4.2a shows the optimal trajectory, as well as the time when the sensor at (8,4) is disabled (first blue dot on the trajectory) and goes online again (second blue dot on the trajectory). One can also observe that the drone is idling for some time at the start of the trajectory, again because travel time is not penalized here.

As before, we now introduce three binary variables to decide which sensor to disable, as well as the start of the time window during which the sensor is disabled. Figure 4.2b shows the corresponding result.

Unfortunately, the problem formulation with both binary variables to decide which sensor to disable and a decision variable for the time window to do so does not appear scale well: problems with six sensors or more can only be solved when good starting guesses for the variables are supplied (which could be computed with a more robust method, i.e. some Monte Carlo algorithm), and accuracy achieved deteriorates from an error of circa $1e-6$ in previous experiments to $1e-2$.

4.2.2 Fractional Sensor Disruption

Instead, we now turn our attention to a problem formulation that avoids binary variables and instead allows to degrade sensor performance fractionally. For this, introduce variables $z_s(t) \in [0,1]$, indicating by how much the performance of sensor s is degraded at time t. The objective then becomes

$$\min \sum_t \left(1 - \prod_{s \in S} (1 - z_s(t) p_s(x_1(t), x_2(t)))\right) \tag{4.5}$$

with detection probability functions p_s depending on the location of the drone as before. We depict the result of a computational experiment with 5 sensors in Figure 4.3. Two sensors are disabled; one at $(1,5)$ first and then the one at $(5,\ 7)$ later for a shorter period of time. As it turns out, for the new variables we have $z_s(t) \in \{0,1\}$ at all times for the optimal solution.

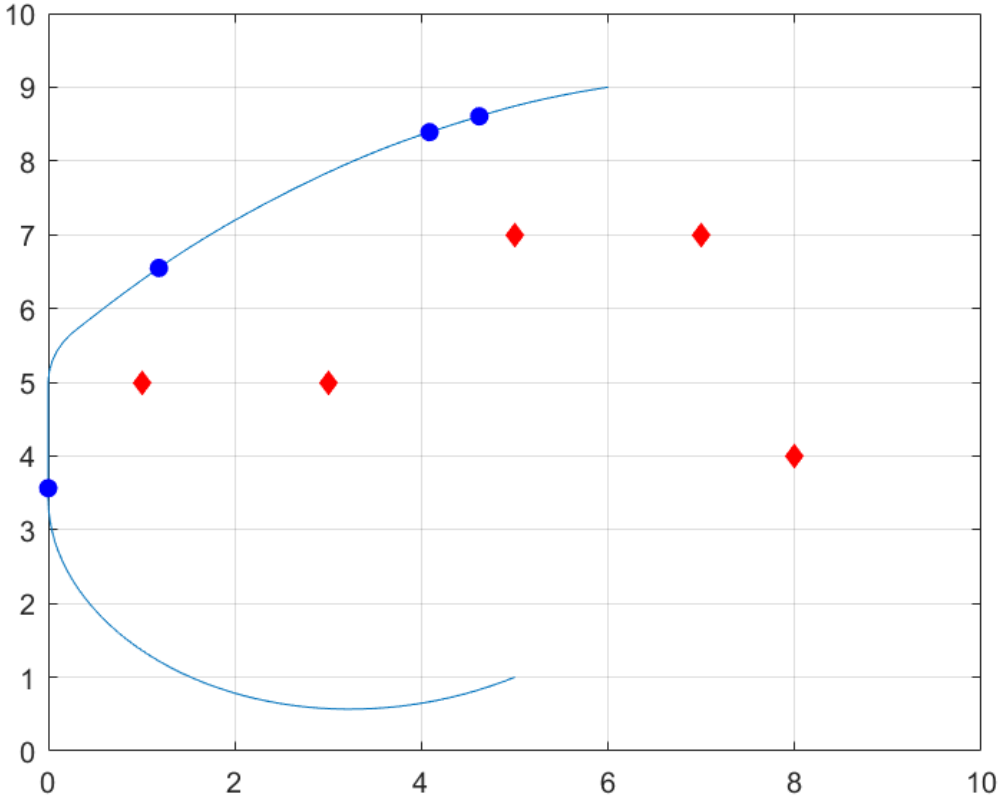


Figure 4.3: Optimal pathing in a 5 sensor system whilst turning off two sensors at different times.

4.3 Conclusions

The problem formulation described above seems to scale well when the 2D space is replaced by 3D (we suspect the bottleneck is the number of binary variables and the type of constraints that couple them). It also appears to be easy to extend the formulation to uncertainties in the location of the sensors, if we allow these locations to vary within balls of a given radius around a center. Furthermore, it seems that the implementation outlined above can easily consider the scenario of moving sensors.

Discussion - Continuous Approach

For the continuum approach to the *Escape the Sensor* problem, there is currently still lots of scope for further study. The implementations discussed in Chapters 3 and 4 are essentially equally capable, thus similar extensions apply to both. Some of these include:

- Introduce more realistic model features, e.g. terrain, obstacles and moving sensors (e.g. patrol);
- Have no/partial knowledge of prior sensor configuration;
- Incorporate multiple agents who are working together into the model (introduce *game playing* mechanics);
- Optimisation of optimal route finding algorithm;
- Create easy to use software which takes as an input various known parameters, and outputs a single/numerous optimal path;
- Study further the Finite Element Method solver for the optimisation problem to obtain an alternative solution approach. This approach may be mathematically more taxing, although may result in more efficient and/or accurate solvers.

Part II

Discrete Approach

Chapter 5

Discrete Approach 1 - John Beasley

This chapter discusses the discrete approach written by John Beasley with minor modifications.

5.1 Overview

The main features of the formulation below are:

- it is a zero-one integer programme with linear constraints, for which very powerful commercial software optimisation packages exist; implying that numeric solution may not necessitate special purpose algorithms but rather direct application of a standard package
- it includes agent movement for multiple agents
- it models the end of the game (i.e. when one agent reaches the target)
- it includes sensor knockout (disablement)
- it includes sensor detection probabilities
- it includes agents confusing sensors
- it includes cost for sensor knockout and cost for sensor confusion
- as it models time to end of game; total cost; total no detection probability over the game; any of these factors can be constrained. Also we can optimise any of these factors individually, or a optimise a weighted combination of them.

5.2 Introduction

We will discretise the problem both in space and time. So consider an (irregular) mesh (graph) placed over the 2-d space. Agents move from vertex to vertex on this graph where a movement from one vertex to an adjacent vertex takes one time step. The aim of the game is to have one agent reach the target vertex as quickly as possible without detection.

Figure 5.1 show an example problem. In that figure we have used a regular grid graph discretisation with the target vertex being shown in green. We have two agents, shown in blue. The red circles are the detection circles associated with each sensor. Note here that although the agents move between vertices, and the target is also at a vertex, there is no requirement for the sensors to be positioned at vertices.

Agent movement

Our notation is:

- we have a graph with N vertices (nodes) and $\Gamma(v)$ is the set of vertices adjacent to vertex v in this graph (where $v \notin \Gamma(v)$)
- we have A agents, where agent a starts at time 0 at vertex $\gamma(a)$

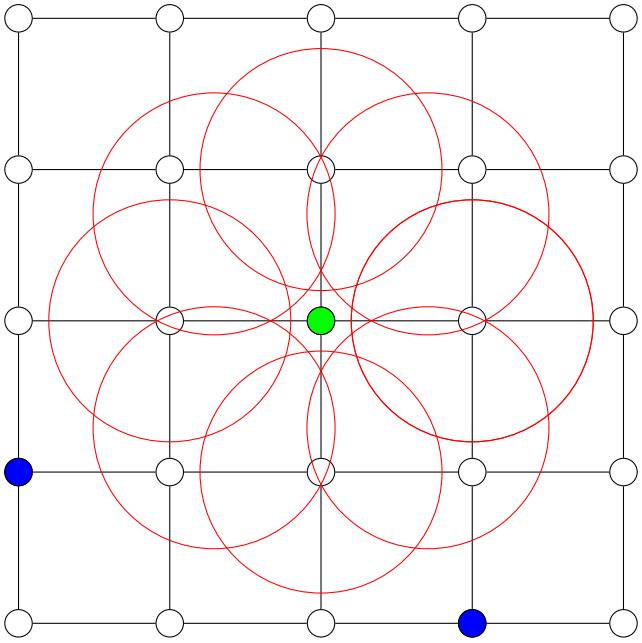


Figure 5.1: Example problem

- the target is vertex N
- the number of time steps allowed is T, so we aim to have finished the game as quickly as possible within T time steps
- to model the end of the game, and formulate the objective function, we introduce an artificial vertex, vertex 0, which is adjacent to all other vertices, i.e. $0 \in \Gamma(v),\ v = 1, \ldots, N$. This vertex acts as an absorbing vertex, so $\Gamma(0) = \emptyset$ and note that all agents can make a transition to vertex 0 from any other vertex $v \neq 0$ at any time.

5.3 Variables

We have the following variables:

- $x_{avt} = 1$ if agent a is at vertex v at time t, $= 0$ otherwise

5.4 Movement constraints

We have the following constraints which govern the movement of agents at each time step:

$$\sum_{v=0}^{N} x_{avt} = 1 \qquad a = 1, \ldots, A;\ t = 1, \ldots, T \tag{5.1}$$

Equation (5.1) ensures that each agent is at some vertex at each time step.

$$x_{av,t+1} \leq x_{avt} + \sum_{k:\ v \in \Gamma(k)} x_{akt} \qquad a = 1, \ldots, A;\ v = 0, \ldots, N;\ t = 0, \ldots, T-1 \tag{5.2}$$

Equation (5.2) ensures that at time $t + 1$ an agent can only be at vertex v if at the previous time step t it was either already at vertex v (so has not moved) or was at some vertex k from which it could reach vertex v in one time step.

Taken together Equation (5.1) and Equation (5.2) ensure that at each time step an agent is only at one vertex, and moreover that vertex must be one which is reachable from the vertex the agent was at at the previous time step. Hence the path for an agent must consist of a succession of adjacent vertices.

Note here that the initial conditions are:

$$x_{a\gamma(a)0} = 1 \quad a = 1, \ldots, A \tag{5.3}$$

$$x_{av0} = 0 \quad a = 1, \ldots, A; v = 0, \ldots, N; v \neq \gamma(a) \tag{5.4}$$

To deal with the end of the game we introduce constraints that force a transition to vertex 0 from vertex N. Namely

$$x_{a0,t+1} \geq x_{aNt} \quad a = 1, \ldots, A;\ t = 1, \ldots, T-1 \tag{5.5}$$

Equation (5.5) ensures that if agent a is at vertex N at time t it is at vertex 0 at time $t+1$. It will remain at vertex 0 thereafter as it cannot transition away.

In addition we add a constraint that ensures that at least one agent reaches vertex N, namely:

$$\sum_{a=1}^{A} \sum_{t=1}^{T} x_{aNt} \geq 1 \tag{5.6}$$

In order to ensure that once an agent reaches vertex 0 they stay there we impose the constraint:

$$x_{a0,t+1} \geq x_{a0t} \quad a = 1, \ldots, A;\ t = 1, \ldots, T-1 \tag{5.7}$$

Our objective function then is:

$$\text{Minimise} \sum_{a=1}^{A} \sum_{t=1}^{T} t x_{aNt} \tag{5.8}$$

In Equation (5.8) we minimise the sum of the times at which each agent reaches the target. Because Equation (5.6) means that at least one agent reaches the target the minimisation here will mean that once one agent reaches the target all agents will make a transition to vertex 0 signifying the end of the game.

Equation (5.8) therefore will in the optimal solution have only one nonzero term corresponding to the time at which some agent reaches the target.

To ensure that all agents remain in the game until one reaches the target we must disallow transitions to vertex 0 until the game has ended. The constraint that ensures this is:

$$x_{a0t} \leq \sum_{b=1}^{A} \sum_{\tau=1}^{t} x_{bN\tau} \quad a = 1, \ldots, A;\ t = 1, \ldots, T \tag{5.9}$$

This constraint means that no agent can make a transition to vertex 0 until the right-hand side is at least one, which will happen once an agent has reached the target.

5.5 Agent detection

We assume that detection of an agent only occurs at vertices (so not during movement between vertices) and so at time step t detection (if any) occurs at the vertex to which the agent has

just moved, i.e. detection is based upon the values of x_{avt} $t \geq 1$. This implies that we assume that at time 0 the agents are positioned so as to be undetectable.

With reference to terminology here note that below we refer to detection of an agent. This (depending upon the context) may refer to detection of an agent by a single sensor, or detection of an agent by two or more sensors.

We assume:

- detection is binary, so either a sensor detects an agent or not (so not a probabilistic model currently)
- we can only disable (knockout) sensors (so not considering confusing sensors, however that defined)

5.5.1 Notation

Our notation is:

- we have S sensors at known locations
- $d_{sv} = 1$ if sensor s can detect an agent at vertex v under normal operating conditions, $= 0$ otherwise. Without significant loss of generality we assume that $d_{sN} = 0,\ s = 1, \ldots, S$ so no sensor can detect an agent at the target vertex N (since that is the end of the game). Similarly $d_{s0} = 0,\ s = 1, \ldots, S$.
- $K_{sv} = 1$ if an agent at vertex v can disable (knockout) sensor s (i.e. it is in range), $= 0$ otherwise
- Δ is the number of time steps for which a sensor is disabled once it has been first knocked out
- no agent can be detected by Ω or more sensors at any time
- Since detection of an agent relies on at least Ω sensors we automatically know that any vertex v which not in range of Ω or more sensors is of no detection interest. Hence in terms of detection we need only concern ourselves with the set of vertices V where $V \subseteq [v \mid v = 1, \ldots, N-1]$ is the set of vertices which are in range of Ω (or more) sensors.
- C is the cost of a single knockout, with the total knockout cost being limited by B

5.5.2 Variables

We have the following variables:

- $\alpha_{avt} = 1$ if agent a at vertex v at time t chooses to knockout all sensors that are in range, $= 0$ otherwise
- $y_{sat} = 1$ if sensor s detects agent a at time t, $= 0$ otherwise

Note there that although we need only restrict potential detections to vertices $v \in V$ it is possible that doing a knockout at some vertex $v \notin V$ would be worthwhile due to the position of that vertex and the sensors within knockout range. For this reason the variables α_{avt} need to be defined over the entire set of vertices.

5.5.3 Detection constraints

Clearly we cannot perform a knockout unless an agent is at a vertex at the appropriate time. The constraint that ensures this is:

$$\alpha_{avt} \leq x_{avt} \quad a = 1, \ldots, A;\ v = 1, \ldots, N;\ t = 1, \ldots, T \tag{5.10}$$

The constraint limiting the total knockout cost is:

$$\sum_{a=1}^{A} \sum_{v=1}^{N} \sum_{t=1}^{T} C\alpha_{avt} \leq B \tag{5.11}$$

Now at vertex $v \in V$ an agent a can only be detected by sensor s at that vertex at time step t if $x_{avt} = 1$. However that detection is negated if the vertex is out of range of sensor s or the associated sensor s has been knocked out either at time step t or in one of the preceding Δ time steps. This therefore leads to the constraint:

$$y_{sat} \geq d_{sv}[x_{avt} - [\sum_{j=1}^{N} K_{sj} \sum_{b=1}^{A} \sum_{\tau=max(1,t-\Delta)}^{t} \alpha_{bj\tau}]]\ \ s = 1, \ldots, S;\ a = 1, \ldots, A;\ \forall v \in V;\ t = 1, \ldots, T \tag{5.12}$$

To clarify Equation (5.12) first note that this constraint is inactive if $d_{sv} = 0$, i.e. if sensor s cannot detect an agent at the vertex v under consideration on the right-hand side of Equation (5.12). Similarly this constraint is inactive if $x_{avt} = 0$. Hence Equation (5.12) can only (potentially) force y_{sat} to be one for agent a at time t if we have a vertex v for which $d_{sv} = 1$ and $x_{avt} = 1$ (recall here that each agent can only be at one vertex at each time step).

Now in the right-hand side of Equation (5.12) the second term will be non-zero (so rendering the constraint inactive) if there is some vertex j from which it is possible to disable sensor s (so $K_{sj} = 1$) and some agent b at some time τ (within the appropriate knockout time range before t) has chosen to knockout the sensors in range of vertex j.

Now as each agent must remain undetected at all times this means that no agent can be detected by Ω or more sensors at any time . The constraint that ensures this is:

$$\sum_{s=1}^{S} y_{sat} \leq \Omega - 1 \quad a = 1, \ldots, A;\ t = 1, \ldots, T \tag{5.13}$$

On a technical note here Equation (5.12) forces y_{sat} to be one if the right-hand side is one, but places no restriction on y_{sat} if the right-hand side is zero. However since Equation (5.13) restricts the sum of the y_{sat} then y_{sat} will in many cases be automatically assigned a value of zero to satisfy this equation. However there may be computational benefit in adding the constraint:

$$y_{sat} \leq d_{sv} x_{avt} \quad s = 1, \ldots, S;\ a = 1, \ldots, A;\ \forall v \in V;\ t = 1, \ldots, T \tag{5.14}$$

Equation (5.14) explicitly forces y_{sat} to be zero if either d_{sv} or x_{avt} are zero. Whilst this constraint does not deal with the case where $d_{sv} = x_{avt} = 1$ and the second term in the right-hand side of Equation (5.12) is nonzero we know that only one $x_{avt} = 1$ (see Equation (5.1)) and so we have addressed in Equation (5.14) the vast majority of cases where y_{sat} should be forced to zero. Moreover since there are many possibilities for a nonzero value in the second term in the right-hand side of Equation (5.12) we leave this case unaddressed. This is still technically correct due to the previous remark above relating to Equation (5.12).

5.6 Other constraints

5.6.1 Probabilities

The formulation above assumes detection by sensor s at vertex v is perfect if $d_{sv} = 1$ and the sensor is not knocked out. Given the framework in terms of variables and constraints established above it is possible to include detection probabilities.

Reinterpret $d_{sv} = 1$ to mean that sensor s can detect an agent at vertex v under normal operating conditions, but with an associated probability q_s of missing such a detection ($0 \leq q_s \leq 1$). Then the probability of there being *no (single sensor, single agent) detections over the course of the game* is given by:

$$\prod_{s=1}^{S} \prod_{t=1}^{T} \prod_{a=1}^{A} \prod_{y_{sat}=1} q_s \tag{5.15}$$

In Equation (5.15) we assume that all the probabilities are independent and the expression seen is the total probability of no detections over the course of the game.

Hence if we let Q be the minimum no detection probability we require over the course of the game we have the constraint:

$$\prod_{s=1}^{S} \prod_{t=1}^{T} \prod_{a=1}^{A} \prod_{y_{sat}=1} q_s \geq Q \tag{5.16}$$

Note here that if we have any sensor s for which $q_s = 0$, so the sensor never misses a detection, then it is impossible to satisfy this constraint unless $y_{sat} = 0 \; \forall a, t$. This would need to be added as a constraint if such sensors exist.

Clearly Equation (5.16) is nonlinear but can be linearised to:

$$\sum_{s=1}^{S} \sum_{t=1}^{T} \sum_{a=1}^{A} \ln(q_s) y_{sat} \geq \ln(Q) \tag{5.17}$$

Notice how in Equation (5.17) we have included the y_{sat} in the linear sum of the log terms.

There is one complexity here. Above when we dealt with detection it was in terms of: no agent can be detected by Ω or more sensors at any time. This enabled us to define the set V for the detection constraints. But here detection is phrased in terms of Q, the minimum no detection probability we require over the course of the game. As such V may need amending and it is unclear whether we might wish to include both types of detection in the formulation or not.

5.6.2 Sensor confusion

We interpret sensor confusion to mean that the previous probability of a sensor missing a detection (q_s) is increased. Let $q_s^c \geq q_s$ be the probability that applies after confusion.

For ease of notation here we assume that agents have the same range for confusion as for knockout, and further that confusion applies for the same number of time steps Δ as knockout.

Let $\beta_{avt} = 1$ if agent a at vertex v at time t chooses to confuse all sensors that are in range, $= 0$ otherwise. Let $z_{sat} = 1$ if sensor s is confused by agent a at time t, $= 0$ otherwise.

Here the β_{avt} variables relate to whether (or not) agent a initiates a confusion signal to sensors when the agent is at vertex v at time t. The z_{sat} variables relate to whether sensor s is confused or not for agent a at time t (possibly by a previous confusion initiation by a different agent).

Confusion can only apply if:

$$z_{sat} \leq [\sum_{j=1}^{N} K_{sj} \sum_{b=1}^{A} \sum_{\tau=max(1,t-\Delta)}^{t} \beta_{bj\tau}] \quad s = 1, \ldots, S;\ a = 1, \ldots, A;\ t = 1, \ldots, T \tag{5.18}$$

Equation (5.18) ensures that z_{sat} is zero if there is no vertex j in range of sensor s for which some agent b has chosen to confuse all sensors at some time τ in the appropriate time range. Clearly we cannot confuse all sensors in range of a vertex v unless an agent is at the vertex at the appropriate time. The constraint that ensures this is:

$$\beta_{avt} \leq x_{avt} \quad a = 1, \ldots, A;\ v = 1, \ldots, N;\ t = 1, \ldots, T \tag{5.19}$$

Confusion cannot apply if an agent is not detected. The constraint that ensures this is:

$$z_{sat} \leq y_{sat} \quad s = 1, \ldots, S;\ a = 1, \ldots, A;\ t = 1, \ldots, T \tag{5.20}$$

Obviously the optimal strategy is to always confuse sensors unless there is some limitation and so here we assume that confusion involves a cost C^c and so the constraint limiting the total cost, (Equation (5.11)), becomes:

$$\sum_{a=1}^{A} \sum_{v=1}^{N} \sum_{t=1}^{T} (C\alpha_{avt} + C^c \beta_{avt}) \leq B \tag{5.21}$$

The linearised constraint involving probabilities (Equation (5.17)) then becomes:

$$\sum_{s=1}^{S} \sum_{t=1}^{T} \sum_{a=1}^{A} [\ln(q_s) y_{sat} + (\ln(q_s^c) - \ln(q_s)) z_{sat}] \geq \ln(Q) \tag{5.22}$$

Collectively Equations (5.18)-(5.20) ensure that z_{sat} will be zero unless the conditions that enable it to be one are satisfied. The nature of the left-hand side of Equation (5.22) will force z_{sat} to one if necessary to satisfy this equation and the other constraints allow it. Technically in the numeric solution of the problem we could have Equation (5.22) satisfied with $z_{sat} = 0$ when the values for the other variables imply that we should have $z_{sat} = 1$. However setting $z_{sat} = 1$ will simply increase the left-hand side of Equation (5.22) and so is simply a post-solution action that results from some indeterminacy in the formulation and does not affect the validity of the formulation.

5.7 Conclusions

In this chapter, a mathematical formulation for the Escape the Sensor problem has been provided where a discrete approach has been taken. For the formulation it has been shown that optimisation provides a flexible and approachable modeling framework as well as several objectives (min. detection probability, max speed, etc) being able to be incorporated easily if the problem requires it. With these objectives, it can be said that interventions may have a greater impact on detection probability than on the selected route as well as this approach having the potential to decouple route finding and managing the probability of detection.

The above method shows a clear way to compute and interpret trade offs in the problem definition with a modest computational complexity. By using a capable solver, a user could input multiple initial conditions to possibly obtain multiple locally optimal paths. However, this chapter has not discussed some points and could be used as an extension to the work. These extensions could be adding uncertainty measures in the sensors which could cause a problem in the robustness of the optimisation as well as some further uncertainties, introducing *rolling horizon* optimisation.

Further work could incorporate extensions to higher dimensions (such as 2D+ or 3D) which would add computation time or even adding movement to the sensors if the system is dynamic. Given that this problem provides a fair amount of knowledge of the sensor network, a modification for little/partial knowledge of the sensor network could be a worthy avenue for practicality purposes.

Chapter 6

Discrete Approach 2 - Jaroslav Fowkes

6.1 Introduction

We discretise the problem both in space and time. We consider an (irregular) mesh (graph) placed over the 2-D space. Agents move from vertex to vertex on this graph where a movement from one vertex to an adjacent vertex takes one time step.

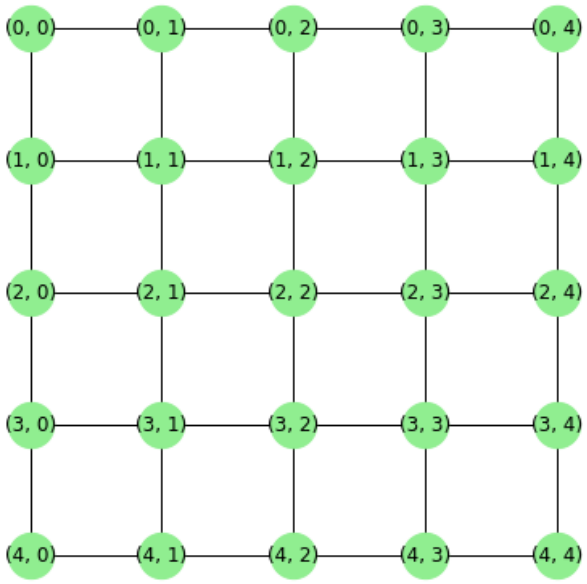


Figure 6.1: Discrete 5×5 grid showing the vertices where agents and sensors can be situated

The main features of the formulation developed using this discretisation are:

- a binary integer program with linear constraints able to find the globally optimal solution
- powerful commercial optimisation packages exist; no need for special purpose heuristics
- includes agent movement for multiple agents
- includes sensor knockout (disablement)
- includes costs for sensor knockout

Additionally, the model has been extended (not shown here, but in the code in Appendex C) to:

- model the end of the game (i.e. when one agent reaches the target)
- include sensor detection probabilities
- include agents confusing sensors
- include costs for sensor confusion.

As it models time to end of game; total cost; total "no detection" probability over the game; any of these factors can be constrained. Also we can optimise any of these factors individually, or a optimise a weighted combination of them.

6.2 Mathematical Formulation

In this section, we look to minimise time to target T

$$\min \sum_{a} \sum_{t} t x_{aTt} \tag{6.1}$$

where we define the binary variables used where $x_{avt} = 1$ if agent a is at vertex v at time t, $\alpha_{a,v,t} = 1$ if agent a at vertex v at time t chooses to knock out all sensors in range, and $y_{s,a,t} = 1$ if sensor s detects agent a at time t.

For this particular problem the authors note constraints of the scenario as; initial agent positions, each agent at one of the N vertices, agent allowed movement, at least one agent reaches target, only permit knockouts when agent at nearby vertex, limit total knockout cost, no agent can be detected by more than O sensors, sensor allowed detections.

6.3 Numerical Example

Consider the following toy problem: 2 blue agents seeking the red target over 4 time steps, avoiding 4 sensors and aiming to minimise the time to reach the target without detection, see Figure 6.2. Using the commercial MOSEK solver, we can solve this in 0.2s to give the solution shown in Figure 6.3.

For Figure 6.3a we see that 2 agents at positioned at (3,1) and (4,3) and turns off sensor (2,1) whilst confusing sensors (1,1), (1,3) and (2,3), from this we move to Figure 6.3b where we confuse the sensors in the same positions and one agent moves to (3,2). Using the MOSEK solver shows that using the constraints discussed, an agent can reach the objective without being detected and could possibly move back to the starting position if the problem requires it. For the full Python code listing to recreate these results, see Appendix C.

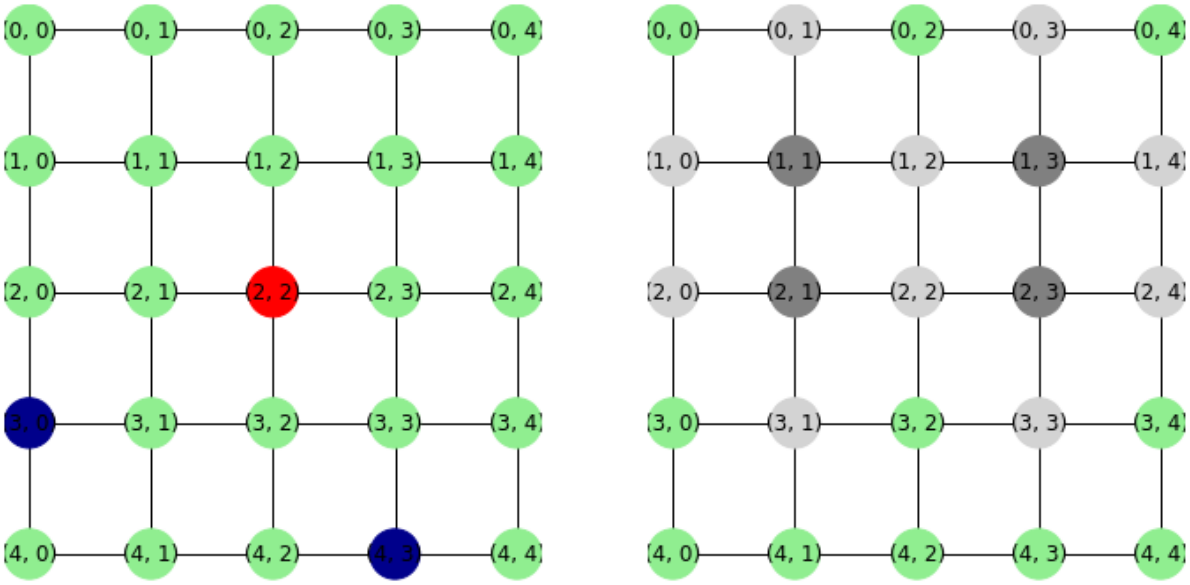
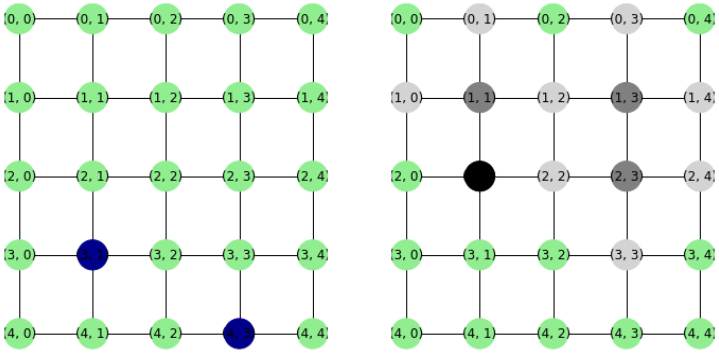
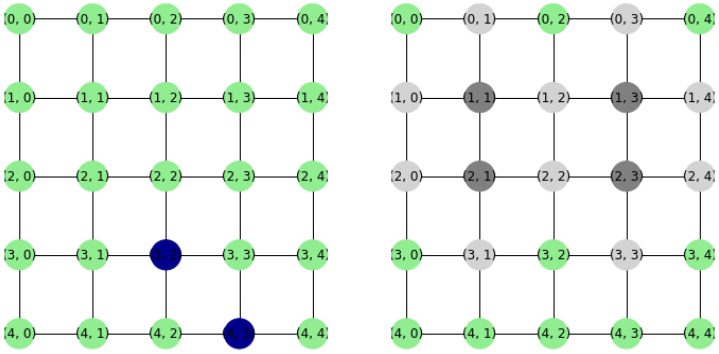


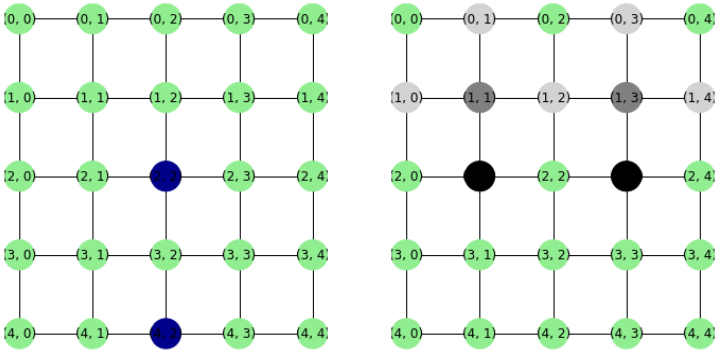
Figure 6.2: Intepretation of the problem field with 2 agents (blue), 1 target (red) and 4 sensors (dark grey) with a 1 adjacent vertex detection range



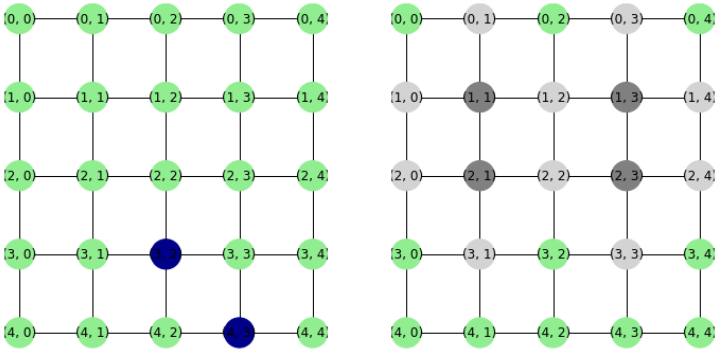
(a) Step 1



(b) Step 2



(c) Step 3



(d) Step 4

Figure 6.3: MOSEK solver graphical solution for the toy problem above

Discussion - Discrete Approach

For the formulation of the detection game in Chapter 5, a zero-one integer program for which commercial software packages (e.g. Cplex, Gurobi) are well developed and very powerful. The same formulation will depend upon the degree of discretisation, with a high resolution costing more in terms of computational resources so experience with the formulation on realistic test data will be needed for optimising performance. Although it may be possible to develop a heuristic solution to the problem, it is hoped that the mathematics given in this chapter will aid any future solutions.

With the formulation provided it can be obvious that there can be future work with introducing variations to total knockouts cost, B, i.e. $B = 0, 1, 2$, etc. and setting the cost of a single knockout, $C = 1$. Given some research in to 3-D graph problems, with the use of the discretised graph, there should be no problem in extending the 2-D in to the higher dimensional space. Introducing a time subscript to both the detection range of a sensor, d_{sv}, and the knockout ability, K_{sv}, of agent s at vertex v, this could possibly allow these quantities to become dynamic with time. In addition, dynamic route optimisation could also occur by solving the formulation at time zero, allowing the game to run using the predetermined agent moves/knockout actions for a number of time steps, then resolving with new information based on current agent positions and knockouts. We would also include here new/updated information as to sensors and their locations and detection ranges.

Appendix A

Continuum Approach - Post Meeting Comment by Kurt Langfeld

This comment is in extension to the FEM solution via an Euler-Lagrange approach discussed at the end of Chapter 2.

I would like to point out the similarity of this approach to a well-studied set of problems in Theoretical Mechanics: the movement of a point-like particle in an external potential $V(x)$. I here only consider time-independent potentials, but generalisations are possible.

Denote here by $x : [0, T] \to \mathbb{R}^2$ a path and T the flight time of the path. The trajectory $x(t)$ of a particle with mass m extremises the *action*

$$S[x] = \int_0^T L \, \mathrm{d}t, \qquad L = \frac{m}{2} \left(\frac{\mathrm{d}x}{\mathrm{d}t} \right)^2 - V(x). \tag{A.1}$$

The *Escape the Sensor* problem is of this kind. Here, we seek to minimise the cost function:

$$F[x] = \int_0^T L \, \mathrm{d}t, \qquad L = \lambda_1 \left(\frac{\mathrm{d}x}{\mathrm{d}t} \right)^2 + P(x). \tag{A.2}$$

where we identify:

$$m = 2\lambda_1, \qquad V(x(t)) = -P(x(t)). \tag{A.3}$$

If we do not admit time-dependent potentials, we are looking at the problem without interventions. Note that regions of high probability corresponds to regions of low potential. While this identification is *trivial*, the intuition, which we have for Theoretical Mechanics, helps a good deal with the problem at hand.

The Euler Lagrange equations are Newton's law for (non-relativistic) mechanics:

$$m \frac{\mathrm{d}^2 x(t)}{\mathrm{d}t^2} = F(x(t)), \quad F(x(t)) = -\nabla V(x(t)), \tag{A.4}$$

where F is called the *force*.

This is a very well-studied problem. If the problem has symmetry, Noether's theorem yields conserved quantities, i.e., combinations of position and velocity that do not depend on time.

Also, if the potential (or better the Lagrangian $\mathcal{L}$) does not explicitly depend on time t, there is a first integral of the equation of motion. For our case, we find that

$$E = \frac{m}{2}\left(\frac{\mathrm{d}x(t)}{\mathrm{d}t}\right)^2 + V(x(t)) \tag{A.5}$$

does not depend on time if $x(t)$ is a solution of the Euler-Lagrangian equation. E is then called (conserved) *energy*.

Let us apply this in the context of the *Escape the Sensor* problem. If $x : [0, T] \to \mathbb{R}^2$ is the optimal path, we find that

$$E(t) = \lambda_1\left(\frac{\mathrm{d}x(t)}{\mathrm{d}t}\right)^2 - P(x(t)) \tag{A.6}$$

is the same for all times $t \in [0, T]$. This has an intuitive interpretation: For any locally optimal path, the agent needs to increase speed, i.e., $\dot{x}$, in regions where the probability P is high (in order to have E a constant), i.e., we advise *run when detection probability is high.*

Towards a New Strategy: An agent in the field can always find locally optimal paths by simply reacting to the *force*[1]:

$$2\lambda_1\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2} = \nabla P(x(t)). \tag{A.7}$$

This can also provide a strategy for the case if the sensor array is not known. Key here would be to gather information, form $P(x)$ and its gradient and react according to the Newton equation of motion. When new sensors are discovered *in the distance*, it would not change much the force at the current position of the agent, and the agent is still moving locally optimally.

Next steps: There is more work to be done to make this practical. The reason is that in mechanics, we treat the equations of motion as an *initial value problem*: For a given position and initial velocity, we calculate the position of the particle at a later time. For this choice, the optimal paths can be found locally.

For the *Escape the Sensor* problem, we are dealing with a *boundary value problem*: We start initially at the origin and want to reach the target position. Whether this can be re-formulated as a *local problem* plus a directive (*keep an eye on the target position when you move*) is an interesting question for further studies.

[1] *Use the force, Luke.* Cannot remember where I heard this before.

Appendix B

Continuum Approach - AMPL Code Listing by Joerg Fliege

```
param NMAX = 110;
param x1start = 5; # drone start coordinates
param x2start = 1;
param x1end = 6; # drone target coordinates
param x2end = 9;

# Disabled for how long?
param dis_delta = 30;

set T  default 1..NMAX;  # timesteps
set T2 default 2..NMAX;
set S  default 1..8; # sensors

# Drone coordinates, box constraints, and possible starting guesses:
var x1{t in T} <= 10, >= 0; # := (t/MAXT) * (x1end-x1start);
var x2{t in T} <= 10, >= 0; # := (t/MAXT) * (x2end-x2start);

var flighttime >= 0, <= 500; # Can be used as a penalty in the objective.

# Probabilities of detection:
var p{S,T} >=0, <= 1; # You don't need these bounds.
#
# Disable sensor s at time t?  (yes=1)
var z{S, T} >=0, <=1;
# var which{S} >=0, <=1; # binary;

# Disabled from when?
# var dis_begin{S} >=0, <= NMAX;
# we have dis_begin[s] = NMAX if which[s] = 0.

# Coordinates of sensors (given in the data statement at the end):
param y1{S}; # = 5, 3, 8;
param y2{S}; # = 7, 5, 4;  (for sensors at (5,7), (3,5), and (8,4).


minimize detect_prob: sum{t in T} ( 1  - ( prod{s in S} (1 - (1-z[s,t])*p[s,t]) ) );
                # ( 1 - (1-p[1,t])*(1-p[2,t])*(1-(1-z[t])*p[3,t]) );
                # ( 1  - ( prod{s in S} (1 - (1-z[s])*p[s,t]) ) );
                # ( 1 - (1-(1-(1-z[1])*p[1,t])*(1-(1-(1-z[2])*p[2,t])*(1-(1-z[3])*p[3,t]) );

# Start point and end point of drone trajectory:
subject to start1: x1[1]    = x1start;
subject to start2: x2[1]    = x2start;
subject to end1:   x1[NMAX] = x1end;
```

```
subject to end2:   x2[NMAX] = x2end;

# Max-norm constraints for velocity:
#subject to vel1a{t in T2}: -0.01 <= (x1[t] - x1[t-1]) / (flighttime/NMAX) <= 0.01;
#subject to vel2a{t in T2}: -0.01 <= (x2[t] - x2[t-1]) / (flighttime/NMAX) <= 0.01;
#
# Euclidean norm constraints for velocity:
subject to velocity{t in T2}:
        (x1[t] - x1[t-1])^2 + (x2[t] - x2[t-1])^2 <= 0.001 * (flighttime/NMAX)^2;
# Here we have used forward Euler to approximate \dot{x}
# and bound either \Vert \xdot \Vert_2 or \Vert \xdot \Vert_{\infty}.

# Couple probabilities of detection by sensor s with location of drone, in each time step t:
subject to prob{s in S, t in T}: p[s,t] = 1 / ( (x1[t] - y1[s])^2 + (x2[t] - y2[s])^2 );
# looks like this can be easily extended to moving sensors:  y1[s, t] instead of y1[s].

# How much energy do we have to disable?
# In each time step not more than one sensor:
subject to disable1{t in T}: sum{s in S} z[s, t] <= 1;
# Thirty time steps in total:
subject to disable2: sum{s in S, t in T} z[s, t] <= 30;

data;

param y1 :=
1 5
2 3
3 8
4 7
5 1
6 3
7 6
8 6;
param y2 :=
1 7
2 5
3 4
4 7
5 5
6 1
7 2
8 5;
# Starting guesses:
let {s in S, t in T} z[s,t] := 0;
let {t in 71..101} z[1,t] := 1;
#
let {t in T} x1[t] := (t/NMAX) * (x1end-x1start);
let {t in T} x2[t] := (t/NMAX) * (x2end-x2start);
let flighttime := 500;
let {s in S, t in T} p[s,t] := 1 / ( (x1[t] - y1[s])^2 + (x2[t] - y2[s])^2 );

end;
```

Appendix C

Discrete Approach - Python Code Listing by Jaroslav Fowkes

```
import sys
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from mosek.fusion import *

# Spatial Discretisation Graph

# graph parameters
na = 2 # number of agents
nvx = 5 # number of vertices in x
nvy = 5 # number of vertices in y
nt = 5 # number of time periods
# generate graph
G = nx.grid_2d_graph(nvx,nvy)
plt.figure(figsize=(5,5))
pos = {(x,y):(y,-x) for x,y in G.nodes()}
nx.draw(G, pos=pos,node_color='lightgreen',with_labels=True,node_size=800)

# get dictionary of adjacent vertices
A = dict(G.adjacency())
#print(A)

# agent start positions and target
astart = [(3,0),(4,3)]
target = (2,2)
# colour graph nodes
node_color = []
for node in G.nodes():
    if node in astart:
        node_color.append('darkblue')
    elif node == target:
        node_color.append('red')
    else:
        node_color.append('lightgreen')

# plot agent start positions and target on graph
plt.figure(figsize=(5,5))
nx.draw(G, pos=pos,node_color=node_color,with_labels=True,node_size=800)

# sensor parameters
ns = 4 # number of sensors
O = 2 # number of sensors needed to detect agent
D = 1 # number of time steps sensor disabled for once first knocked out
```

```
45
46  # sensor knockout costs
47  B = 10.0 # total knockout budget
48  C = 5.0 # cost of single knockout
49  # sensor positions
50  spos = [(1,1),(1,3),(2,1),(2,3)]
51
52  # if sensor s can detect at vertex (v_x,v_y)
53  d = np.zeros((ns,nvx,nvy))
54  for ips, ps in enumerate(spos): # for each sensor
55      d[ips,ps[0],ps[1]] = 1 # add sensor location
56      for nps in G.neighbors(ps):
57          d[ips,nps[0],nps[1]] = 1 # add nodes adjacent to sensor
58  #print(d)
59
60  # if sensor s can be disabled at vertex (v_x,v_y)
61  K = d # same as if it can detect (for now)
62  # colour graph nodes
63  node_color = []
64  neighbours = []
65  for ps in spos:
66      neighbours += list(G.neighbors(ps))
67  for node in G.nodes():
68      if node in spos:
69          node_color.append('gray')
70      elif node in neighbours:
71          node_color.append('lightgrey')
72      else:
73          node_color.append('lightgreen')
74
75  # plot sensor positions and ranges on graph
76  plt.figure(figsize=(5,5))
77  nx.draw(G, pos=pos,node_color=node_color,with_labels=True,node_size=800)
78
79
80  # From J. Beasley's writeup (equation numbers in brackets)
81
82  M = Model('toy')
83
84  # Variable: if agent a at vertex (v_x,v_y) at time t
85  x = M.variable([na,nvx,nvy,nt], Domain.binary())
86
87  # Variable: if sensor knockout for agent a at vertex (v_x,v_y) at time t
88  alpha = M.variable([na,nvx,nvy,nt], Domain.binary())
89
90  # Variable: if sensor detection for agent a at time t
91  y = M.variable([ns,na,nt], Domain.binary())
92
93  # Constraint: intial agent positions
94  for a in range(na):
95      apos = astart[a]
96      M.constraint(x.index([a,apos[0],apos[1],0]), Domain.equalsTo(1.0))
97
98  # Constraint: each agent at one vertex (v_x,v_y) (1)
99  for a in range(na): # agents
100     for t in range(nt): # time periods
101         M.constraint( Expr.add(
102             [x.index([a,vx,vy,t]) for vx in range(nvx) for vy in range(nvy)]
103         ), Domain.equalsTo(1.0))
104
105 # Constraint: agent allowed movements (2)
106 for a in range(na): # agents
107     for i in range(nvx): # vertices in x
```

```
        for j in range(nvy): # vertices in y
            for t in range(nt-1): # time periods
                M.constraint(
                 Expr.sub(Expr.sub(x.index([a,i,j,t+1]),
                 x.index([a,i,j,t])),
                 Expr.add([x.index([a,v[0],v[1],t]) for v in A[i,j]]))
                , Domain.lessThan(0.0))

# Constraint: at least one agent reaches target (6)
M.constraint(Expr.add(
    [x.index([a,target[0],target[1],t]) for a in range(na) for t in range(nt)]
), Domain.greaterThan(1.0))

# Constraint: only permit knockouts when agent at nearby vertex (10)
M.constraint(Expr.sub(alpha,x), Domain.lessThan(0.0))

# Constraint: limit total knockout cost (11)
M.constraint(Expr.sub(Expr.mul(C,Expr.sum(alpha)),B), Domain.lessThan(0.0))

# Constraint: no agent can be detected by more than O sensors (13)
for a in range(na): # agents
    for t in range(nt): # time periods
        M.constraint(Expr.add(
            [y.index([s,a,t]) for s in range(ns)]
        ), Domain.lessThan(O-1.0))

# Constraint: sensor allowed detections (12)
for s in range(ns): # sensors
    for i in range(nvx): # vertices in x
        for j in range(nvy): # vertices in y
            if d[s,i,j] == 0: # skip empty constraints
                continue
            for a in range(na): # agents
                for t in range(nt): # time periods
                    M.constraint(Expr.sub(y.index([s,a,t]),
                     Expr.mul(d[s,i,j],Expr.sub(x.index([a,i,j,t]),
                      Expr.add([ Expr.mul(K[s,vx,vy],alpha.index([b,vx,vy,m]))
                        for vx in range(nvx) for vy in range(nvy)
                        for b in range(na) for m in range(max(0,t-D),t+1)])
                     ))
                    ), Domain.greaterThan(0.0))


# Constraint: force y to zero if either d or x are zero (14)
for s in range(ns): # sensors
    for i in range(nvx): # vertices in x
        for j in range(nvy): # vertices in y
            for a in range(na): # agents
                for t in range(nt): # time periods
                    M.constraint(Expr.sub(y.index([s,a,t]),
                    Expr.mul(d[s,i,j],x.index([a,i,j,t]))
                    ), Domain.lessThan(0.0))

# Set the objective function to minimize target times (8)
M.objective(ObjectiveSense.Minimize, Expr.add(
    [Expr.mul(t,x.index([a,target[0],target[1],t])) for a in range(na) for t in range(nt)]
))

# Set solver random seed (to its default value)
M.setSolverParam("mioSeed", 42)
# Set max solution time
M.setSolverParam('mioMaxTime', 180.0)
# Set max relative gap (to its default value)
```

```
M.setSolverParam('mioTolRelGap', 1e-4)
# Set max absolute gap (to its default value)
M.setSolverParam('mioTolAbsGap', 0.0)

# Solve the problem
M.setLogHandler(sys.stdout)
M.solve()

# Visualise agent positions at each time step
def vis_agents(tt):

    # get agent positions
    apos = []
    for a in range(na): # agents
        for i in range(nvx): # vertices in x
            for j in range(nvy): # vertices in y
                if x.index([a,i,j,tt]).level()[0] == 1.:
                    apos.append((i,j))

    # get agent detections
    adet = [0]*na
    for s in range(ns): # sensors
        for a in range(na): # agents
                if y.index([s,a,tt]).level()[0] == 1.:
                    adet[a] += 1

    # colour nodes
    node_color = []
    for node in G.nodes():
        if node in apos:
            if adet[apos.index(node)] >= 0: # detected
                node_color.append('darkred')
            elif adet[apos.index(node)] == 1: # sensed (but not detected)
                node_color.append('darkorange')
            else: # unsensed
                node_color.append('darkblue')
        else:
            node_color.append('lightgreen')

    # plot graph
    plt.subplot(1,2,1)
    nx.draw(G, pos=pos,node_color=node_color,with_labels=True,node_size=800)

# Visualise sensor states at each time step
def vis_sensors(tt):

    # get sensors disable actions
    sact = []
    for a in range(na):
        for i in range(nvx): # vertices in x
            for j in range(nvy): # vertices in y
                if alpha.index([a,i,j,tt]).level()[0] == 1.:
                    sact.append((i,j))

    # get disabled sensors (if any)
    sdis = []
    for ps in sact:
        for s in range(ns):
            if K[s,ps[0],ps[1]] == 1:
                sdis.append(spos[s])

    # colour graph nodes
    node_color = []
```

```
234         active_neighbours = []
235         for ps in set(spos)-set(sdis):
236             active_neighbours += list(G.neighbors(ps))
237         for node in G.nodes():
238             if node in sdis: # node disabled
239                 node_color.append('black')
240             elif node in spos:
241                 node_color.append('gray') # node enabled
242             elif node in active_neighbours:
243                 node_color.append('lightgrey')
244             else:
245                 node_color.append('lightgreen')
246 
247         # plot sensor positions and ranges on graph
248         plt.subplot(1,2,2)
249         nx.draw(G, pos=pos,node_color=node_color,with_labels=True,node_size=800)
250 
251     time = 0
252     plt.figure(figsize=(12,6))
253     vis_agents(time)
254     vis_sensors(time)
255 
256     time = 1
257     plt.figure(figsize=(12,6))
258     vis_agents(time)
259     vis_sensors(time)
260 
261     time = 2
262     plt.figure(figsize=(12,6))
263     vis_agents(time)
264     vis_sensors(time)
265 
266     time = 3
267     plt.figure(figsize=(12,6))
268     vis_agents(time)
269     vis_sensors(time)
270 
271     time = 4
272     plt.figure(figsize=(12,6))
273     vis_agents(time)
274     vis_sensors(time)
```

Bibliography

[1] Marcos de Sales Guerra Tsuzuki, Thiago de Castro Martins, and Fábio Kawaoka Takase. "ROBOT PATH PLANNING USING SIMULATED ANNEALING". In: *IFAC Proceedings Volumes* 39.3 (2006). 12th IFAC Symposium on Information Control Problems in Manufacturing, pp. 175–180. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20060517-3-FR-2903.00105. URL: http://www.sciencedirect.com/science/article/pii/S1474667015358250.

[2] Marianna De Santis et al. "Solving Multiobjective Mixed Integer Convex Optimization Problems". In: *SIAM Journal on Optimization* 30.4 (2020), pp. 3122–3145. DOI: 10.1137/19M1264709. eprint: https://doi.org/10.1137/19M1264709. URL: https://doi.org/10.1137/19M1264709.

[3] A. Dolgui and J.M. Proth. *Supply Chain Engineering: Useful Methods and Techniques*. Springer London, 2010. ISBN: 9781849960175. URL: https://books.google.co.uk/books?id=A8QzC6wE-KsC.

[4] P. E. Farrell, Á. Birkisson, and S. W. Funke. "Deflation Techniques for Finding Distinct Solutions of Nonlinear Partial Differential Equations". In: *SIAM Journal on Scientific Computing* 37.4 (2015), A2026–A2045. DOI: 10.1137/140984798. eprint: https://doi.org/10.1137/140984798. URL: https://doi.org/10.1137/140984798.

[5] O.T. Kosmas and D.S. Vlachos. "Simulated annealing for optimal ship routing". In: *Computers & Operations Research* 39.3 (2012), pp. 576–581. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2011.05.010. URL: http://www.sciencedirect.com/science/article/pii/S0305054811001341.