POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Distributed algorithms for autonomous target search problems

Relatori
Prof. Fabio Fagnani
Prof.ssa Sara Bernardini
Prof. Nicola Gatti

Candidato
Michele Da Re
263222

Anno Accademico 2019-2020

Contents

The trick, of course, is to be more interested in solving problems than in having solved them [...] Much like an avid reader never wants to reach the end of the book.

François Chollet

Chapter 1

Introduction

With the recent development of autonomous vehicles, such as drones, it is increasingly important to investigate their capability of acting in full autonomy and undertaking intelligent decisions in front of complex problems. One of the most interesting challenges arising is that of organizing a research. From a practical standpoint, this problem arises for example in law enforcement operations, in the search of missing people during natural disasters or shipwrecks, in management of wildlife and many other settings. From an abstract standpoint, the problem of teaching a machine how to look for something is interesting because in such a problem we must try to imagine where the target could be or could be going, while at the same time quickly organizing a search plan adapted to several obstacles and limited resources. In particular, the focus here is on this latter issue: not unlike humans, machines are limited, they cannot look everywhere, and therefore they must intelligently pick between conflicting tasks.

The broad category of problems that we consider is that of Search-and-Tracking (SaT), which is the name given to the problem of monitoring the position of a moving target from the point of view of an autonomous agent. Such an agent is typically a vehicle provided with powerful sensors for detection and that does not receive instructions from a centralized entity nor from a human intelligence. The task of monitoring the target's position of typically thought as composed of two distinct phases. The "tracking" phase refers to the part where the autonomous agent is continuously detecting the target and is simply following its motion. On the other hand, the "search" phase refers to the part that starts whenever the target is lost, for example because of imperfect sensors or physical obstacles. In this phase the agent must decide where to search for the target, resuming the tracking phase whenever the target is found again. Of course, the most challenging phase of the problem from the point of view of research on Artificial Intelligence is the search phase, where the agent must be able to take intelligent decision quickly.

This work is a development on a research effort carried by professor Sara Bernardini of Royal

Holloway University of London and her collaborators. The research has been presented by means of several papers, the most important of which is *Autonomous Target Search with Multiple Coordinated UAVs* (Piacentini, Bernardini, Beck, 2018) [1]. This project considers a scenario where the autonomous agent (as in most practical applications) is composed by one or more "Unmanned Aerial Vehicles" or UAVs, commonly known as 'drones'. The target, on the other hand, is a vehicle that moves on the ground, for example a car that is bound to move along a road network across a large geographical area such as central Scotland or northern England. Then, Search-and-Tracking is formulated as a combinatorial optimization problem. It is assumed that the target does not have an evasive behaviour, meaning that it does not try to hide, disguise its intentions or trick the UAVs, but simply travels towards an unknown destination along the shortest route available. When the target is lost and the search phase begins, this assumptions allows the UAVs to partially forecast the future position of the target and to select a certain number of locations, throughout the map, that cover well the possible paths. The motion of the UAVs is decomposed into two kind of actions: the action of flying between two of these locations and the action of performing the actual search, which employs standard flight patterns such as spirals and lawnmowers. The area centred into the selected position and covered by the flight pattern is called a *search pattern*. This model effectively discretizes space, since now the problem consists in finding a sequence of search patterns, known as a *plan skeleton*, that is time-feasible and that maximizes a performance function. Moreover, by assuming an a priori distribution on the destination of the target it is possible to find an approximate formula for the probability of success and another for its expected time.

Of course, the complexity of this optimization problem largely lies in the constraints that bound a subset of search patterns to be time feasible in order to be a candidate search pattern. For example, consider the case of a single UAV. Choosing to include a search pattern in the candidate solution means having to choose when the search in that location starts. There needs to be time to fly there, it needs to be within the time window when the target might actually be in that area, and it prevents many other search patterns to be chosen, either because they would have to start during the search or because there would not be time to fly there anyway. The contribution developed by this work is a new algorithm for building candidate search plans that satisfy these time constraints. The algorithm is referred to as "Distributed Algorithm" because the problem of finding a feasible plan skeleton is radically rephrased within a context of network games and best response dynamics. In this context, there are nodes that "play" a game with other nodes over time, in such a way that the overall configuration can be mathematically described and exploited. This is the meaning that "distributed" assumes here - the computation in itself is not distributed, but the desired effect is obtained by means of the dynamic behaviour of a whole network. This mathematical part was developed with the decisive contribution of the main tutor, professor Fabio Fagnani of Politecnico di Torino, and the oversight of professor Nicola Gatti of

Politecnico di Milano.

The algorithm has been implemented and the code was successfully integrated with the rest of the project previously developed, working for a span of three months at Royal Holloway University in London. This has allowed to test the solver on working simulations, to work extensively on the tuning of parameters and to compare performances with other solvers that were already deployed.

Chapter 2

Mathematical background

In this chapter, we will define the mathematical theory behind the algorithm. This requires to define network games, that are games played by nodes in a graph, and the dynamics known as Noisy Best Response, which defines a Markov chain with known stationary distribution. Moreover, the chapter will describe how a network game can be designed to solve problems over networks, and the idea is applied to the problem of finding maximal independent sets of a graph.

2.1 Notions of graph theory

In this section, a formal definition for the concept of 'graph' is provided, along with notions of graph theory and with the nomenclature that is needed in the rest of this work. Most importantly, the core concepts of independent sets and maximal independent sets are defined.

A graph is a mathematical structure that models pairwise relations between discrete objects. In its most general definition,

Definition 1 (Graph)**.** *A graph $\mathcal{G}$ is a triple*

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$$

where

- *$\mathcal{V}$ is a countable set called the set of 'nodes';*
- *$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is called the set of 'edges';*
- *$W \in \mathbb{R}_+^{\mathcal{V}\times\mathcal{V}}$ is called the 'weight matrix' and has the property that*

$$W_{ij} > 0 \iff (i, j) \in \mathcal{E}.$$

In a graph, the nodes $\mathcal{V}$ model distinct objects and the edges $\mathcal{E}$ model pairwise relations between them. $\mathcal{V}$ is usually denoted, when finite, by $\mathcal{V} = \{1, \dots, n\}$ with $n = |\mathcal{V}|$. In general, edges may represent a directed relation and may be associated with a nonnegative weight, which is modelled by the elements of the matrix W. In the present work, however, edges always represent an unweighted and undirected link. This means that edges are unordered pairs (i.e. $(i, j) \in \mathcal{E} \iff (j, i) \in \mathcal{E}$), that W is symmetric and that its elements are either 1 or 0. In this case, W is also called the *adjacency matrix*. Finally, for a node $i \in \mathcal{V}$ we define the set of its *neighbours* N_i as the set of nodes that i shares an edge with:

$$N_i = \{j \in \mathcal{V} \,:\, (i, j) \in \mathcal{E}\}.$$

If the graph was directed, we should distinguish out-neighbours and in-neighbours, but otherwise it is not necessary.

We now define independent sets and maximal independent sets of a graph.

Definition 2 (Independent set)**.** *An independent set of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ is a nonempty subset of its nodes $I \subseteq \mathcal{V}$ such that no pair of nodes in I shares an edge:*

$$\forall\, i, j \in I, \qquad (i, j) \notin \mathcal{E}.$$

Consequently, if I is an independent set then each edge in $\mathcal{E}$ has at most one endpoint in I. Then, we consider maximal independent sets. These are independent sets which cannot be possibly expanded by adding new nodes:

Definition 3 (Maximal independent set)**.** *A maximal independent set I of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ is an independent set that is not a proper subset of any other independent set I'.*

There are two more equivalent definitions of a maximal independent set that will be useful, despite seeming trivial. The first one is the following:

Proposition 1. *$I \subseteq \mathcal{V}$ is a maximal independent set if and only if it is nonempty and*

- *no pair of nodes in I shares an edge:*

$$\forall\, i, j \in I, \qquad (i, j) \notin \mathcal{E};$$

- *if any new node is added to I, the resulting set is no longer an independent set:*

$$\forall\, i \in \mathcal{V} \setminus I, \qquad I \cup \{i\} \text{ is not an independent set.}$$

The first requirement is the very definition of an independent set, while the second requirement simply ensures that I is not a proper subset of any other independent set. Another reformulation of the definition is the following:

Proposition 2. *$I \subseteq \mathcal{V}$ is a maximal independent set if and only if it is nonempty and*

- *whenever a node belongs to I, none of its neighbours does:*

$$\forall\, i \in I, \qquad N_i \cap I = \emptyset;$$

- *whenever a node does not belong to I, at least one of its neighbours does:*

$$\forall\, i \in \mathcal{V} \setminus I, \qquad N_i \cap I \neq \emptyset.$$

Similarly, the first requirement is again the definition of an independent set and the second ensures maximality. In fact, if neither i nor any of its neighbours were in I, we could add i and obtain a larger independent set. The advantage of this last definition is that it states a node-by-node requirement for a subset I of the nodes $\mathcal{V}$ to be independent maximal, which will turn out to be useful.

It is to be remarked that the notion of maximal independent set is only loosely related to that of cardinality. To explore this, consider the family $\mathcal{I}$ of all independent sets of a graph and assume that $I_{max} \in \mathcal{I}$ is an independent set of maximum cardinality:

$$\forall I' \in \mathcal{I}, \quad |I_{max}| \geq |I'|$$

where the symbol $|\cdot|$ denotes cardinality of a set. Since there is no independent set that is larger, I_{max} is also a maximal independent set. However, the converse does not hold: there could be some maximal independent set I' with smaller cardinality $|I'| < |I|$.

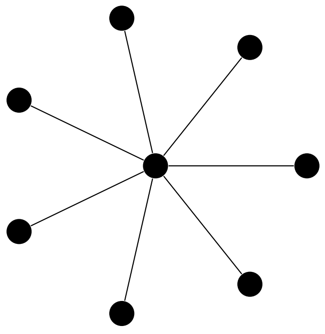

Figure 2.1: The graph S_7.

The most clear example of this fact is the star graph $S_n = \{0, 1, \ldots, n\}$ depicted in Figure 2.1 with $n = 7$. This graph has a root node, which we will call 0, and n leaf nodes, for a total of $n + 1$ nodes. In this graph, the family of independent sets $\mathcal{I}$ is given by the set $\{0\}$ and by any nonempty set of leaves, for a total of $|\mathcal{I}| = 2^n$ independent sets. The set $\{1, \ldots, n\}$ consisting of all the leaves is the independent set of largest cardinality and is also maximal independent. However, there is also another maximal independent set, and that is the set $\{0\}$ consisting of only the root node.

2.2 Network games

In this chapter, the concept of 'network game' is defined, as well as the concept of 'best response dynamics' and its variation, the noisy best response dynamics. They allow to describe the evolution over time of the network game as a Markov chain and to study its stationary distribution. Although this chapter is apparently unrelated to the notions of maximal independent set defined in the previous chapter, the link will soon be clear.

In the context of this work, a *strategic form game* is a mathematical construction that models the choice of actions by conflicting players. The model consists of players who must choose an action with the objective of maximizing their own payoff, modelled by an utility function. The players may have distinct action sets available and distinct utility functions. These depend potentially on all the choices of all players and the players have perfect knowledge of what is at stake, i.e. how all the payoffs depend on the chosen actions. If the game attempts to model a real-world conflict, it is assumed that the utility function embodies all that interests the player, meaning that the players may not be 'rational' beings as long as their irrational preferences are captured by the utility functions. However, this is not the case of this work, where the concept of network game is an algorithmical device that does not model an actually existing dynamics.

Mathematically, the definition of a strategic form game that will be used is the following:

Definition 4 (Strategic form game)**.** *A strategic form game is defined as a triple* $(\mathcal{V}, \{\mathcal{A}_i\}_{i\in\mathcal{V}}, \{u_i\}_{i\in\mathcal{V}})$ *where*

- $\mathcal{V}$ *is a the set of 'players';*
- $\mathcal{A}_i$ *is the set of 'actions' available to the player* $i \in \mathcal{V}$*;*
- *The set of all possible configurations of action choices is denoted by*

$$\mathcal{X} = \prod_{j\in\mathcal{V}} \mathcal{A}_j$$

  and it called the 'configuration set';
- u_i *is the 'utility functions' of player* $i \in \mathcal{V}$*,*

$$u_i : \mathcal{X} \longrightarrow \mathbb{R}.$$

In this work, however, all players will choose from the same set of actions, so that

$$\mathcal{A}_i = \mathcal{A} \qquad \forall i \in \mathcal{V}$$

and consequently the configuration set is $\mathcal{X} = \mathcal{A}^{\mathcal{V}}$. A configuration $x \in \mathcal{X}$ is an assignment of an action to each player: we denote by x_i the action of node i and, with some notational ease, we denote by x_{-i} the actions of the other nodes. We also define the set of *best response* as follows:

Definition 5 (Best response set)**.** *Given a strategic form game* $(\mathcal{V}, \mathcal{A}, \{u_i\}_{i \in \mathcal{V}})$*, we define the 'best response set' of player* i *to the actions* x_{-i} *as*

$$B_i(x_{-i}) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}(u_i(a, x_{-i})).$$

The best response set is made of those actions that allow to maximize player i's utility in response to the observed actions x_{-i} from other players.

A category of games with peculiar properties is the following:

Definition 6 (Potential games)**.** *A game is called 'potential' if there exist a 'potential function' from the configuration set to* $\mathbb{R}$*,*

$$\Phi : \mathcal{X} \longrightarrow \mathbb{R},$$

such that, whenever two configurations x *and* y *differ in only one component (for example the action of player* i*),* Φ *maps these configurations to points whose difference is equal to that between the two utilities:*

$$\forall x, y \in \mathcal{X} \quad s.t. \quad x_{-i} = y_{-i} \quad for\ some\ i \in \mathcal{V}, \quad \Phi(y) - \Phi(x) = u_i(y) - u_i(x).$$

In a context where players change their action one at the time, changes in the potential function will always be equal to the changes in the utility of the player who just switched action. A potential function can be loosely interpreted as an assessment of the overall state of the game. For example, if all the players always choose an action that increases their utility, then the potential function is always increasing. The existence of a potential function for a given game, however, is in general not granted.

Before exploring further this dynamics, we impose a network structure on the game by essentially requiring that the utility functions depend on what only some of the other players do, i.e. that players are like nodes in a graph and their utilities depend only on their own and their neighbours' actions:

Definition 7 (Network game)**.** *A network game for a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ *is defined by considering the nodes* $\mathcal{V}$ *as the set of players, a set of actions* $\mathcal{A}$ *and by requiring that the utility functions* $\{u_i\}_{i \in \mathcal{V}}$ *depend each only on the action of* i *and those of its neighbours* $j \in N_i$*.*

We consider now a dynamics where each node in a graph is activated randomly, observes the actions of its neighbours and updates its own action according to some rule, possibly stochastic. From now on, when not otherwise specified 'random' refers to a draw from a uniform distribution.

In the **Best Response dynamics**, at each discrete time step t one node i is randomly selected and chooses the action that *maximizes* its own utility (or, if this action is not unique, one of them at random). This means that if the node chosen has action x_i, it will randomly choose an action $y_i \in B_i(x_{-i})$. The Best Response dynamics defines therefore a discrete-time Markov chain on the configuration space $\mathcal{X}$, where the transition probability p_{xy} between distinct configurations x and y is the following:

- if x and y differ in more than one entry, then $p_{xy} = 0$;
- otherwise, if they differ at most for entry i (that is if $x_{-i} = y_{-i}$),

$$p_{xy} = \frac{1}{n} \cdot \begin{cases} 0 & \text{if } y_i \notin B_i(x_{-i}) \\ \dfrac{1}{|B_i(x_{-i})|} & \text{if } y_i \in B_i(x_{-i}). \end{cases}$$

Here, the factor $\frac{1}{n}$ is due to the fact that first node i needs to be the one that is selected between the n possible nodes.

As mentioned, if a potential function Φ exists and players always increase their utility, for example following the Best Response dyamics, then clearly the potential function increases too. More specifically, if the network game is potential than the potential function Φ is non-decreasing on the trajectories of this Markov chain. Intuitively, this means that the chain will eventually arrive in some region that it will never escape, wandering between configurations with a constant value of Φ. This value will not necessarily be the maximum, however the set $\operatorname{argmax}(\Phi)$ will surely be such a region. This amounts to proving (see [3] for more details) that the Markov chain has a trapping subset, that it is reached in finite time and that it contains the set $\operatorname{argmax}(\Phi)$.

This work, however, will involve a slightly different dynamics known as Noisy Best Response. In this dynamics, some noise is introduced in the choice of the action. This implies that, although the chain still tends to increase its potential function and thus to be "attracted" by the once-trapping region, it can also escape it and in general wander more freely. The exact dynamics is the following. Let $x \in \mathcal{X}$ be the current configuration and let $i \in \mathcal{V}$ be the randomly selected node. The node has currently action x_i and possible actions $\mathcal{A}$. The node will choose its next action $a \in A$ sampling from the distribution

$$f(a) = \frac{1}{Z_\eta} e^{\eta u_i(a, x_{-i})} \qquad \forall a \in \mathcal{A}$$

where Z_η is the normalising constant, given by

$$Z_\eta = \sum_{b \in \mathcal{A}} e^{\eta u_i(b, x_{-i})}.$$

The higher the utility of a, given by $u_i(a, x_{-i})$, the more likely a is chosen, but nevertheless all actions have a nonzero probability. The parameter $\eta > 0$ can be interpreted as a tuning variable whose inverse is a measure of noise. In fact, when η approaches $+\infty$ the distribution f also approaches a uniform distribution over the best response set $B_i(x_{-i})$. Therefore, the dynamics approaches the Best Response dynamics previously mentioned. On the other hand, when η approaches 0 the distribution also approaches a uniform distribution over $\mathcal{A}$. This corresponds to a completely noisy dynamics.

The dynamics that has been described defines a discrete-time Markov chain on the configuration space $\mathcal{X}$ and the transition rates p_{xy} are the following:

- if x and y differ in more than one entry then $p_{xy} = 0$;
- otherwise, if they differ only for entry i ($x_i \neq y_i$ and $x_{-i} = y_{-i}$),

$$p_{xy} = \frac{1}{n} \cdot \frac{1}{Z_\eta} e^{\eta u_i(y_i, x_{-i})} = \frac{1}{n} \cdot \frac{e^{\eta u_i(y_i, x_{-i})}}{\sum_{a \in \mathcal{A}} e^{\eta u_i(a, x_{-i})}}$$

  where again $\frac{1}{n}$ refers to the fact that first node i needs to be the chosen one.

It must be noted that, unless x and y differ in more than one component, p_{xy} is never zero. This implies for example that if the chain is in x than it can reach any other configuration y by changing one component at the time, which always has a nonzero probability of happening. Therefore, all configurations are globally reachable and they can reached in at most $n = \mathcal{V}$ steps. Moreover, if the network game is potential and $\eta > 0$, then the Markov chain of the Noisy Best Response dynamics enjoys several properties. Most importantly, the invariant distribution can be explicitly written, and is strongly related to the value of Φ.

Theorem 1 (Stationary distribution of the Noisy Best Response)**.** *Let* $(\mathcal{V}, \mathcal{A}, \{u_i\}_{i \in \mathcal{V}})$ *be a network game on a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$*, with* $\mathcal{V}$ *and* $\mathcal{A}$ *both finite. Let* Φ *be a potential function and let there be a Markov Chain defined by the Noisy Best Response as previously specified. Then, the Markov chain is irreducible, reversible and has the following invariant probability distribution, where* Z_η *is the normalising constant:*

$$\pi_x = \frac{1}{Z_\eta} e^{\eta \Phi(x)} \qquad \forall x \in \mathcal{X}$$

Proof. $\mathcal{X}$ is finite and, given two configurations $x, y \in \mathcal{X}$, the chain has a nonzero probability of going from one to another by changing one component at a time. Therefore, the Markov chain is irreducible and it has a unique stationary probability distribution π. Consider $x, y \in \mathcal{X}$ such that they differ in exactly one entry: $x_{-i} = y_{-i}, x_i \neq y_i$. Then,

$$\begin{aligned}
\frac{p_{xy}}{p_{yx}} &= \frac{e^{\eta u_i(y_i, x_{-i})}}{e^{\eta u_i(x_i, y_{-i})}} \\
&= e^{\eta(u_i(y_i, x_{-i}) - u_i(x_i, x_{-i})} \\
&= e^{\eta(\Phi(y_i, x_{-i}) - \Phi(x_i, x_{-i}))} \\
&= \frac{e^{\eta \Phi(y_i, x_{-i})}}{e^{\eta \Phi(x_i, x_{-i})}}
\end{aligned}$$

which implies

$$e^{\eta\Phi(x_i,x_{-i})}p_{xy} = e^{\eta\Phi(y_i,x_{-i})}p_{yx}.$$

Since this trivially holds even when $x = y$ and when they differ in more than one entry, it holds for any $x, y \in \mathcal{X}$. This implies that the chain is reversible, and that its unique invariant probability distribution is exactly

$$\pi_x = \frac{1}{Z_\eta}e^{\eta\Phi(x)} \qquad \forall x \in \mathcal{X}.$$

□

It must be noted that the "most likely" configurations (those with highest probability of being visited by the chain) are exactly those in argmax(Φ) and that the larger η is, the most sharp their probability "peaks" in π are. As it was previously remarked, the edge cases are $\eta \to 0$, where π approaches a uniform distribution on $\mathcal{X}$, and $\eta \to \infty$, where π approaches a uniform distribution on argmax(Φ).

In conclusion, if a network game is defined such that it is potential, then the Noisy Best Response dynamics defines a Markov chain whose unique invariant distribution π_x can be explicitly computed and that depends on the configuration x only via the value of the potential function $\Phi(x)$. Moreover, tuning of η allows to morph this invariant distribution from the uniform distribution over $\mathcal{X}$ ($\eta \longrightarrow 0$) to the uniform distribution over the maxima of Φ ($\eta \longrightarrow +\infty$). This theoretical framework helps to describe the behaviour of this Markov chain and it will be applied to a specific game, where we are interested in sampling from the set argmax(Φ). It must be noted, at this point, that in implementing the noisy best response we face a tradeoff on the noise parameter η. On one hand, the interest on argmax(Φ) suggests that a large η should be used, such that the limit distribution samples from that set. However, for large η the transient is extremely long: the chain is very greedy on maximizing Φ and it takes a long time to escape the trapping subsets. On the other hand, low values of η are also not recommended, since the limit distribution approaches a uniform sampling over $\mathcal{X}$. Therefore, it will be necessary to tune η on an intermediate value.

2.3 Game theoretic approach to problems over networks

This section will show how some problems over networks can be approached by considering an appropriate network game and by sampling from its Noisy Best Response dynamics. The outline of the idea is the following. Suppose that the problem can be reframed as assigning a value to each node in the network in such a way that it abides by some constraints. Many problems can be reframed in this way, for example the k colouring problem. Then, it is possible to write a function whose maxima are exactly the solutions, and that it is also the potential function of an appropriate network game. Therefore, we resort to the Noisy Best response Markov chain outlined in the previous chapter to sample from these maxima.

The formal derivation is the following. Consider a problem where there is a set of variables $\mathcal{V} = \{1, \dots, n\}$, each with a domain $\mathcal{A}_i$, and where there exists a family of subsets of $\mathcal{V}$ denoted by $\mathcal{F} \subseteq 2^{\mathcal{V}}$. The problem is the following: for each family $F \in \mathcal{F}$ there is a *constraint*, i.e. a set of valid configurations

$$C_F \subseteq \prod_{i \in F} \mathcal{A}_i$$

and it is desired to find a global configuration $x \in \mathcal{X} = \prod_{i \in \mathcal{V}} \mathcal{A}_i$ such that each individual constraint is satisfied.

Example. An example is the classic k-colouring problem, where the nodes of a graph must be painted with k colours such that there are no nodes with the same colour sharing an edge. In this setting, the colours $\mathcal{A} = \{1, \dots, k\}$ are actions that each node must choose from. The requirement that neighbours have different colours can be written as a series of pairwise constraints requiring their actions to be different. In this case, the subsets subject to constraints are exactly the edges ($\mathcal{F} = \mathcal{E}$) and the constraints are defined by

$$C_{(i,j)} = \{(x_i, x_j) \in \mathcal{A} \times \mathcal{A} \,:\, x_i \neq x_j\} \qquad \forall (i,j) \in \mathcal{E}$$

meaning that regarding $\{(i,j)\}$, a valid configuration is a configuration where $x_i \neq x_j$.

For a single constraint C_F, define by $x_{|F}$ the current configuration restricted to F and by

$$\Phi^F(x) = -\mathbb{1}_{x_{|F} \notin C_F}$$

the negative indicator function of C_F being violated. Then, the "state" of the problem can be described by a function that counts with a negative sign how many constraints are violated:

$$\Phi : \prod_{i \in \mathcal{V}} \mathcal{A}_i \longrightarrow \mathbb{R}$$

given by

$$\Phi(x) = \sum_{F \in \mathcal{F}} \Phi^F(x) = -\sum_{F \in \mathcal{F}} \mathbb{1}_{x_{|F} \notin C_F}.$$

Therefore, it is clear that in a configuration $x \in \mathcal{X}$ all constraints are satisfied if and only if $\Phi(x) = 0$ and that since $\Phi(x) \leq 0$, the solutions to be problem are exactly the set $\text{argmax}(\Phi)$. Moreover, utility functions can be written such that the resulting network game is potential with potential function Φ. For node i, such utility is the negative number of violated constraints among those that involve i:

$$u_i(x) = \sum_{F \ni i} \Phi^F(x) = -\sum_{F \ni i} \mathbb{1}_{x_{|F} \notin C_F}.$$

In plain words, this has the following interpretation. Each family $F \in \mathcal{F}$ must abide by some constraint. Then, $\Phi(x)$ is the negative number of families in violation and $u_i(x)$ is the negative number of families in violation involving i. Such a network game is also potential: in fact, for any $x, y \in \mathcal{X}$ with $x_{-i} = y_{-i}$ it holds that

$$\begin{aligned} \Phi(y) - \Phi(x) &= \sum_{F \in \mathcal{F}} \left[\Phi^F(y_{|F}) - \Phi^F(x_{|F})\right] \\ &= \sum_{F \ni i} \left[\Phi^F(y_{|F}) - \Phi^F(x_{|F})\right] \\ &= u_i(y) - u_i(x) \end{aligned}$$

where the second equality holds because $x_{-i} = y_{-i}$ implies that a summation over all families F can be restricted to a summation over those involving i.

Observation 1. These utility functions $u_i(x)$ define a network game, but one that is not played on the original graph. In fact, $u_i(x)$ depends on the actions of all nodes sharing a family F with i. The network game, therefore, is with respect to the network given by nodes $\mathcal{V}$ and edges

$$\mathcal{E}' = \{(i, j) \in \mathcal{V} \times \mathcal{V} \,:\, i, j \in F \text{ for some } F \in \mathcal{F} \}.$$

For example, in the next section this framework will be applied to a problem instance where these families are the "augmented neighbourhoods", that is

$$\mathcal{F} = \{N_i \cup \{i\} \,:\, \forall i \in \mathcal{V}\}.$$

Therefore, a node i's utility will depend on the action of all nodes that are in some augmented neighbourhood together with i. That is i itself, its neighbours', and their neighbours'.

Observation 2. The previous construction is actually valid not only for problems involving this kind of constraints, but for the maximization of any target functional in the form

$$\Phi(x) = \sum_{F \in \mathcal{F}} \Phi^F(x_{|F}),$$

form that does not necessarily involve the notion of constraints.

In conclusion, if we have a problem over a network that can be cast as the search of a configuration $x \in \mathcal{X}$ that does not violate any constraint C_F, then we can construct a potential network game with the feature that the feasible configurations are exactly its maxima. Moreover, as shown in the previous chapter we can define a Noisy Best Response dynamics, whose invariant distribution is $\pi_x \propto e^{\eta\Phi(x)}$ and therefore for it approximately samples from the set of feasible configurations $\mathcal{X}^* = \text{argmax}(\Phi)$.

2.4 Sampling of maximal independent sets

In this section, the theoretical framework of the previous section is applied to a specific problem instance, which will be needed later. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, the problem is to find its maximal independent sets, as defined in Section 2.1. Recall that a maximal independent set is a subset of nodes $I \subseteq \mathcal{V}$ such that

- whenever a node belongs to I, none of its neighbours does:

$$\forall\, i \in I, \qquad \in N_i \cap I = \emptyset.$$

- whenever a node does not belong to I, at least one of its neighbours does:

$$\forall\, i \in \mathcal{V} \setminus I, \qquad N_i \cap I \neq \emptyset.$$

Consider for each node the possible actions $\mathcal{A} = \{0, 1\}$, where 1 means that the node has been selected and 0 that it has not. For each node $i \in \mathcal{V}$, we define the two following constraints, matching the definition:

- $C^{(1)}_{N_i \cup \{i\}} = \{x \in \mathcal{X} : x_i = 1, x_j = 0 \quad \forall j \in N_i\}$;
- $C^{(2)}_{N_i \cup \{i\}} = \{x \in \mathcal{X} : x_i = 0, \exists j \in N_i \text{ s.t. } x_j = 1\}$.

Here, the constraints apply to the sets $N_i \cup \{i\}$, meaning that the families of nodes previously denoted by $F \in \mathcal{F}$ are $F_i = N_i \cup \{i\}$. Since they can be indexed by the nodes, for simplicity the notation might involve i instead of F_i when not ambiguous. This leads, according to the construction of the previous section, to the functions

$$\Psi^{(i)}(y) = -\mathbb{1}_{(y_{|F_i} \notin C^{(1)}_i \cup C^{(2)}_i)} = \begin{cases} 0 & \text{if } y_i = 1, \quad y_{-i} = 0 \\ 0 & \text{if } y_i = 0, \quad y_{-i} \neq 0 \\ -1 & \text{otherwise} \end{cases}$$

that signal violation of either one of those constraints. Here, y_{-i} refers to the actions in F_i excluding that of i (i.e. the actions of N_i). The potential function is

$$\Psi : \{0,1\}^{\mathcal{V}} \longrightarrow \mathbb{R}$$
$$\Psi(x) = \sum_{i \in \mathcal{V}} \Psi^{(i)}(x).$$

Again, $\Psi(x)$ counts with the negative sign how many nodes are in violation of a constraint. Clearly, $x \in \mathcal{X}$ corresponds to a maximal independent set if and only if $\Psi(x) = 0$, and the set of all independent sets is $\mathcal{X}^* = \text{argmax}(\Psi)$. As explained in the previous section, this is the potential of a network game of utilities

$$u_i(x) = \sum_{F_j \ni i} \Psi^{(j)}(x) = \sum_{j \in N_i \cup \{i\}} \Psi^{(j)}(x).$$

The utility of node i, then, is the number of its neighbours, plus itself, that are in violation of a constraint. From this definition it follows that each node has utility depending on the actions of himself, of its neighbours and of their neighbours.

Observation. It could be questioned whether it is really necessary to employ such a complex utility function, that counts the number of neighbours in violation of a constraint, instead of a more simple one that only counts if the node itself is in violation. After all, the function Φ to maximize does count simply the number of nodes in violation. The issue with this approach, however, is that Φ would no longer be a potential function of the game. In fact, a node i could change action and improve its utility while at the same time inducing some of its neighbours into violation of a constraint, therefore lowering Φ (if more than one) or leaving it unchanged (if only one). Overall, i would improve his utility while Φ does not improve. Therefore, for this kind of network game Φ is not a potential function, and all the theoretical guarantees of Section 2.2 do not hold.

Chapter 3

Autonomous target search

In recent years, the use of unmanned aerial vehicles, or UAVs, has expanded from military applications to several purposes in agriculture, commerce, scientific research and other fields. An UAV is an aircraft without a human pilot, with the capability of flying while accurately scanning the environment around and below, and possibly enjoying various degrees of autonomy ranging from complete autonomous flight to remote-controlling by a human operator. The technological push towards autonomous flying agents has significantly increased further research on the problem of autonomous flight. The present work focuses, in particular, on the problem of autonomous search for a lost target. This problem has a long history in mathematics and operation research and is relevant, for example, in surveillance and law enforcement, assisted agriculture, disaster response, and protection of wildlife.

When the problem involves a target in motion, not necessarily escaping but in general travelling towards some unknown destination, it is often cast as a "Search-And-Tracking" problem (SaT). This nomenclature means that two distinct phases are contemplated, as shown in Figure 3.1. On one hand, there is a "tracking" phase during which the observer is required to follow the target as long as its sensors are detecting it, tracking it all the way to its destination. On the other hand whenever the target is lost, for example due to imperfections in the sensors or because of obstacles on the ground, the observer starts a "search" phase. The tracking phase is usually handled by a reactive controller such as that described in [2] (Bernardini, Fox, Long, & Bookless, 2013). The search phase of course is the most challenging part of the problem, since the observer faces different sources of uncertainty and needs to quickly devise a search plan. The present work is focused on this search phase and it is based on the work of [1] (Bernardini, Piacentini, & Beck, 2019), which focuses on the decision-making process to build long-term strategies for the observers.

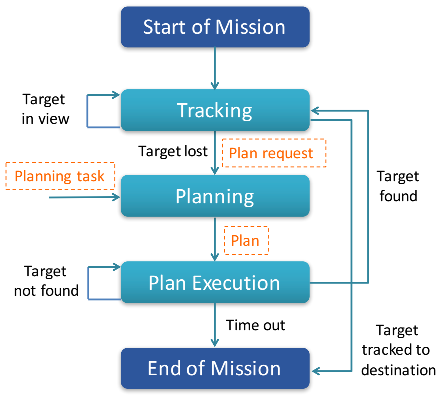

Figure 3.1: Structure of a typical SaT mission.

3.1 Construction of the optimization problem

More specifically, in professor Bernardini's work the following scenario was considered.

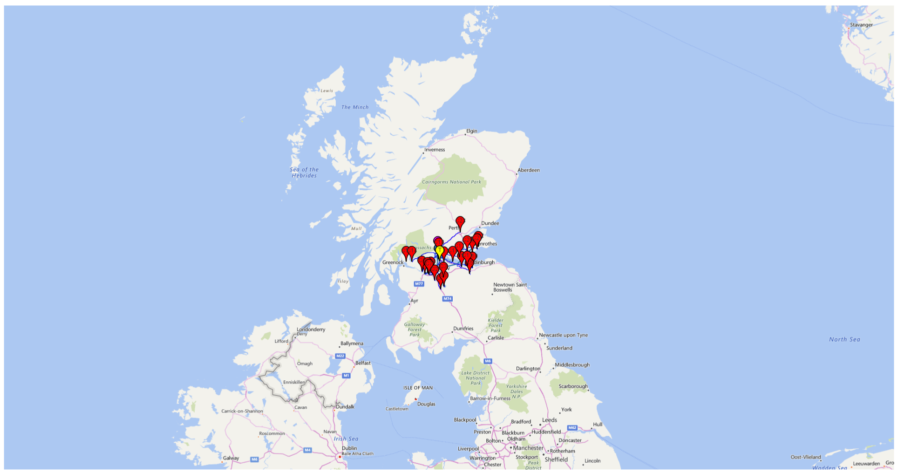

Figure 3.2: The portion of central Scotland used for the simulations. Red pins indicate possible destinations of the target.

The target moves along a road network embedded on a two-dimensional space. For the simulations that have been built, portions of the road system of central Scotland (see Figure 3.2) and northern England (see Figure 5.4) were considered. The starting point is a countryside location, north-west of Edinburgh, and the target is travelling with the goal of reaching a distant destination. One or more UAVs are tracking the target, following it closely, and at some point they lose contact, which triggers the beginning of the search phase. The UAVs exploit standard flight patterns to search for the target, such as *spirals* and *lawnmowers*.

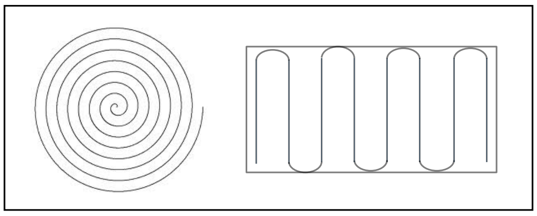

Figure 3.3: Standard search patterns: spirals (left) and lawnmowers (right).

It is also assumed that the observers have perfect knowledge of the road network and know that the target is moving towards one destination out of a set $\mathcal{D}$ of possible destinations. Each of them is associated to an a priori probability $\pi(x), x \in \mathcal{D}$, modelling the uncertainty over the real destination of the target. Furthermore, it is assumed that the target moves towards its destination along the shortest possible path, regardless of whatever the observer does. This is an important assumption, since it implies that the target has a passive behaviour and does not interact or change its actions in result of what the UAVs do. Mathematically, this means that the a priori distribution π over the destinations $\mathcal{D}$ induces an a priori probability over the paths that the target may follow, and thus an a priori probability over the locations where the target may be in at each time instant. In practice, however, further sources of uncertainty may be involved, such as the level of evasiveness and speed at which the target moves on each road. As we will see, they are all taken into account when approximating the probability distribution of the target's true location with a Monte Carlo simulation.

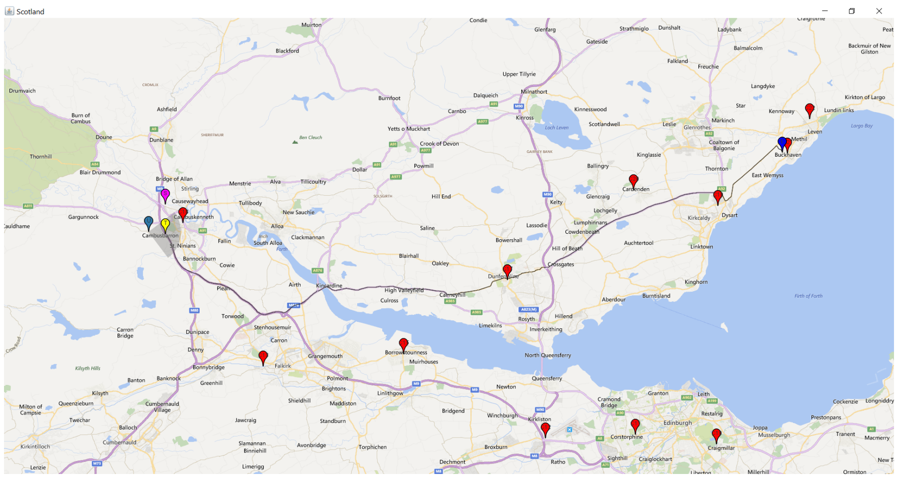

Figure 3.4: The UAV and the target during the tracking phase.

In Figure 3.4, a simulation of the tracking phase can be observed more in detail. The target T is represented by the yellow pin and it started from the source location S marked by the violet pin.

The target is travelling along the optimal route (thinly marked in blue) towards its destination E, marked by the bright blue pin on the upper-right portion of the picture. The observer UAV (U), marked by the dark blue pin near the target, flies around the source following it. A shaded cone represents the area scanned by the UAV's sensors. Finally, red pins show some of the possible destinations in $\mathcal{D}$ that the UAV contemplates.

Thanks to these assumptions on the SaT model, the search can be cast as the research for a search plan, composed of a number of locations, named "search patterns", where to execute a flight pattern (either a spiral or a lawnmower). This search plan should be such that

- the observers are actually able to execute it, i.e. there are not time constraints between the search patterns;
- a performance function, e.g. the probability of finding the target, is maximized.

Regarding the latter point, a recursive formula that gives an approximation of such probability can be computed. Regarding the former, the constraints are modelled by casting the problem as a combinatorial optimization problem. Its formal characterization is the following.

Let $\mathcal{O}$ be a set of observers, all with the same flight and sensing capabilities. Let $\mathcal{C}$ be a set of search patterns located in specific locations on the map. These search patterns are selected as follows. The search area is defined as a circular sector centred on the target's last known position and its road network is represented as a graph (V, E) built by discretizing the search area in cells with fixed side-length. Vertices V represent the cells and edges E represent pairs of adjacent cells connected by at least one road. Then, a Monte Carlo simulation of the target's motion on the graph is run. The sources of uncertainty assumed are the target's destination (represented, as mentioned, by an a priori distribution $\pi(x)$ on a set of possible destinations $x \in \mathcal{D}$), the concealment that the target is assumed to adopt resulting in sub-optimal routes, and its speed. A number of time check points $t_0 < t_1 < \cdots < t_n$ are considered and for each check point t_i the two nodes with the largest number of Monte Carlo particles at that time are considered. Candidate search patterns are obtained centering a flight pattern (spiral or lawnmower) on these nodes. Therefore, a candidate search pattern $\sigma \in \mathcal{C}$ is a physical area with a spatial extension and corresponds to an area where, at some point in time, the target has a larger probability of visiting. Each candidate search pattern σ is associated with:

- A start point and an end point. These are the exact point where the search of σ will begin and terminate, and are necessary to define the flight times between search patterns;
- A duration d_σ, given by the time the UAV needs to perform a search on the pattern;
- A time window $[t_\sigma^-, t_\sigma^+]$, given by the possible initial times of a search of σ. These are related to the time window when the target has a nonzero probability of being in the area

covered by σ. In turn, this time window is calculated by considering the distance between σ the last known position of the target, and the maximum and minimum speed that the target could have;

- A detection probability γ_σ, defined as the probability of detecting the target while searching σ conditioned on the event of the target being actually there. This detection probability encodes both the trustworthiness of the UAV's sensors and the conditions on the ground, such as the presence of obstacles like buildings or lights.

Each candidate in $\mathcal{C}$ can be executed more than once, but we can transform the problem to an equivalent version where each pattern can be executed at most once by creating k_σ copies of each search pattern σ, where k_σ is the maximum number of times it can be executed. We call $\widetilde{\mathcal{C}}$ the set of patterns obtained by this procedure.

3.2 Constraints

In order to represent the time constraints that bind execution of different search patterns, matrices are defined encoding the flight times between patterns. Let $m = |\widetilde{\mathcal{C}}|$ be the number of candidate search patterns and define for each observer $o \in \mathcal{O}$ a matrix $D^o \in \mathbb{R}_+^{(\widetilde{\mathcal{C}} \cup \{0\}) \times (\widetilde{\mathcal{C}} \cup \{0\})}$, where 0 is a symbol representing the initial position. Position $(0, \sigma_i)$ is the flight time from the initial position 0 to the start point of search pattern σ_i, with $i = 1, \ldots, m$. Position (σ_i, σ_j) is the flight time from the end point of search pattern σ_i to the start point of the pattern σ_j, with $i, j = 1, \ldots, m$.

We wish to find a subset of search patterns $\Sigma = \{\sigma_1, \ldots, \sigma_n\} \subseteq \widetilde{\mathcal{C}}$ such that the available UAVs are able to search all of them. The assignment of search patterns to the observers is formalized by requiring the existence of a partition $\mathcal{P}(\Sigma)$ with at most $|\mathcal{O}|$ elements and an assignment function $S : \mathcal{O} \longrightarrow \mathcal{P}(\Sigma)$. We call the assignment $S(o)$ for observer o a *plan*. These plans should be such that the observer is able to execute them, and this is formalized as follows. Let $S(o) = (s_1, \ldots, s_k)$ be a plan, with the search plans ordered from the earliest to the latest. Then, there should exist a time assignment function

$$\mathbf{t} : S(o) \longrightarrow \mathbb{R}_+$$

such that

- Each search plan is assigned to a time within its time window:

$$\mathbf{t}(s_i) \in [t^-_{s_i}, t^+_{s_i}] \qquad \forall s_i \in S(o);$$

- The initial search pattern can be reached:

$$\mathbf{t}(s_1) \geq D^o(0, s_1);$$

- For every two consecutive patterns s_i and s_{i+1}, between their two starting times there is enough time to search the first and to fly between them:

$$\mathbf{t}(s_{i+1}) - \mathbf{t}(s_i) \geq d_{s_i} + D^o(s_i, s_{i+1}) \qquad \forall i = 1, \ldots, k-1.$$

For a given plan, whether such a time assignment exists can be rapidly checked by the following procedure:

- Check if the first search pattern s_1 can be reached in time:

$$t^+_{s_1} \geq D^o(0, s_1).$$

  If it can be, assign it to

$$\mathbf{t}(s_1) = \max\{D^o(0, s_1), t^-_{s_1}\}$$

  which is the earliest reach time or the starting of the time window, whichever comes second;

- Similarly, for each of the following search patterns check if they can be reached in time and if they can, assign them the time instant they can be reached if inside the opportunity window, or to the initial time of the time window otherwise. Namely, if s_i was assigned time instant $\mathbf{t}(s_i)$, check whether

  $$t^+_{s_{i+1}} \geq \mathbf{t}(s_i) + d_{s_i} + D^o(s_i, s_{i+1})$$

  and if true, assign it to

  $$\mathbf{t}(s_{i+1}) = \max\{\mathbf{t}(s_i) + d_{s_i} + D^o(s_i, s_{i+1}), t^-_\sigma\}$$

  for each $i = 1, \dots, k-1$.

Given a subset $\Sigma = \{\sigma_1, \dots, \sigma_n\} \subseteq \widetilde{\mathcal{C}}$, we say that Σ is *executable* if there exist such a partition $\mathcal{P}(\Sigma)$, such an observer assignment $S : \mathcal{O} \longrightarrow \mathcal{P}(\Sigma)$ and such a time assignment function $\mathbf{t} : S(o) \longrightarrow \mathbb{R}_+$. The feasible set of this optimization problem is the set of executable plans $\Sigma \in 2^{\widetilde{\mathcal{C}}}$. However, we can actually be more precise on where the optimum will be. Given a plan skeleton Σ, we say that it is *maximal* if for any possible addition $\sigma \in \widetilde{\mathcal{C}} \setminus \Sigma$, the resulting plan skeleton $\Sigma \cup \{\sigma\}$ is never executable. As we will see, the objective function $G(\Sigma)$ will be non-decreasing: therefore, the optimum will be surely a maximal plan skeleton.

It must be remarked that in this optimization problem, the part that is actually challenging is to build feasible solutions. Compiling a search plan is the task of choosing a number of search patterns keeping track of their starting times and their feasibility. The choice of including a search pattern with a certain starting time disables completely some of the other search patterns and limits the potential starting times for others. At the same time, the choice of excluding a search pattern should compel to include at least one of the patterns that it would be in conflict with. Framed in these terms, the problem starts to resemble that of finding independent maximal sets. The exact definition of what graph is considered is stated in the next chapter. For now, we mention more broadly that a general category of planning problems can be cast in terms of independent sets: in fact, by defining a discrete set of tasks $\mathcal{V}$ and by defining edges $\mathcal{E}$ between them whenever they *cannot* be executed both, then a feasible plan corresponds to an independent set. Moreover, if the objective function is increasing its maximum is necessarily a set of tasks that no other search pattern can possibly be added to, which means a maximal independent set.

3.3 Objective function

Let $P(\Sigma)$ be the total probability of finding the target by executing the plan skeleton Σ. This probability can be explicitly computed (after a few simplifying assumptions) and can serve as an objective function to be maximized. The simplifying assumption is to regard only what locations have been searched, disregarding all the issues related to time: the initial instant where the search has started, the duration of the search, the actual time of discovery. On the other hand, the result is a recursive formula that is effective and easy to compute. The derivation of this formula is the following.

Theorem 2 (Recursive formula for the probability of finding the target)**.** *Let $\mathcal{D}$ be the set of possible target destinations and $\mathcal{D}_\sigma$ be the subset of possible target destinations compatible with the search pattern σ, i.e. such that their path passes through σ. Let $\pi(x)$, $x \in \mathcal{D}$ be the a priori probability that the target's destination is x. Furthermore define, for $k = 0, 1, \ldots, \bar{k} - 1$:*

- $P_\Sigma^{(k)}$ *as the probability that the target is found before or at step k;*
- $P_{\Sigma^*}^{(k)}$ *as the probability that the target is found at step $k+1$ while knowing that the plan has failed up to the step k;*
- $P_\Sigma^{(k)}(x)$ *as the probability that the target's destination is x, knowing that the plan has failed up to the step k (also defined for $k = \bar{k}$, unlike the previous ones);*

The total probability of finding the target by executing the plan skeleton Σ is obtained when noticing that $P(\Sigma) = P_\Sigma^{(\bar{k})}$. Then, the following recursion holds:

- $P_\Sigma^{(k)}(x) = \frac{P_\Sigma^{(k-1)}(x) \cdot \left(1 - \gamma_{\sigma_k} \mathbb{1}_{(x \in \mathcal{D}_{\sigma_k})}\right)}{1 - P_{\Sigma^*}^{(k-1)}} \qquad k = 1, \ldots, \bar{k} - 1;$
- $P_{\Sigma^*}^{(k)} = \gamma_{\sigma_{k+1}} \cdot \sum_{y \in \mathcal{D}_{\sigma_{k+1}}} P_\Sigma^{(k)}(y) \qquad k = 1, \ldots, \bar{k} - 1\ ;$
- $P_\Sigma^{(k)} = P_\Sigma^{(k-1)} + P_{\Sigma^*}^{(k-1)} \cdot \left(1 - P_\Sigma^{(k-1)}\right) \qquad k = 1, \ldots, \bar{k}.$

with initial conditions

- $P_\Sigma^{(0)}(x) = \pi(x) \qquad \forall x \in \mathcal{D};$
- $P_\Sigma^{(0)} = 0;$
- $P_{\Sigma^*}^{(0)} = \gamma_{\sigma_1} \cdot \sum_{y \in \mathcal{D}_{\sigma_1}} P_\Sigma^{(0)}(y)$

Proof. Given a plan skeleton $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{\bar{k}})$, let $\mathcal{F}_\sigma$ be the event of finding the target in the area of search pattern σ and $\overline{\mathcal{F}}_\sigma$ its negation, i.e. the event of not finding the target in σ.

Let also $\mathcal{F}_\Sigma^{(k)}$ be the event of finding the target when executing the plan skeleton Σ until step k and $\overline{\mathcal{F}}_\Sigma^{(k)}$ its negation. Denote by

$$P_\Sigma^{(k)} = \mathbb{P}[\mathcal{F}_\Sigma^{(k)}]$$

the probability of finding the target when executing Σ until step k: then, the probability of success of Σ (whose length is $\bar{k}$) is

$$P(\Sigma) = P_\Sigma^{(\bar{k})}.$$

Since we have $\mathcal{F}_\Sigma^{(k-1)} \subseteq \mathcal{F}_\Sigma^{(k)}$,

$$\mathbb{P}[\mathcal{F}_{\sigma_k}] = \mathbb{P}[\mathcal{F}_\Sigma^{(k)} \setminus \mathcal{F}_\Sigma^{(k-1)}] = P_\Sigma^{(k)} - P_\Sigma^{(k-1)}.$$

Now let $\mathcal{D}$ be the set of possible target destinations, $\bar{x} \in \mathcal{D}$ the true destination of the target and $\mathcal{D}_\sigma$ be the subset of possible destinations compatible with the search pattern σ. Let $\pi(x)$, $x \in \mathcal{D}$ be the a priori probability that the target's destination is x. Furthermore define

$$P_{\Sigma^*}^{(k-1)} = \mathbb{P}[\mathcal{F}_{\sigma_k} \,|\, \overline{\mathcal{F}}_\Sigma^{(k-1)}]$$

the probability that the plan Σ succeeds at k conditioned on the event that it has failed up to $k-1$. Since it holds that

$$\mathbb{P}[\mathcal{F}_\Sigma^{(k)}] = \mathbb{P}[\mathcal{F}_\Sigma^{(k-1)}] + \mathbb{P}[\mathcal{F}_{\sigma_k} \,|\, \overline{\mathcal{F}}_\Sigma^{(k-1)}] \cdot \mathbb{P}[\overline{\mathcal{F}}_\Sigma^{(k-1)}]$$

then we have that, re-writing this result,

$$P_\Sigma^{(k)} = P_\Sigma^{(k-1)} + P_{\Sigma^*}^{(k-1)} \cdot (1 - P_\Sigma^{(k-1)}).$$

Now, define

$$P_\Sigma(x)^{(k)} = \mathbb{P}[\bar{x} = x \,|\, \overline{\mathcal{F}}_\Sigma^{(k)}]$$

the probability that the true destination is x conditioned on the event that the plan has failed up to k. Of course, the initial condition for this is

$$P_\Sigma^{(0)}(x) = \pi(x).$$

The probability $P_{\Sigma^*}^{(k-1)}$ of finding the target at k conditioned on having failed up to $k-1$ is given by the product between the probability of the target being in σ conditioned on having failed up to $k-1$ and the probability γ_σ of the UAV detecting the target:

$$P_{\Sigma^*}^{(k-1)} = \sum_{y \in \mathcal{D}_\sigma} P_\Sigma^{(k-1)}(y) \cdot \gamma_{\sigma_k}.$$

Consider now the probability of the destination being x conditioned on the fact that the plan failed up to $k-1$ and succeeded in k:

$$\mathbb{P}[x = \bar{x} \,|\, \overline{\mathcal{F}}_\Sigma^{(k-1)} \cap \mathcal{F}_{\sigma_k}].$$

Finding the target in σ_k brings the information that the destination is compatible with σ_k: therefore, this probability is 0 whenever x is not compatible ($x \notin \mathcal{D}_{\sigma_k}$). Otherwise, it can be computed by simply conditioning the a priori $P_\Sigma^{(k-1)}(x)$ to the compatible destinations, obtaining

$$\mathbb{P}[x = \bar{x} \,|\, \overline{\mathcal{F}}_\Sigma^{(k-1)} \cap \mathcal{F}_{\sigma_k}] = \frac{P_\Sigma^{(k-1)}(x)}{\sum\limits_{y \in \mathcal{D}_{\sigma_k}} P_\Sigma^{(k-1)}(y)} \cdot \mathbb{1}_{(x \in \mathcal{D}_{\sigma_k})}.$$

Finally, we have the following equation:

$$\begin{aligned}
P_\Sigma^{(k-1)}(x) &= \mathbb{P}[x = \bar{x} \,|\, \overline{\mathcal{F}}_\Sigma^{(k-1)}] \\
&= \mathbb{P}[x = \bar{x} \,|\, \overline{\mathcal{F}}_\Sigma^{(k-1)} \cap \mathcal{F}_{\sigma_k}] \cdot \mathbb{P}[\mathcal{F}_{\sigma_k} \,|\, \overline{\mathcal{F}}_\Sigma^{(k-1)}] + \mathbb{P}[x = \bar{x} \,|\, \overline{\mathcal{F}}_\Sigma^{(k-1)} \cap \overline{\mathcal{F}}_{\sigma_k}] \cdot \mathbb{P}[\overline{\mathcal{F}}_{\sigma_k} \,|\, \overline{\mathcal{F}}_\Sigma^{(k-1)}] \\
&= \frac{P_\Sigma^{(k-1)}(x)}{\sum\limits_{y \in \mathcal{D}_{\sigma_k}} P_\Sigma^{(k-1)}(y)} \cdot \mathbb{1}_{(x \in \mathcal{D}_{\sigma_k})} \cdot P_{\Sigma^*}^{(k-1)} + P_\Sigma^{(k)}(x) \cdot (1 - P_{\Sigma^*}^{(k-1)}) \\
&= P_\Sigma^{(k-1)}(x) \mathbb{1}_{(x \in \mathcal{D}_{\sigma_k})} \cdot \gamma_{\sigma_k} + P_\Sigma^{(k)}(x) \cdot (1 - P_{\Sigma^*}^{(k-1)})
\end{aligned}$$

which implies that

$$P_\Sigma^{(k)}(x) = \frac{P_\Sigma^{(k-1)}(x) \cdot (1 - \gamma_{\sigma_k} \mathbb{1}_{(x \in \mathcal{D}_{\sigma_k})})}{1 - P_{\Sigma^*}^{(k-1)}}.$$

□

Besides this formula for the probability of detection, it is also possible to compute a similar approximate recursive formula for the expected time of detection, denoted as $T(\Sigma)$. These two performance metrics will be combined in a single objective function. The formula is the following:

Theorem 3 (Approximate recursive formula for the expected time of finding the target)**.** *Let the expected time of finding the target be approximated by the formula*

$$T(\Sigma) = \sum_{j=1}^{\bar{k}} t_{\sigma_j} \mathbb{P}[\mathcal{F}_{\sigma_j}] = \sum_{j=1}^{\bar{k}} t_{\sigma_j} (P_\Sigma^{(j)} - P_\Sigma^{(j-1)}).$$

where $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{\bar{k}})$ *and*

$$t_\sigma = \frac{t_\sigma^+ + t_\sigma^-}{2}.$$

is the mid-point of the time window $[t_\sigma^-, t_\sigma^+]$ *of search pattern* σ*. Define also*

$$T_\Sigma^{(k)} = \sum_{j=1}^{\bar{k}} t_{\sigma_j} \mathbb{P}[\mathcal{F}_{\sigma_j}].$$

Similarly to the previous theorem, we obtain the desired result by observing that, since Σ *has length* $\bar{k}$*,*

$$T(\Sigma) = T_\Sigma^{(\bar{k})}.$$

Then, the following recursion holds:

$$T_\Sigma^{(k)} = T_\Sigma^{(k-1)} + t_{\sigma_k} \cdot \left(P_S^k - P_S^{(k-1)} \right)$$

with initial condition $T_\Sigma^{(0)} = 0$ *and* $P_S^{(k)}$ *defined and given by the previous theorem.*

Proof. The proof is trivial: the result follows immediately after recalling that

$$\mathbb{P}[\mathcal{F}_{\sigma_k}] = P_S^k - P_S^{(k-1)}.$$

□

The objective function that we wish to take into account is one that involves both the probability of detection and the expected time of detection. More specifically, the function is the performance measure

$$G(\Sigma) = P(\Sigma) - k \cdot T(\Sigma).$$

Here, k is first of all a parameter regulating the tradeoff between these two conflicting requirements - maximizing the probability and minimizing the expected time. It does also have another interpretation. If we assume that whenever the detection does not succeed, it contributes to the expected time as a detection at time A very large, then the resulting expected time is

$$\hat{T}(\Sigma) = T(\Sigma) + A \cdot (1 - P(\Sigma)).$$

Then, we have that

$$-\hat{T}(\Sigma) = A \cdot P(\Sigma) - T(\Sigma) + A = A \cdot \left(P(\Sigma) - \frac{1}{A} T(\Sigma) \right) + 1.$$

Therefore, the maximization of objective function can be though as the minimization of this modified expected time of detection $\hat{T}(\Sigma)$, with $k = \frac{1}{A}$. This is meaningful as long as A exceeds all of the expected times of detection t_σ, which yields an upper bound on k:

$$A \geq t^* = \max_{\sigma \in \widetilde{\mathcal{C}}} (t_\sigma)$$

Chapter 4

The Distributed algorithm

The most challenging part of this optimization problem is, as in most combinatorial optimization problems, that of building feasible solutions. As mentioned, the task is to choose from the set of possible search patterns $\widetilde{\mathcal{C}}$ a subset Σ that is

- *executable*, meaning that it can be partitioned between to the available observers $\mathcal{O}$ and that each observer receives a plan that follows the time constraints;
- *maximal*, meaning that each additional search pattern would yield a plan that is not executable.

The time constraints are, essentially, due to the fact that the UAV must be able to arrive to each search pattern σ within its time window $[t_\sigma^-, t_\sigma^+]$.

Observe now that each search pattern can be regarded, in abstract, as a task to be executed. The constraints of the problem amount essentially to an indication of whether two tasks are conflicting. Restricting the analysis to the case of a single UAV, we wish to find a set of search patterns that presents no conflicts. Suppose then that search patterns are nodes of a graph and that two nodes are connected by an edge whenever they cannot be both selected, i.e. if they are conflicting tasks. Then, a plan with no conflicts is an independent set of this graph. Moreover, since the objective function $P(\Sigma)$ is non-decreasing, the optimum is certainly a search plan that no other search pattern can be added to. This implies that the optimum is a maximal independent set of this graph.

The main issue of this approach, however, is that the feasibility of a search plan cannot really be reduced to a local requirement on each search pattern. For a single UAV o, a search plan Σ is feasible if there exist a time assignment function

$$\mathbf{t} : S(o) \longrightarrow \mathbb{R}_+$$

such that each search plan is assigned to a time instant within its time window, and that the time constraints due to search duration and flight time are respected. For most pairs of search patterns, they either have time windows far enough that there is always time to fly and execute both, or they have time windows close enough that they are never both executable. In some crucial cases however, whether two search patterns are in conflict is dependent on their time assignments, which could or could not give the UAV enough time to fly between them. This is why we can define two search patterns to be conflicting tasks only if provided with a starting time.

This issue can be solved by creating for each search pattern $\sigma \in \widetilde{\mathcal{C}}$ a fixed number κ of nodes, where each node consists of the search pattern σ and a start time t. Possible initial times are given by the discretization of the time window $[t_\sigma^-, t_\sigma^+]$ into κ points. Therefore, σ yields the nodes

$$\begin{aligned}
&(\sigma, t_\sigma^-),\\
&(\sigma, t_\sigma^- + \frac{t_\sigma^+ - t_\sigma^-}{\kappa - 1}),\\
&(\sigma, t_\sigma^- + 2\frac{t_\sigma^+ - t_\sigma^-}{\kappa - 1}),\\
&(\sigma, t_\sigma^- + 3\frac{t_\sigma^+ - t_\sigma^-}{\kappa - 1}),\\
&\dots,\\
&(\sigma, t_\sigma^+).
\end{aligned}$$

This defines the set of nodes $\mathcal{V}$, whose cardinality is $|\mathcal{V}| = \kappa \cdot |\widetilde{\mathcal{C}}|$. On the other hand, two nodes (σ, t) and (σ', t') with $t \le t'$ are defined to have an edge connecting them if

$$t' - t < d_\sigma + D^o(\sigma, \sigma')$$

i.e. if there is not enough time between t and t' to execute the search of σ (which takes d_σ) and fly to σ' (which takes $D^o(\sigma, \sigma')$).
Regarding the other constraint, which we recall involves only the first search pattern of the sequence s_1 and asks to have enough time to reach it from the initial position 0:

$$\mathbf{t}(s_1) \ge D^o(0, s_1),$$

it can be addressed as follows. Since while building the plan the first search pattern of the sequence may change, we simply check each time whether a proposed plan respects this constraint. In practice, search pattern with the earliest windows are also closest to the UAV, therefore this constraint is not an issue.
We can now apply to this graph the network game described in section 2.4 in order to sample maximal independent sets. This amount to a solution proposal algorithm: the algorithm is agnostic with respect to the objective function and only proposes feasible and maximal solutions.

4.1 Pseudocode

The pseudocode of the algorithm is the following. Given the set of candidate search patterns $\tilde{\mathcal{C}}$,

```
1. Define the list of all nodes $\mathcal{V}$ from the set of candidate search patterns $\tilde{\mathcal{C}}$;
2. Define their edges by checking, for each pair in $\mathcal{V}$, if they comply to the condition stated in the previous section;
3. Initialize all nodes with action $x_i = 0 \quad \forall i \in \mathcal{V}$ ("not selected");
4. Initialize $x_{best} = (0, \dots, 0)$;
5. Set $T$ time limit, let $t$ be the time elapsed since the beginning of the algorithm;
6. Set $\eta$ noise parameter;
7. While $t < T$:
    (a) Sample uniformly a node $i \in \mathcal{V}$;
    (b) Compute its utilities $u_i(0, x_{-i})$ and $u_i(1, x_{-i})$;
    (c) Sample the new action $a$ from the distribution
            $\frac{1}{Z}(e^{\eta u_i(a=1,x_{-i})}, e^{\eta u_i(a=0,x_{-i})})$
        where $Z$ is the normalization constant;
    (d) Set the new action $x_i = a$;
    (e) If resulting the configuration $x$ is a maximal independent set, compute $G(x)$. If $G(x) > G(x_{best})$, set $x_{best} = x$.
8. Build from $x_{best}$ the resulting plan skeleton $S_{best}$.
```

The solution to the optimization problem is S_{best}. More precisely on the last step, a search pattern σ is part of S_{best} if there exist a node $i = (\sigma, t)$ such that $x^i_{best} = 1$.

4.2 Adjustments and heuristics

We enumerate here a few of the relevant features of the algorithm implementation. Although the basic structure of the actual code follows this pseudocode, in implementing it several slight modifications were adopted in order to reduce the number of operations needed at each iteration.

- Each node i was given an attribute $\Psi^{(i)}$ of value -1 if it is in violation and 0 otherwise. Whenever a node's action is changed, its $\Psi^{(i)}$ and its neighbours' are updated. This allows to immediately compute the utility of that action using the formula

$$u_i(a, x_{-i}) = \sum_{j \in N_i \cup \{i\}} \Psi^{(i)};$$

- The algorithm should feed to $P(\Sigma)$ possible feasible solutions. The algorithm, however, samples maximal independent sets from the graph $\mathcal{G}$ since, as explained in 2.4, the independent maximal sets are exactly given by $\operatorname{argmax}(\Psi) = \{x \in \mathcal{X} : \Psi(x) = 0\}$. Independent maximal set surely correspond to a feasible plan, but that plan is not guaranteed to be maximal. This is due to the fact that search patterns were forced to have some initial time instant: it may be impossible to add another search pattern within those choices, but it could be possible under another time assignment. On the other hand, feasibility of a sequence can be rapidly checked. Moreover, there is no guarantee that a feasible plan skeleton always can be represented by a maximal independent set. Sometimes, in fact, they cannot have their time assignments anchored to the discretization of the time window without resulting some conflicts between nodes, i.e. a set that is not independent. Therefore, the algorithm does not feed to the objective function configurations with $\Psi(x) = 0$, but configurations whose associated search plan is feasible;

- Feasibility of the new configuration x is checked only if the node's action changed;

- The objective function $G(x)$ is computed only for configurations that were not previously seen. Computation of $G(x)$ is in fact the part that includes most of the effort needed, and the algorithm very often returns to configurations already previously visited. This check is performed by keeping a register of all configurations previously seen. This might pose scaling issues, however for most practical cases (see Chapter 5.2 for a discussion on the size of problem instances) the number of feasible plans is limited to the $\approx 10^3$;

- Since the objective function has a recursive form, if the new configuration to be evaluated shares part of the search patterns with the previous one part of the computations are avoided. The recursion is developed starting from the first term that is different.

4.3 Parameter tuning

The algorithm's performance involves two important tuning parameters: the time discretization and the noise parameter η. They are relevant for different reasons:

1. **Noise**. As mentioned in section 2.2 (Network games), the utility functions involve a parameter η whose inverse is a measure of noise. For large η, the dynamics approaches the Best Response dynamics: the nodes are strongly attracted by their largest utilities, therefore the potential function Ψ rarely decreases and the Markov chain is most of the time stuck in some configuration that is a local maximum of Ψ. On the other hand, for small η the dynamics approaches a completely noisy behaviour and the Markov chain spends most of the time away from the maxima of Ψ. The optimal tradeoff between these two behaviours must be determined by experiment.

2. **Time discretization**. As it was explained, to ponder whether two search patterns are conflicting they need to be coupled to some time instant. Therefore, each search pattern had its time window discretized into κ points and κ nodes were created. The finer the time discretization is, the more accurate is the approximation of the time constraints. However, graph size is an important issue. In the instances considered, a number of $|\widetilde{\mathcal{C}}| = 30$ search patterns were considered, meaning that the graph has $|\mathcal{V}| = \kappa \cdot |\widetilde{\mathcal{C}}| = 30\kappa$ nodes and that the configuration space that the Markov chain explores has size $|\mathcal{X}| = 2^{\kappa \cdot |\widetilde{\mathcal{C}}|} = 2^{30\kappa}$ growing exponentially in κ. This is a very large configuration space, but also consider that the related graph (associated to the Markov chain) is well connected: the distance between any two configurations, in fact, cannot ever exceed $|\widetilde{\mathcal{C}}|$ because it is possible to travel between one and the other changing one action at the time. Moreover, the Noisy Best Response dynamics ensures that the chain is attracted by the most interesting regions, where $\Psi(x)$ is higher. In conclusions, small values of κ allow to quickly find all the independent sets but they correspond to poor search plans, while larger values of κ allow for the existence of better solution but make them harder to be found.

Tuning was conducted for both these parameters, averaging the results on a large number of seeds. The optimal values that were found for this problem instance are $\kappa = 5$ for the time discretization and $\eta = 2.4$ for the noise parameter.

Chapter 5

Results

In this chapter, we will compare the results of the Distributed Algorithm with those of three different solvers on a variety of settings, including different values of the tradeoff parameter k and two different geographical situations.

5.1 Other algorithms

Besides the Distributed Algorithm, other solution approaches had been used in the SaT Simulator in order to solve this optimization problem. The other algorithms are the following.

5.1.1 Greedy Algorithm

This approach involves a greedy algorithm that directly builds a solution by iteratively adding search patterns. The algorithm starts with an empty plan Σ and at each iteration finds what available search pattern $\sigma \in \widetilde{\mathcal{C}} \setminus \Sigma$ provides the greatest increase in the objective function, i.e. the greatest

$$\Delta_\sigma = P(\Sigma \cup \{\sigma\}) - P(\Sigma)$$

while the resulting plan $\Sigma \cup \{\sigma\}$ remains executable. The algorithm stops when no other search pattern can be added, thus ensuring that the solution is also maximal. Such a greedy algorithm performs well on this problem due to the properties of the objective function. In fact, the function is submodular and non-decreasing as proven in [4] (Piacentini, Fagnani, & Bernardini, 2020), and this fact provides good theoretical guarantees on how close the result is to the actual optimum. In this problem, however, those guarantees are damaged by the presence of constraints.

5.1.2 Constraint Programming approach

This approach casts the problem into a Constraint Programming formulation, which is a paradigm for solving combinatorial problems involving decision variables. Given the combinatorial nature of the problem and the temporal constraints, SaT lends itself well to a CP formulation. Just like the Distributed Algorithm, this approach requires to discretize the time windows, thus compromising the accuracy of the time constraints. However, this is necessary because a finer discretization would need a very large number of variables, damaging the solver's capacity to find a solution. The solver itself that was employed is an external commercial software, developed by IBM [5].

5.1.3 Planning approach

In this approach, the plans are built using the planner POPF-TIF [6] (Piacentini, Alimisis, Fox, & Long, 2015), built on top of the partial order temporal planner POPF2 [7] (Coles, Coles, Fox, & Long, 2010) and coupled with an external solver. The problem is expressed under the formal framework of PDDL (Planning Domain Definition Language) and is fully explained in [8] (Bernardini, Fox, Long, & Piacentini, 2016).

5.2 Comparisons

In order to evaluate the performance of our algorithm and to compare the results with those of the other solvers, it is necessary to study how they solve several different instances of a Search-and-Tracking problem. There are two main ways to do so. The first is to change the geographical features of the problem, such as the starting point, the destinations of the target and even the whole map. The second is to simply control the seed used. This mostly affects the true destination of the target and also the point where the UAV loses the target and starts the search, resulting each time a different set of search patterns. Changing the seed, of course, also affects the Monte Carlo simulation and the sample path of the Markov chain. Most comparisons will be shown by changing the seed and keeping the same geographical map, a portion of central Scotland already shown in section 3.1, the same starting point and the same destinations. A comparison involving another map, northern England, will also be shown.

Before we analyse the results over several problem instances, we will show in detail one of them. In this instance, the target starts outside the city of Stirling has for destination a location near Glasgow.

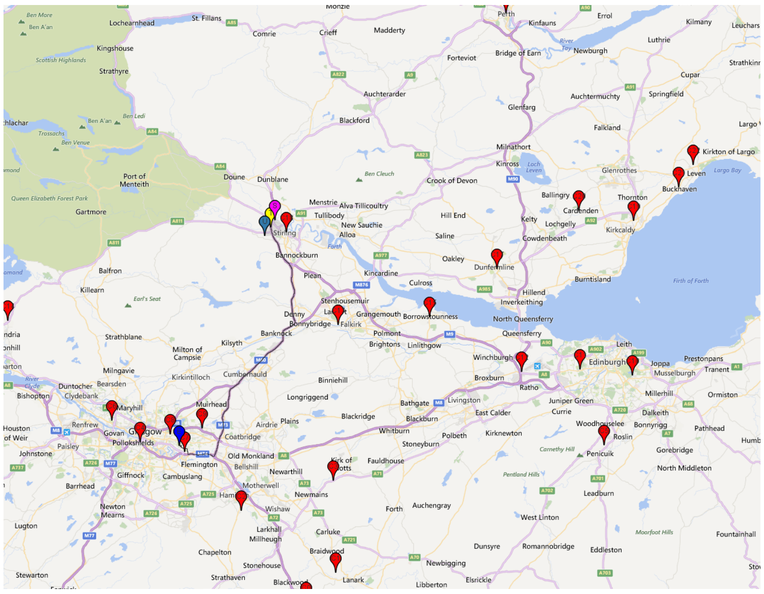

Figure 5.1: The target path is a slightly thicker line connecting Stirling (center) to a location outside Glasgow (lower-left).

The target is lost at time 291 after the beginning of the tracking. In order to have comparable results, both the times windows and the expected detection time will be shown as time after this moment, when the search begins. The candidate search patterns $\mathcal{C}$ are computed by the

Monte Carlo simulation. By default they are 30 (resulting from 15 time checks, each providing two candidate locations) and their time windows ()are as follows:

sp1	90.0-91.0	sp2	90.0-95.0	sp24	845.0-950.0
sp5	259.0-290.0	sp32	1036.0-1160.0	sp10	463.0-485.0
sp16	640.0-725.0	sp38	1254.0-1355.0	sp17	695.0-755.0
sp33	1118.0-1235.0	sp11	477.0-560.0	sp53	1718.0-1910.0
sp45	1486.0-1655.0	sp25	872.0-980.0	sp54	1704.0-1880.0
sp46	1486.0-1655.0	sp6	272.0-290.0	sp39	1295.0-1475.0
sp68	2100.0-2300.0	sp61	1922.0-2120.0	sp69	2127.0-2315.0
sp62	1909.0-2105.0	sp78	2318.0-2480.0	sp88	2509.0-2780.0
sp103	2945.0-3170.0	sp79	2277.0-2540.0	sp89	2509.0-2780.0
sp104	2904.0-3200.0	sp96	2713.0-3005.0	sp97	2740.0-3035.0

Their locations are also shown in figure 5.2, where we see that the UAV is searching one of the first ones while the target is elsewhere.

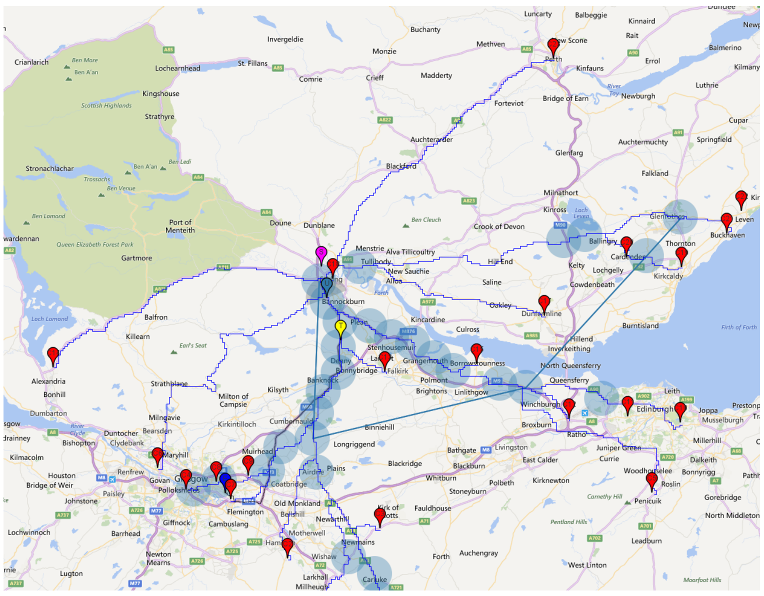

Figure 5.2: Blue circles are the 30 candidate search patterns for this problem instance.

In table 5.2, we compare the solutions that the four solvers (Greedy Algorithm, the Constraint

Programming formulation, the Planning approach and Distributed Algorithm) find when they optimize with respect to $G(\Sigma) = P(\Sigma)$, i.e. when $k = 0$.

	Greedy	CP	Planning	Distributed
Plan	sp1, sp38, sp62, sp97, sp104	sp1, sp11, sp38, sp46, sp97, sp104	sp1, sp11, sp38, sp46, sp97, sp104	sp1, sp11, sp32, sp68, sp97, sp104
$P(\Sigma)$	0.92322029387704	0.93395287517952	0.93395287517952	0.93096355967337
$T(\Sigma)$	10287.5	10369.5	10369.5	10792.5

As we can see, CP and Planning find the same optimum while the result of Distributed is slightly worse. Overall, their solutions are all quite close to each other - this is a feature of most seeds. In fact, their plans share quite a few search patterns (sp1, sp97 and sp104, for example, are in all four solutions), that are most likely responsible for most of the probability, while the most struggling part if finding which others provide the best additional increments. Other data on the Distributed algorithm are the following: the Markov chain does approximately two millions iterations (1871968) and roughly 0.18% of them are feasibles plans. There are many repetitions, however, and the number of distinct plans evaluated is actually 1484. As we will we see, this is approximately half of the total estimated number of feasible plans. This behaviour is consistent throughout all seeds, and it implies that the algorithm finds the true optimum approximately 50% of the times.

Now, we will show the optimum of the objective function

$$G(\Sigma) = P(\Sigma) - k \cdot T(\Sigma)$$

that the solvers found averaged over 40 seeds and for several values of k. Each solver was given 60 seconds of time to find their solution. In fact, all of them (except the Greedy Algorithm) guarantee to eventually find the optimum if enough time is given, but for a limited time their performances are not the same. The Greedy Algorithm is different because it directly builds a solution with a fast routine that only takes a few seconds. We also display results for three different values of the tradeoff parameter k. Following the approach of previous work on this SaT simulator, for each problem instance we define (recalling that $\bar{t}_\sigma$ is the midpoint of the time window of σ and that it is assumed to be the time at which the detection in σ would happen) the maximum time of detection

$$t^* = \max_{\sigma \in \widetilde{\mathcal{C}}} \{t_\sigma\}.$$

For example, in the problem instance previously analysed we have $t^* = 3057.5$, midpoint of sp103. As mentioned in section 3.3, we have that $A = k^{-1}$ must be larger than t^*, meaning that k must

	$k = 0$	$k = 0.25(t^*)^{-1}$	$k = 0.5(t^*)^{-1}$	$k = 0.75(t^*)^{-1}$	$k = (t^*)^{-1}$
Greedy	0.9179083	0.9065239	0.8985676	0.8856578	0.8703581
CP	0.9323648	0.9114946	0.8986684	0.8853056	0.8702885
Planning	0.9314915	0.9155787	0.9005554	0.8858280	0.8705217
Distributed	0.9294582	0.9125766	0.8988027	0.8831834495	0.8705886

be smaller than $(t^*)^{-1}$. Therefore, we display results by fractions of $(t^*)^{-1}$. From this table, we clearly see that the Greedy Algorithm performs quite well, arriving close to the optimum. As mentioned in 5.1.1, this is due to the properties of submodularity of the objective function. They are all quite close to each other and the Distributed Algorithm's performance is satisfactory, given its simplicity with respect to the Constraint Programming approach, which needs a model to be defined and an external, commercial solver, and with respect to the Planning approach. However, it also falls short of their benchmark, failing to outperform them consistently.

Now we show the performance of the Distributed algorithm over time, studying both the average number of distinct plans found and the average optimum at 30 seconds, 1 minute, 2 minutes, 3 minutes, 4 minutes and 5 minutes. The results are an average over 20 seeds, and the optimization was conducted with respect to $G(\Sigma) = P(\Sigma)$ $(k = 0)$. We see that eventually the optimum reaches a plateau, although the time limit of one minute is too restrictive and leaves room for improvement.

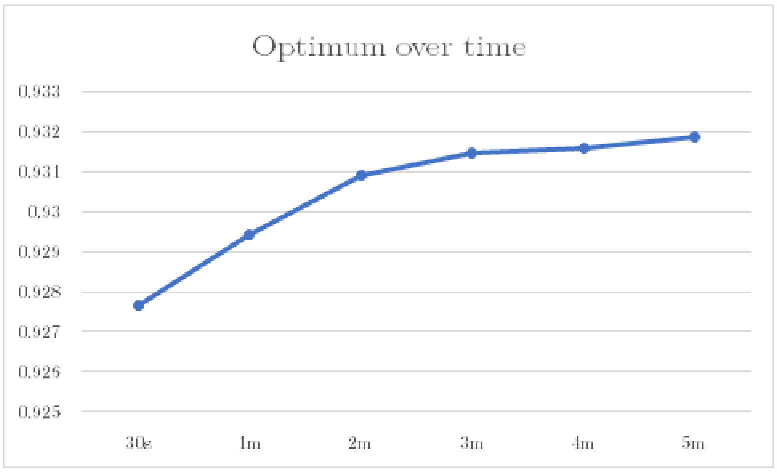

Figure 5.3: Average optimum over time for the Distributed algorithm.

Finally, we show results over a different map, this time a region of northern England whose geography of this region is more urban than Scotland's. Results are averaged over 40 seeds, and again they are the result of optimization with respect to $G(\Sigma) = P(\Sigma)$ ($k = 0$). The results

	Scotland	England
Greedy Algorithm	0.9191073412	0.8861336183
Constraint Programming	0.9323615072	0.8943964071
Planning approach	0.9323627057	0.8942147043
Distributed Algorithm	0.9297457115	0.8894374257

are similar: the numerical results are close to each other, Greedy Algorithm underperforms the others, and Distributed Algorithm obtains slightly worse results of the two competitors.

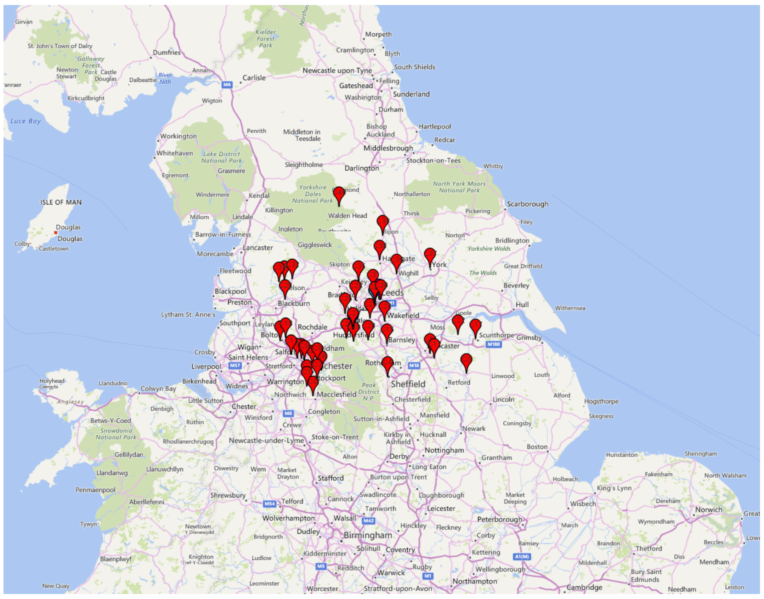

Figure 5.4: The simulation over northern England.

Chapter 6

Conclusions and future work

This work has shown that distributed algorithms are effective in tackling Search-and-Tracking problems and, more generally, can be used to solve planning problems with combinatorial features, where it is necessary to develop a plan choosing between conflicting tasks. When compared to the planning approach and the Constraint Programming paradigm, the distributed algorithm appears to be a more elegant and simple solution. It is also a flexible solution in many regards. Since it is a solution proposal algorithm, agnostic with respect to the objective function, it can be used to evaluate several performance functions at the same time. However, it has not been possible yet to consistently outperform existing approaches.

In order to do so, there exist a different approach that is worth of being further investigated and could be the topic of future work. Instead of discretizing the time window and creating κ nodes for each search pattern, it could be more effective to have a single node for each search pattern σ and to consider the time instant t as an action. In this framework, nodes must choose whether they should activate and, if they choose so, they also must pick at what time they start out of $[t_\sigma^-, t_\sigma^+]$: their action space would be

$$\mathcal{A}_\sigma = \{0\} \cup [t_\sigma^-, t_\sigma^+]$$

where 0 symbolizes that the node σ is not selected. In this case, the nodes are $\mathcal{V} = \widetilde{\mathcal{C}}$. We say that two nodes σ and σ' of actions $t_\sigma \in \mathcal{A}_\sigma$, $t_{\sigma'} \in \mathcal{A}_{\sigma'}$ are *in conflict* if they have chosen actions such that it is impossible to execute both, i.e. if their actions t and t' are not 0 and either

$$t_\sigma + d_\sigma + D^o(\sigma, \sigma') > t_{\sigma'}$$

or

$$t_{\sigma'} + d_{\sigma'} + D^o(\sigma', \sigma) > t_\sigma$$

holds. We define the graph by stating that there are edges between nodes that are in conflict for some choice of action, which amounts to:

$$\mathcal{E} = \{(\sigma, \sigma') \,:\, (t^+_\sigma + d_\sigma + D^o(\sigma, \sigma') > t^-_{\sigma'}) \vee (t^+_{\sigma'} + d_{\sigma'} + D^o(\sigma', \sigma) > t^-_\sigma)\}.$$

We wish to have no nodes in conflict and that for any inactive node he cannot activate without being in conflict with some neighbour. With a similar construction of the previous algorithm, for $y \in \mathcal{X}$ configuration let

$$\Psi^{(\sigma)}(y) = \begin{cases} 0 & \text{if } y_\sigma \neq 0 \quad \text{and} \quad (\sigma, y_\sigma), (\sigma', y_{\sigma'}) \quad \text{are not in conflict}, \quad \forall \sigma' \in N_\sigma \\ 0 & \text{if } y_\sigma = 0 \quad \text{and} \quad \forall t \in [t^-_\sigma, t^+_\sigma] \quad \exists \sigma' \in N_\sigma \,:\, (\sigma, t), (\sigma', y_{\sigma'}) \quad \text{are in conflict} \\ -1 & \text{otherwise} \end{cases}$$

be the negative indicator of violation either one of those constraints. The potential function is

$$\Psi : \{0,1\}^{\mathcal{V}} \longrightarrow \mathbb{R}$$
$$\Psi(x) = \sum_{\sigma \in \mathcal{V}} \Psi^{(\sigma)}(x).$$

Again, $\Psi(x)$ counts with the negative sign how many nodes are in violation of a constraint. Unlike before, now $x \in \mathcal{X}$ corresponds to a feasible plan if and only if $\Psi(x) = 0$. The utilities are

$$u_\sigma(x) = \sum_{\sigma' \in N_\sigma \cup \{\sigma\}} \Psi^{(\sigma')}(x).$$

and, like before, count with a negative sign how many nodes in $N_\sigma \cup \{\sigma\}$ are in violation. Therefore, the problem is not anymore that of sampling maximal independent sets: now nodes directly choose what their time assignments should be. One of the issues of this approach is that now the Noisy Best response is a discrete stochastic process, with the Markov property, but on a continuous state space. Although this is an issue for the existence of a stationary distribution, it is actually not a problem for the choice of a new action. Consider node σ and suppose he needs to pick a new action t from $\mathcal{A}_\sigma = \{0\} \cup [t^-_\sigma, t^+_\sigma]$, sampling from the (now continuous) distribution

$$\frac{1}{Z_\eta} \left(e^{\eta u_\sigma(t, x_{-\sigma})} \right)_{t \in \mathcal{A}_\sigma}.$$

However, the fact that other nodes' actions $x_{-\sigma}$ are fixed implies that $u_\sigma(\cdot, x_{-\sigma})$ is piecewise constant over $\mathcal{A}_\sigma$, which makes it very easy to sample t.

Another relevant issue that could be addressed is the generalization of the distributed approach to the multidrone case. One again, the most simple way this could be done is by changing the actions that each node could choose from and defining Φ accordingly. For example, for two drones we could have the action set $\mathcal{A} = \{0, 1, 2, 12\}$, meaning respectively that the seach pattern is assigned to no drone, to drone 1, to drone 2, or to both drones.

The main issue of all of these approaches, of course, is again the size of the state space, that increases exponentially. As mentioned, the issue is partially limited by the capacity of the Markov chain of being attracted by higher values of Φ, thus travelling only throughout the most interesting region. Nevertheless, experiments show that the size of the state space still must be contained and could severely affect performance. The challenge of distributed algorithms, therefore, is to be able to effectively describe the problem with a limited number of nodes, a limited number of actions, or with carefully designed utility functions.

Bibliography

[1] Piacentini, C.; Bernardini, S.; and Beck, J.C. (2019), Autonomous Target Search with Multiple Coordinated UAVs. In *Journal of Artificial Intelligence Research (JAIR)*, vol. 65.

[2] Bernardini, S.; Fox, M.; Long, D., & Bookless, J. (2013). Autonomous search and tracking via temporal planning. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS).*

[3] Como G., Fagnani, Giacomo Como and Fabio Fagnani, Lectures on Network Dynamics.

[4] Piacentini, C.; Fagnani, F.; Bernardini, S. (2020), Through the lens of sequence sbumodularity. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020).*

[5] IBM, CP optimizer CPLEX.
https://www.ibm.com/se-en/analytics/cplex-cp-optimizer

[6] Piacentini, C.; Alimisis, V.; Fox, M.; and Long, D. 2015. An extension of metric temporal planning with application to AC voltage control. Artificial Intelligence 229:210–245.

[7] Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling* (ICAPS-10).

[8] Bernardini, S.; Fox, M.; Long, D.; and Piacentini, C. 2016, Leveraging Probabilistic Reasoning in Deterministic Planning for Large-Scale Autonomous Search-and-Tracking.