Optimal Strategies of Autonomous Reconnaissance Missions

Technical report of my internship at the Netherlands Defence Academy

Source: Unsplash

Author:
Lander Verlinde

Supervisor:
Dr. Relinde Jurrius

Coordinator:
Dr. David de Laat

December 1st, 2023

Contents

Introduction

The role of Unmanned Aerial Vehicles (UAVs), more commonly known as drones, in society continues to become more significant every day. The civil market alone is estimated to be worth 7.2 billion USD in 2022 and this value is expected to grow to 19.2 billion USD in 2031 [1]. Applications range from agriculture over disaster response to package deliveries. But also its military use has become more relevant. Already in the Vietnam War, the US army deployed drones as a weapon [4]. When surveillance technology improved, it became clear that drones could also be used for survey missions in enemy terrain [3]. Moreover, these automated missions need not be performed by aerial systems, depending on the terrain and the characteristic of the mission, an automated ground vehicle (UGV) or an unmanned under water vehicle (UUV) may be more adept. The ongoing wars in Ukraine and Gaza have shown the importance of hybrid warfare [7]. In such warfare, the distinction between different modes of warfare (conventional warfare, cyber warfare, political warfare) tend to blurred. This makes the adversary in such war fluid and harder to predict [5]. Intelligence in physical and non-physical infrastructure is key in gaining advantage. Hence the scale in which unmanned vehicles are used for both offensive as well as reconnaissance missions is at an all time high [8].
To expand the number of operational systems while managing costs, it is desirable to deploy systems that can operate fully independently. For a survey mission, this requires a planning of the complete mission before the drone leaves for enemy territory. The setting of such a mission can be stated as follows: starting from a secure base, multiple surveillance locations need to be safely reached and the acquired information has to be brought back to the base. There are three possible ways of bringing back information. At every surveillance location, there is the possibility of transmitting information back to the base camp. Hence the first option is that the drone travels to a surveillance location, gathers the info and immediately sends it back to the base. Secondly, the drone could also store the information and go to the next surveillance location. After having obtained the information there, the totality of info could then be transmitted together at that location. Thirdly, the drone could also store information and return to the base. In this case the information is physically obtained back from the drone. However, each action in the mission carries the risk of detection – the drone could be spotted during flight, or transmissions might be intercepted. Both ways of detection give away the position of the drone, ending the mission abruptly.
This report investigates how to find the optimal strategy of these reconnaissance missions. Such an optimal strategy consists of two elements: both the route and the send strategy have to be optimal to maximise the amount of retrieved information. Hence the specific questions for which an answer is sought in this research are:

- In which order should the different locations be travelled to?
- Where is it beneficial to make a transmission and where is it better to hold on to the gathered information?

The report is structured in four chapters. First of all, the problem is described mathematically. A model based on weighted graphs is proposed and two ways to compute the expected value of retrieved information are discussed. Furthermore, it is investigated whether the problem can be written as an Integer Program and a motivation is given for the choice of a heuristic algorithm. Secondly, the case where one drone is deployed during the mission is examined in great detail. A genetic program that yields the optimal strategy is presented. This genetic program is tested on several mission scenarios and different algorithms are compared in terms of success rate and complexity. Thirdly, the scope is extended to missions with multiple drones. The best genetic program from the single drone scenario is adapted and improved such that it can also solve the multiple drone scenario. This improved algorithm is tested in the same way as the single drone scenario and a comparison is made. Lastly, the conclusion is presented. Results are put within their mathematical and societal perspective and open questions are addressed.

Chapter 1

Mathematical Exploration of the Problem

1.1 How to Model a Reconnaissance Mission?

Two mathematical formulations of a recon mission are presented in this chapter: one focusing on the transmissions and one considering the journey of the information from each vertex. The first one is more intuitive and is suitable for the first part of the research. However, the intuitive formulation leads to a less useful objective function. Therefore a second formulation is stated which is slightly less straightforward to construct, but easily generalised to scenarios with multiple drones.

1.1.1 Focus on the Transmission Vertices

A mission on k surveillance locations can be naturally formulated as a problem on a weighted graph $G = (V, E)$, with $V = \{0, \ldots, k\}$. The weights consist of both edge weights and vertex weights.
Every edge ij has weight $q_{ij} \in [0, 1]$. These weights correspond to the crossing probabilities: the survival chance of crossing from i to j. Moreover, every vertex i has two weights. The first one $p_i \in [0, 1]$ is the transmission probability: the probability of making a successful transmission at vertex i. The second weight z_i is the amount of information that can be gathered at that vertex. Having these weights and probabilities, finding the optimal strategy of the mission, boils down to finding a walk with corresponding send strategy that maximises the expected amount of retrieved information. In such a strategy, the following rules apply:

- Vertex 0 corresponds to a safe base camp: successful transmission has probability one, but there is no information to retrieve here. This means that p_0 is equal to 1 and z_0 is 0.
- The walk can have repeated vertices. However, once the information has been retrieved at a vertex i, z_i is set to 0.
- If information is retrieved but not transmitted, it is carried to the next vertex in the walk and can be transmitted there or further down the walk. A transmission always sends all information that has been retrieved but not yet transmitted. At the last vertex of the walk – the base camp – a transmission is always made, but this transmission can be empty.

- Once a crossing or a transmission fails, the mission is over and no more information can be retrieved nor transmitted.

Following the above rules, the expression for the expected amount of received information for a walk with corresponding send strategy is formulated. By looking at the different vertices where information is transmitted, it is computed how much information every transmission is expected to contain. Let $R = [v_1, v_2, \ldots, v_m]$ with $v_1 = v_m = 0$ be the sequence of vertices that make up the walk. Say v_i has transmission probability p_{v_i} and the probability of crossing from v_i to v_{i+1} is $q_{v_i v_{i+1}}$. Moreover, let $S = [v_{s_1}, v_{s_2}, \ldots, v_{s_n}]$ be the subsequence of R of n vertices where information is transmitted. Then we compute the probability to survive the entire route with corresponding send strategy with

$$\mathbb{P}(\text{survival}) = \prod_{j=1}^{m-1} q_{v_j v_{j+1}} \prod_{i=1}^{n} p_{s_i}.$$

And if X is the random variable that denotes the amount of retrieved information, then the total expected information is given by

$$\mathbb{E}[X] = \sum_{i=1}^{n} \prod_{j=0}^{s_i - 1} q_{v_j v_{j+1}} \prod_{\substack{u: v_{s_u} \in S \\ u \leq i}} p_{s_k} \sum_{s_{i-1} < h \leq s_i} 1.$$

Example 1.1.1. To get a better insight in the above formula, it is useful to work out an example. Consider a mission which is given by the graph in Figure 1.1. This graph consists of a base camp at vertex 0 and three surveillance locations at vertices 1, 2 and 3. The crossing and transmission probabilities are shown on the graph. At every surveillance location there is one unit of information to be retrieved. Now let's assume that the following strategy is chosen:

Route: 01230

Send: 01001.

This send strategy means that a transmission is made at vertex 1 and back in the base camp upon return.

Let X be the random variable for the amount of well-received information for this route and send strategy. X_1 is the random variable that defines the amount of received information at the first transmission and X_2 at the second one. By linearity of expectation: $\mathbb{E}[X] = \mathbb{E}[X_1] + \mathbb{E}[X_2]$. To compute $\mathbb{E}[X_1]$, first the probability of safely reaching vertex 1 needs to be computed. That is 0.6. The transmission probability is 0.9 and there is one unit of information being transmitted. So:

$$\mathbb{E}[X_1] = 0.6 \cdot 0.9 \cdot 1 = 0.54.$$

We make a similar computation for the second transmission. The probability of reaching the base camp is $0.6 \cdot 0.9 \cdot 0.3 \cdot 0.9 \cdot 0.6$. The transmission probability is 1 and there are two units of information transmitted: the information from vertices 2 and 3. The expected amount of transmitted information with the second transmission then becomes:

$$\mathbb{E}[X_2] = 0.6 \cdot 0.9 \cdot 0.3 \cdot 0.9 \cdot 0.6 \cdot 1 \cdot 2 = 0.17496.$$

Combining this yields the total expected value:

$$\mathbb{E}[X] = 0.54 + 0.71496 = 0.71496.$$

So for this graph, if the route 01230 is chosen with transmissions at vertices 1 and 0, the mission is expected to retrieve 0.71496 units of information out of the possible 3 units. The question can be raised whether it is worth setting up a reconnaissance mission if only a small fraction of information is expected to be retrieved. In the following sections we will investigate whether this value can be increased using an algorithm to find a better strategy.

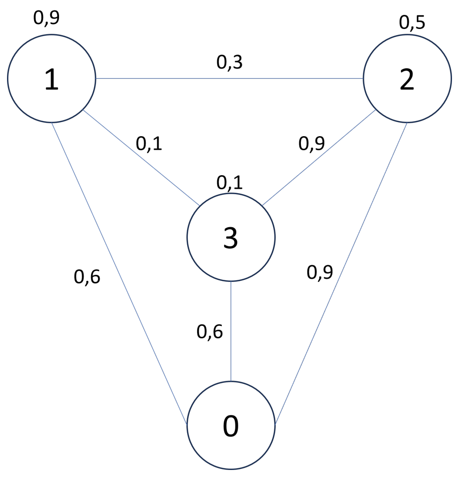

Figure 1.1: Example of a graph with send and crossing probabilities.

1.1.2 Focus on All Vertices

Intuitively, it makes sense to focus on the transmission vertices, as we are interested in the expected amount of transmitted information. However, this formula is hard to generalise to the case of multiple drones, since information sent by multiple drones should only be counted once towards the total retrieved information. Therefore in this case it makes sense to construct an equivalent formulation that focuses on how much information from each vertex is expected to reach the base camp.

The expected amount of information that is safely transmitted can also be formulated as the sum of the expected fraction of each unit of information that is safely transmitted. If vertex i contains z_i units of information, then we compute $\mathbb{E}[X_i]$: the expected amount of information from vertex i that reaches the base.

In the case of one drone, this is equal to the product of z_i with the probability that the drone safely reaches the first transmission vertex after i and the probability that the transmission at this vertex is successful.

For the case with two drones, some more notation is required. Let D_{1i} be the event that drone number 1 succeeds in transmitting the information from vertex i. Similarly, let D_{2i} be the event that drone number 2 succeeds in transmitting the information from vertex i. Then we are interested in the probability that D_{1i} or D_{2i} happens. Note that this is an inclusive disjunction.

Using independence of probabilities we obtain:

$$\begin{aligned}\mathbb{P}\left(D_{1i} \vee D_{2i}\right) &= \mathbb{P}\left(D_{1i}\right) + \mathbb{P}\left(D_{2i}\right) - \mathbb{P}\left(D_{1i} \wedge D_{2i}\right) \\ &= \mathbb{P}\left(D_{1i}\right) + \mathbb{P}\left(D_{2i}\right) - \mathbb{P}\left(D_{1i}\right) \cdot \mathbb{P}\left(D_{2i}\right).\end{aligned}$$

Hence this leads to a formula for the expected value $\mathbb{E}\left[X_i\right]$:

$$\mathbb{E}\left[X_i\right] = z_i \cdot \left(\mathbb{P}\left(D_{1i}\right) + \mathbb{P}\left(D_{2i}\right) - \mathbb{P}\left(D_{1i}\right) \cdot \mathbb{P}\left(D_{2i}\right)\right).$$

Linearity of expectation provides an alternative formula for the total expected value:

$$\mathbb{E}[X] = \sum_{i=1}^{|V|} \mathbb{E}\left[X_i\right].$$

This perspective on the expected value might be a little less computationally advantageous: for every vertex we need to recompute the probability that the drone reaches the next transmission vertex and is able to make a transmission. But it gives rise to a formula for an arbitrary amount of drones. The formula for the union probability can easily be adapted to ℓ drones, as

$$\mathbb{E}[X_i] = z_i \cdot \mathbb{P}\left(D_{1i} \vee \cdots \vee D_{\ell i}\right),$$

which we again compute using the independence of probabilities.

Example 1.1.2. Returning to Figure 1.1, the expected value of the strategy

Route: 01230
Send: 01001

can be computed with the new formula as well. There are three surveillance locations, so the expected value of retrieved information is decomposed in a sum over these three vertices:

$$\mathbb{E}[X] = \mathbb{E}[X_1] + \mathbb{E}[X_2] + \mathbb{E}[X_3].$$

The three terms are computed separately. The one unit of information from vertex 1 is immediately transmitted at this vertex. The probability of reaching this vertex is 0.6, the transmission probability is 0.9, plugging this in yields

$$\mathbb{E}[X_1] = 1 \cdot 0.6 \cdot 0.9 = 0.54.$$

The information from vertex 2 is sent at vertex 0. The probability of this information reaching its transmission vertex is $0.6 \cdot 0.9 \cdot 0.3 \cdot 0.9 \cdot 0.6$ and the transmission probability is 1. Additionally, The information from vertex 3 is also transmitted at vertex 0, so we compute both values with the same formula:

$$\mathbb{E}[X_2] = \mathbb{E}[X_3] = 1 \cdot 0.6 \cdot 0.9 \cdot 0.3 \cdot 0.9 \cdot 0.6 \cdot 1 = 0.08748.$$

The sum of the above values provides the total expected value:

$$\mathbb{E}[X] = 0.54 + 0.08748 + 0.08748 = 0.71496.$$

Indeed, this is the same value as obtained in Example 1.1.1.

1.2 How to Construct the Optimal Strategy?

1.2.1 The Optimal Send Strategy

Even though the task at hand is to find an optimal route and corresponding send strategy, it is insightful to first look how to choose the best send strategy for a given (optimal) route. *Suppose we were able to find the best route, can we then easily find the best send strategy?*
Finding this best send strategy was the subject of the bachelor thesis of Abe Boswinkel. [2] This section is mostly based on that thesis. Suppose a route is presented for which the send strategy is wanted. At every vertex in the route, the choice needs to be made whether it is better to send the information now, or hold on to it and send it at a later stage. Both options involve a risk and to decide if the information should be transmitted at a certain vertex, all possible next transmission vertices need to investigated. However, for one vertex there is no choice: at the last vertex of the route, all remaining information should be transmitted. This means that a dynamic program can be used that starts at the last vertex and work its way back.
Such a program requires multiple parameters. Again, let p_i be the probability that the transmission at vertex i is successful and let q_{ij} be the probability of crossing safely from vertex i to vertex j. Moreover, it is assumed that there are z_i units of information available at vertex i. Let $C_i(h)$ be the potential gain in information at vertex i, holding h units of information. Suppose the considered route is $[v_1, \dots, v_m]$. Then the dynamic program then becomes

$$\begin{aligned} C_{v_i}(h) &= \max\left(p_{v_i} \cdot \left(h + z_{v_i} + q_{v_i v_{i+1}} \cdot C_{v_{i+1}}(0)\right), q_{v_i v_{i+1}} \cdot C_{v_{i+1}}(h + z_{v_i})\right) \\ C_{v_m}(h) &= p_{v_m} \cdot (h + z_{v_m}). \end{aligned}$$

The first line consists of a maximum of two values, $p_{v_i}\left(h + z_{v_i} + q_{v_i v_{i+1}} \cdot C_{v_{i+1}}(0)\right)$ corresponds to the probability of making a successful transmission at vertex v_i, sending $h + z_{v_i}$ units of information and then crossing to vertex v_{i+1} where the drone arrives with 0 units of information. Since it is impossible to cross to the next vertex when the transmission is intercepted, $q_{v_i v_{i+1}} \cdot C_{v_{i+1}}(0)$ is multiplied with p_{v_i}. The other value is the crossing probability to the next vertex, where the drone arrives with $h + z_{v_i}$ units of information.
The second line is the potential gain in information at the last vertex. Since there is no other vertex to travel to, the only option is to send all information that was still held on to. This leads to the probability p_{v_m} multiplied with $h + z_{v_m}$. Note that since v_m is the base camp, we know that $p_{v_m} = 1$ and $z_{v_m} = 0$.
The strength of this dynamic program is that it can quickly find the optimal send strategy for any given route. This shows that if the optimal route can be found, the problem is basically solved. This seems to simplify the problem to finding the best route.

1.2.2 The Optimal Route

The ideal case to find the optimal route would be the existence of some heuristic or local decision rule to decide which vertex to travel next to. A first possible approach would be to try to find a so called Hamilton cycle. This is a route that starts at the base, passes by every vertex exactly once and returns to the base. Such a cycle requires the least number of crossings, so this might be a good idea in terms of the survival probability. It also has practical applications in terms of distance. As a way of choosing the best Hamilton cycle, one could look at the cycle with the largest product of edge weights i.e., the cycle with the highest survival probability not taking into account the transmissions.

Example 1.2.1. Let us return to Example 1.1.1. For this graph, the best such cycle is 01230 or 03210. Using the dynamic program, one can find that the first cycle has an expected value of retrieved information equal to 0.715 and the second one has expected value 0.613. The former is the best possible Hamilton cycle in terms of expected value. These values are not that high, considering there are three units of information to be retrieved from this graph. Furthermore, one can note that vertex 3 in this graph only has one edge with high crossing probability. Hence it could make sense to go back and forth over that edge, for example with route 012320. And indeed, this yields to an expected value of 0.776. So a Hamilton cycle is not necessarily the best route. For this graph it is possible to work out that the route with the highest possible expected value is actually 0232010. This means that it is optimal to return to the base, but not to fly back and forth the base and every single surveillance location. This is partly because the graph is not metric, so the triangle inequality doesn't hold. Travelling from vertex 3 to the base camp at vertex 0 is more secure via vertex 2. Then the survival probability is 0.81 instead of the 0.6 probability of directly going from 3 to 0. The send strategy that yields the optimal value is 0000111 and the expected value ends up to be 1.666. All considered strategies and their respective expected values are shown in Table 1.1. The table shows that the optimal expected value is more than twice the value of the best Hamilton cycle. But if a Hamilton cycle is not necessarily the best route, what does characterise the optimal one?

Option 1	Route: Send:	01230 01001	Expected Value:	0.715
Option 2	Route: Send:	03210 00111	Expected Value:	0.613
Option 3	Route: Send:	012320 010001	Expected Value:	0.776
Option 4	Route: Send:	0232010 0000111	Expected Value:	1.666

Table 1.1: Table with possible routes, send strategies and their corresponding expected values. 0 in a send strategy means not to send, while 1 means that a transmission is made. Clearly, it can be worth it to not choose a cycle but go back and forth through the graph.

1.2.3 The Necessity of Large Enough Probabilities

As the problem is formulated now, the edge and vertex weights can be any probability, i.e. any value between 0 and 1. But what happens if every such value is small? From a mathematical point of view this is allowed: the optimal route can still be computed. But from a practical point of view something has to be noted: as the risk of being detected is very high everywhere, is the mission still worth it? To formulate an answer to this question, we study the following example.

Example 1.2.2. Consider Figure 1.2. In this graph, the crossing or transmission probability is never greater than 0.3. So that means, there is only a 30 percent change the mission even makes it to the first point. Let's work out what the best strategy is in this high risk environment.
Travelling to any additional vertex will always lead to a strictly positive extra term in the sum that makes up the expected value. Hence this expected value will be maximised by a route that

passes by all points. As the graph is still small, it is possible to find out that the optimal strategy is given by:

$$\text{Route: } 0102030$$
$$\text{Send: } 0010101.$$

This corresponds with going to a point and returning the information to the base in decreasing order of probability of getting there. This yields an expected value of 0.0985. Knowing that there are three units of information to be retrieved, this is not a great score. Computing the expected value of every transmission gives more insight in this mission. There are three transmissions made – all at the base camp – so let X_1, X_2 and X_3 be their corresponding random variables.

$$\mathbb{E}[X_1] = 0.3 \cdot 0.3 = 0.09$$
$$\mathbb{E}[X_2] = (0.3)^4 = 0.0081$$
$$\mathbb{E}[X_3] = (0.3)^4 \cdot (0.25)^2 = 0.00050625$$

This shows that the only transmission vertex that actually contributes to the expected value is the first one. The second vertex only adds a value that is ten times smaller than the first one. So even though the optimal value is attained after having passed all points, it seems more reasonable to only maximise the expected value of the first transmission and then not visit any other point. In that case there is still a chance that the drone returns home.

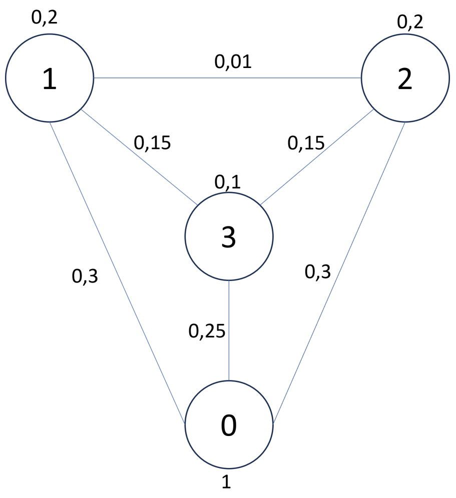

Figure 1.2: Example of a graph where the probability of crossing is low everywhere.

But doesn't this hold for any graph with probability weights? Travelling to a next vertex and transmitting the gathered information will always lead to at least two additional elements in the product that makes up expected value of the corresponding transmission. Since all these terms are between 0 and 1, the product can rapidly decrease. It seems that the first term has by far the most influence on the entire expected value.
The question arises whether maximising the first transmission of a route followed by some other way to order the remaining nodes imply an optimal or near-optimal solution. To investigate

this, one can look into the opposite of the previous example: a graph with high probabilities everywhere.

Example 1.2.3. An example of a high probability graph is depicted in Figure 1.3. Suppose purely the first transmission is maximised. Since the probability of crossing is high everywhere, the expected value of the first transmission is maximised if the drone stores all information and sends the three units of info at the third point or fourth. It can be found that the best such strategy is

Route: 03120
Send: 00011.

The expected value then becomes 2.134350. But if the send strategy is changed to 01111, so a transmission is made at every vertex, the total expected value increases to 2.33110, even though the expected value of the first transmission decreased. So for this route the strategy isn't optimal. Moreover, the strategy

Route: 01230
Send: 01111.

yields an expected value that is even higher with a value of 2.334820. So even for very small graphs the considered strategy is not the way to go. On larger graphs, the transmission would be further postponed, increasing the gap between the expected value given by this strategy and the optimal one. Therefore a better approach to find the optimal route needs to be researched.

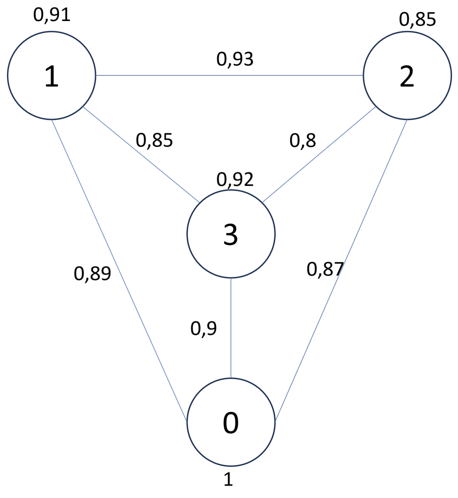

Figure 1.3: Example of a graph where the probability of crossing is high everywhere.

It can be concluded from this section that for the problem to be mathematically interesting and also practically worth investigating the probability on the edges and vertices need to be high enough. Otherwise the only term that really adds to the expected value is the first one and the probability of actually obtaining information is too low to start the mission in the first place. Also, since there is no clear way of predicting what the optimal route for a graph looks like, the route and send strategy have to be simultaneously optimised. Both have an impact on the expected value that is maximised and one cannot be looked at without the other.

1.2.4 Formulating the Problem as an ILP

Ideally, the problem can be stated as an Integer Linear Program (ILP). Since there exist efficient solvers for this type of program, it might provide a way to consistently find the optimal strategy. When using the formulation of the expected value in terms of the expected amount of retrieved information from each vertex as proposed in subsection 1.1.2, we need to know at which moment every vertex is visited and at which moment the gathered information is transmitted. This allows to compute the probability that the drone successfully transmits the information. This can be encoded with the following decision variables:

$$X_{it} = \begin{cases} 1 & \text{vertex } i \text{ is visited at time } t \\ 0 & \text{otherwise} \end{cases}$$
$$Y_{it} = \begin{cases} 1 & \text{a transmission is made at vertex } i \text{ at time } t \\ 0 & \text{otherwise.} \end{cases}$$

These two decision variables are enough to represent a strategy and given a strategy, we can always construct the corresponding nonzero X_{it} and Y_{it}.

Example 1.2.4. Suppose we are given the following strategy:

Route: 03120
Send: 00011.

Then $X_{01} = 1$, $X_{32} = 1$ etc. Similarly, the only Y_{it} not equal to 0 are $Y_{24} = 1$ and $Y_{05} = 1$.

We will also need two decision variables that are closely related to X_{it} and Y_{it}:

$$A_{ijt} = \begin{cases} 1 & \text{there is a crossing from vertex } i \text{ to vertex } j \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$
$$B_{ijst} = \begin{cases} 1 & \text{information gathered from vertex } i \text{ at time } s \\ & \text{is transmitted at vertex } j \text{ at time } t \\ 0 & \text{otherwise.} \end{cases}$$

Now the required constraints for such a program can be formulated. Immediately, the first issue arises. All vertices are part of the vertex set V. Similarly, a time set T is needed that consists of all possible values of t. But it is not known how long the optimal strategy actually is. However, from the examples we have seen so far and those we will see in the next chapters, it is a reasonable assumption that the length of the optimal strategy lies between $|V|$ and $2|V|$. This means the program can be run for all different options for T. This seems tedious, but let's assume this is possible, for the sake of investigating the required constraints.
First of all, at each moment in time, there is only vertex that can be visited. This can be imposed with

$$\sum_i X_{it} = 1 \quad \forall t \in T. \tag{1.1}$$

Moreover, the same vertex cannot be visited twice in a row. A crossing can only be made between two sequential vertices and there are $|T|-1$ crossings in each route. This leads to four additional constraints:

$$A_{iit} = 0 \quad \forall i \in V, \forall t \in T \tag{1.2}$$
$$A_{ijt} \leq X_{it} \quad \forall i,j \in V, \forall t \in T \tag{1.3}$$
$$A_{ijt} \leq X_{jt+1} \quad \forall i,j \in V, \forall t \in T \tag{1.4}$$
$$\sum_{i,j,t} A_{ijt} = |T| - 1. \tag{1.5}$$

A transmission at a vertex can only be made at the moment that vertex is visited. This relates Y_{it} and X_{it} with

$$Y_{it} \leq X_{it} \quad \forall i \in V, \forall t \in T. \tag{1.6}$$

Since the information from each vertex has to be transmitted exactly once, the following constraints also arise:

$$B_{ijst} \leq X_{is} \quad \forall i,j \in V, \forall s,t \in T \tag{1.7}$$
$$B_{ijst} \leq Y_{jt} \quad \forall i,j \in V, \forall s,t \in T \tag{1.8}$$
$$\sum_{i,j,s,t} B_{ijst} = |V|. \tag{1.9}$$

Lastly a variable is wanted that indicates whether the information of vertex i is has been gathered and is gone at time t. This binary variable allows for only considering the probabilities up until the transmission of the information to compute the expected transmitted portion of this information. The indicator variable needs to have the following form:

$$Z_{it} = \begin{cases} 1 & \text{information from vertex } i \text{ has been transmitted before time } t \\ 0 & \text{otherwise.} \end{cases}$$

Using the existing decision variables, this becomes:

$$Z_{it} = \begin{cases} 1 & \sum_{s\leq t} \sum_{s\leq q\leq t} \sum_{j\in V} B_{ijsq} \geq 1 \\ 0 & \text{otherwise.} \end{cases}$$

This indicator variable is encoded with two constraints [9]. First of all, a new variable is defined:

$$C_{it} = \sum_{s\leq t} \sum_{s\leq q\leq t} \sum_{j\in V} B_{ijsq} - 1. \tag{1.10}$$

For a large enough value of M, this inequality holds:

$$-2 \leq C_{it} \leq M.$$

So to impose the implication $C_{it} \geq 0 \Rightarrow Z_{it} = 1$, the following constraint can be used:

$$MZ_{it} - C_{it} > 0 \quad \forall i \in V, \forall t \in T. \tag{1.11}$$

At the same time, the implication $C_{it} < 0 \Rightarrow Z_{it} = 0$ has to hold. This can be obtained with:

$$2(1 - Z_{it}) + C_{it} \geq 0 \quad \forall i \in V, \forall t \in T. \tag{1.12}$$

Lastly, all decision variables need to be binary:

$$X_{it}, Y_{it}, A_{ijt}, B_{ijst}, Z_{it} \in \{0,1\} \quad \forall i,j \in V, \forall s,t \in T. \tag{1.13}$$

These 13 constraints enable deciding which route to take and where to send. But now it is desired to use them to maximise the expected value from subsection 1.1.2. This expected value was formulated as:

$$\mathbb{E}[X] = \sum_{i=1}^{|V|} \mathbb{E}\left[X_i\right],$$

where every X_i is the random variable that states how much information of vertex i is successfully transmitted. If q_{ij} is the crossing probability between vertices i and j, p_i is the transmission probability at vertex i and z_i the amount of information to be retrieved at i, $\mathbb{E}[X_i]$ is optimised by maximising the objective function

$$f_i(x) = \ln(z_i) + \sum_i \sum_j \sum_t \left[\left(\ln(q_{ij})A_{ijt} + \ln(p_i)Y_{it}\right)(1 - Z_{it})\right].$$

The logarithmic function is used to turn the product of all probabilities into a sum. However, the objective function is the sum over all $\mathbb{E}[X_i]$. Therefore it is unclear how to make the entire objective function linear, as taking the sum over all f_i clearly doesn't work. Finding a linear objective function seems to be a hard task where the best idea so far is to use a binary decision variable for each route that is 1 if that specific route is chosen and 0 otherwise. This is a silly way of setting up the program, as it would be equivalent to a search over all possible routes. The hope remains that (parts of) the above method could still be improved to obtain some integer program, but further research is required. In any case, such an IP would require a lot of decision variables and the choice was made to research an efficient and reliable heuristic instead.

Chapter 2

Single Drone Missions

2.1 The Genetic Algorithm

Instead of looking for an integer program, which would be able to consistently find the optimal strategy, this chapter investigates how to construct a heuristic algorithm when the mission deploys a single drone. Such a heuristic does not necessarily return the optimal strategy, but if it does so often enough and the algorithm is fast, it provides a good alternative. If the graph is small, it is still possible to go over all routes and find the optimal ones. Yet the number of routes increases extremely fast as the number of vertices increases. The only known property in advance of the optimal route is that it starts and ends at the base camp and passes by all other vertices, so there are a lot of possible routes to consider.

Example 2.1.1. Consider a complete graph on six vertices with probabilities as given in the following graph:

$$\begin{bmatrix} 1 & 0.97 & 0.81 & 0.97 & 0.95 & 0.96 \\ 0.97 & 0.93 & 0.92 & 0.87 & 0.89 & 0.87 \\ 0.81 & 0.92 & 0.96 & 0.81 & 0.98 & 0.93 \\ 0.97 & 0.87 & 0.81 & 0.85 & 0.93 & 0.93 \\ 0.95 & 0.89 & 0.98 & 0.93 & 0.91 & 0.89 \\ 0.96 & 0.87 & 0.93 & 0.93 & 0.89 & 0.90 \end{bmatrix}.$$

The diagonal entries correspond to the transmission probabilities, the off-diagonal entries are the crossing probabilities. For this larger mission scenario, it is extremely difficult to find the optimal route by trial and error. A more clever approach is needed.

As the length of the optimal strategy is not known in advance and the route and send strategy have to be optimised simultaneously, the approach of a genetic algorithm seems a great fit. Genetic programming is based on the idea of survival of the fittest and natural selection. Suppose a list of possible strategies is given. This list can be seen as a *generation* in the program. From this list, natural selection would like to pick the best routes to become the *parents* of new routes. Hence a new generation is created in which we hope that the *children* are even better strategies. This is repeated until hopefully at some point the optimal route is found.

2.1.1 Detailed Explanation of the Genetic Program

Natural selection and crossing strategies to find new ones may sound like a great idea, but how does this algorithm work practically? First of all, the algorithm needs to be initialised. In the first iteration, a list of random routes is generated. These routes are constructed by taking a number of random steps through the graph starting from the base. The length of this walk is chosen in terms of the size of the considered graph. For example, for K_6, the length chosen uniformly at random be between the number of vertices of the graph minus four and the number of vertices plus four. If the last vertex of this walk is not the base, the shortest path between those two vertices is added. One can imagine that when the graph becomes large, the length of these routes can become large as well. But eventually, the first generation is filled, as sketched in Figure 2.1.

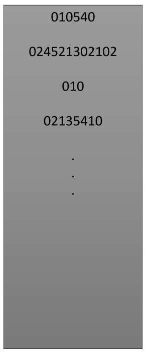

Figure 2.1: To initialise the genetic program, a lot of random routes are generated. Every number represents a vertex in K_6.

To know which routes are great and which we don't want to use for a next generation, we need to compute the expected value. Therefore we need a send strategy. Hence for every route, the dynamic program is run and the expected values are stored with their routes. The send strategies are not stored, as in every generation, we create new ones with the dynamic program. Based on these values, the routes are ranked from best to worst. Three categories are created: the best $\pm 10\%$, the worst $\pm 7\%$ and all other routes (see Figure 2.2).

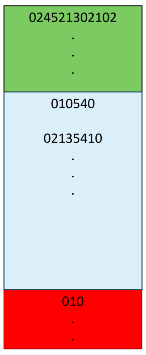

Figure 2.2: The generation is split in three categories: the best $\pm$ 10%, the worst $\pm$ 7% and all other routes.

At this point the algorithm has a list of ranked strategies and their expected values. Now a new generation has to be constructed. Thus, first the routes with the very best expected values are put directly in the next generation. This means that the best value of this new generation cannot be worse than the previous best value. Then, the worst routes from the previous generation are deleted. To construct a route in the next generation, two routes are picked uniformly at random from the remaining ones. These two routes will become the parents of new route, hoping that the good genes of the parents will be inherited by the child. To cross these two parents, a vertex that is present in both routes is randomly chosen and the parent routes are sliced in two parts at the first occurrence of this vertex. From the one route we use the first part, from the other route the second part.

Example 2.1.2. Suppose the two routes picked uniformly at random are 015412342050 and 030120250410. The next step is to pick at random one vertex in the graph. Let's say that vertex 2 was picked to make the crossover. The first occurrences of vertex 2 are 015412342050 and 030120250410. Hence the crossing of these two routes becomes 0154120250410.

It is possible that two routes are not compatible to construct a child route. If two attempts of finding a vertex present in both routes fail, two new routes are picked and these are crossed instead.
This strategy keeps being repeated: the routes in this generation are ranked from best to worst using the dynamic program and a new generation is constructed. After enough generations the hope is that eventually the globally optimal strategy will appear.

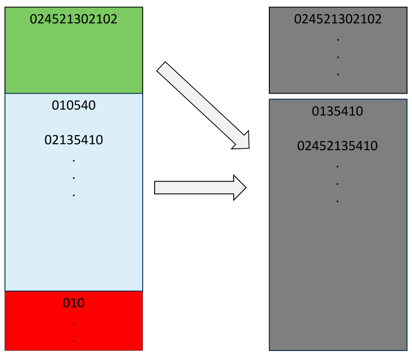

Figure 2.3: The best routes are immediately put in the next generation. Thereafter, the next generation is filled with crossings from the previous generation.

To test this algorithm, the graph from Example 2.1.1 was used. 60 generations of 500 routes were constructed. In each generation, the 30 best routes were chosen and the 20 worst routes were deleted. This algorithm was run ten times. The best strategy in these runs is given by

Route: 010304250
Send: 001010101

with expected value 4.115090. This strategy was found in five out of ten runs. Since the algorithm is fast enough, it is feasible to run it for example ten times, which results in almost always finding this solution. So for K_6, the genetic algorithm in its current form is sufficient. However, in subsection 2.2.2, a better genetic algorithm is required and proposed. This genetic algorithm was run a hundred times on the above example and the same strategy was found 98 times. This strengthens the belief that this is indeed the optimal strategy.

2.2 Results

2.2.1 Sparse Graphs

The first generation of the genetic algorithm is a list with randomly generated routes. When dealing with a complete graph, these are easily generated: construct a vector where every element is picked uniformly at random from all vertices of the graph and delete repeated vertices.
Just as easy, two routes in a complete graph can be crossed: pick a vertex in each route and slice the route in two at these points. Since the graph is complete, these slices can just be put together and the solution will always be feasible. However, things become more complicated when the graph is not complete, or equivalently, complete but with crossing probability zero for non-existing edges. If the genetic algorithm with the aforementioned naive initialisation and crossing techniques is used, the generations are almost exclusively filled with non-feasible routes. This means that the optimal solution is almost never found. To make sure that only feasible routes are considered in the initialisation, a random walk starting from the base is used. The length of this walk is also randomly chosen. Such a random walk does not necessarily end back in the base, which is a requirement for a feasible route. Hence Dijkstra's algorithm is used to take the shortest route back to the base from the last vertex of the random walk. The crossover method from Example 2.1.2 is constructed in such a way that only feasible routes are constructed in the following generations as well.

Example 2.2.1. In certain scenarios, a complete graph does not model the reality of a reconnaissance mission adequately. There might be an enemy base that cannot be flown right over between surveillance locations, or there might be a river flowing through the enemy terrain, with only a few bridges that can be crossed. Both in aerial and ground operations, the graph corresponding to the mission could be sparse. Such a graph is depicted in Figure 2.4. With the implementation of this approach, the genetic algorithm with 60 generations of 500 routes, again with the best 30 immediately to the next generation and the worst 20 deleted, the global minimum for this graph was found in 9 out of 10 times. This best route is given by

$$\text{Route: } 013245678790$$
$$\text{Send: } 001000010101$$

and has expected value 6.113745. The fact that this optimal route is found in so many iterations shows the strength of the genetic algorithm. Could this be further extended to even larger graphs?

2.2.2 Nine Surveillance Locations

For a recon mission, it is realistic to assume there are around nine points to be explored in one mission. If there are more points to be explored, the mission becomes hard to plan and to execute safely. Yet it is not hard to believe that in certain scenarios the ten points (base camp included) can be formulated as a complete graph where one can travel from any point to another. Such a complete graph on ten vertices further increases the number of possible strategies compared to the sparse example from Example 2.2.1.

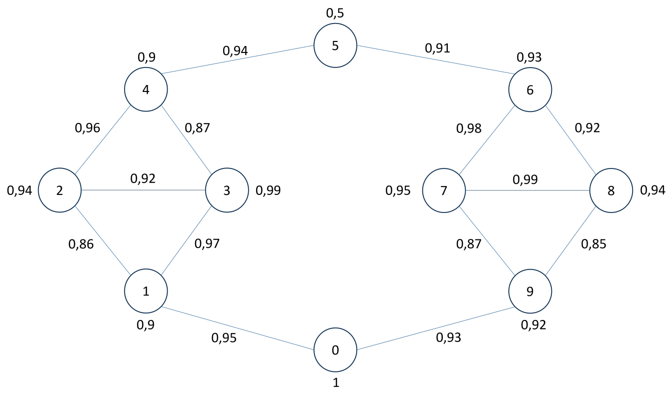

Figure 2.4: This graph is not complete. One could for example think of this graph as a river dividing the points in two and a bridge crossing this river. Naive initialisation and crossing methods don't work on this graph.

Example 2.2.2. The matrix which will be considered in this section is the following one:

$$\begin{bmatrix} 1 & 0.95 & 0.87 & 0.93 & 0.99 & 0.96 & 0.92 & 0.88 & 0.9 & 0.93 \\ 0.95 & 0.9 & 0.86 & 0.97 & 0.93 & 0.85 & 0.82 & 0.91 & 0.93 & 0.96 \\ 0.87 & 0.86 & 0.94 & 0.92 & 0.96 & 0.98 & 0.99 & 0.82 & 0.85 & 0.91 \\ 0.93 & 0.97 & 0.92 & 0.99 & 0.87 & 0.93 & 0.9 & 0.9 & 0.89 & 0.95 \\ 0.99 & 0.93 & 0.96 & 0.87 & 0.9 & 0.94 & 0.82 & 0.85 & 0.92 & 0.9 \\ 0.96 & 0.85 & 0.98 & 0.93 & 0.94 & 0.95 & 0.91 & 0.92 & 0.91 & 0.96 \\ 0.92 & 0.82 & 0.99 & 0.9 & 0.82 & 0.91 & 0.93 & 0.98 & 0.92 & 0.93 \\ 0.88 & 0.91 & 0.82 & 0.9 & 0.85 & 0.92 & 0.98 & 0.95 & 0.99 & 0.87 \\ 0.9 & 0.93 & 0.85 & 0.89 & 0.92 & 0.91 & 0.92 & 0.99 & 0.94 & 0.85 \\ 0.93 & 0.96 & 0.91 & 0.95 & 0.9 & 0.96 & 0.93 & 0.87 & 0.85 & 0.92 \end{bmatrix}.$$

The previously used parameters don't suffice anymore to find the optimal strategy. Nevertheless, when increasing the number of generations and the number of strategies per generation, the optimal one can be found. Specifically, the genetic algorithm was used with 200 generations of 1000 routes. In each iteration, the best 50 routes and the worst 40 routes are separated. The first generation was again made with random walks of length between 10-4 and 10+4. This yields the following best strategy:

$$\begin{aligned} \text{Route: } & 0405267813930 \\ \text{Send: } & 0010001001011, \end{aligned}$$

with value 7.305181.

Out of 100 runs, this planning is found 19 times. Compared to the previous graphs - where the optimal strategy was almost always found - this is not a high success rate. With this performance, a lot of runs are required to be pretty sure of finding the optimal planning. If we want to have a probability greater or equal than 0.95 to find the optimal strategy assuming a success chance at every try of 19/100, we need to solve the following equation:

$$1 - \left(\frac{81}{100}\right)^n \geq 0.95,$$

which gives that $n \geq 15$. While this is not an infeasible number of runs, it would be more compelling to increase the success rate of a single try.
To see how the genetic algorithm behaves, one can look at the best value of the strategies in each generation. This is depicted in Figure 2.5 for four runs of the algorithm. The x-axis shows the generation, the y-axis the best expected value. In all four figures, it is clear that the most progress is in the first generations of the algorithm. After a while no further progress is made, even though the optimal value has not been found yet. One of the plots only finds an optimal value of approximately 7, which is quite far away from the optimal 7.305181.

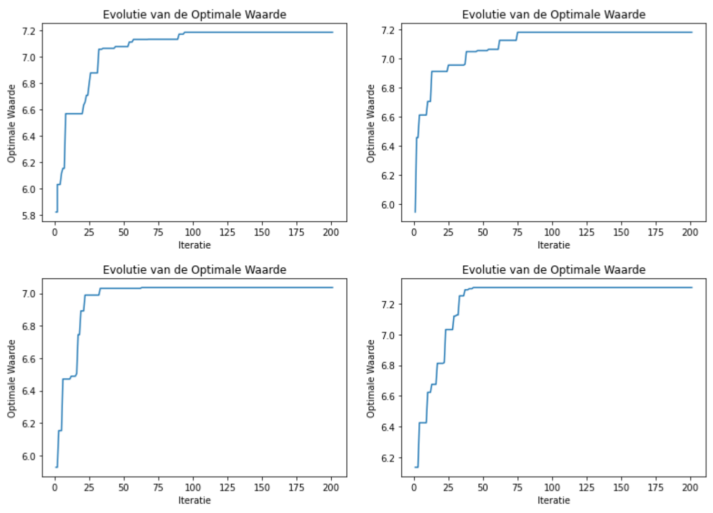

Figure 2.5: Four runs of the genetic algorithm for Example 2.2.2. The x-axis shows the generation, the y-axis the best expected value. The plot in the bottom right corner is of a run that finds the optimal solution, the other ones get stuck in a local maximum.

The plots show that the algorithm gets stuck more often in local optima. This could be caused by a lack of variation in the different generations which means that other routes are left unexplored. Indeed, the 200th generation of the algorithm contains almost always exactly only one strategy that fills the entire list: the (local) maximum where it got stuck. Thus it is clear that the variation throughout the algorithm should be increased to hopefully find the optimal strategy more often.
To emphasize the strength of the genetic program and at the same time investigate the sudden drop in performance for a graph with nine surveillance locations, it is worth noting how many possible strategies there actually exist. This enables comparing the searched space with the entire solution space.
A feasible strategy consists of a feasible route and a feasible send strategy. We define a feasible route as a route which begins and ends at the base vertex and passes via all other vertices at least once. Moreover, it is assumed that the maximum length of a route is twice the total number of vertices. This leads to a lower bound on the total number of possibly considered strategies by the algorithm, but makes the computations easier. A feasible send strategy needs to have the same length as the corresponding route. The first entry should be a 0, the last one a 1.

Proposition 2.2.1. Let the number of possible routes of fixed length n in K_m, $n > m$, be denoted by $\mathcal{R}_{m,n}$. Then the following inequality holds:

$$\mathcal{R}_{m,n} \geq \binom{n-2}{m-1} \cdot (m-1)!$$

Proof. This inequality follows from the fact that every vertex that is not the base camp has to be visited at least once in the route. As the first and last vertex in each route is the base camp, there are $\binom{n-2}{m-1}$ ways to pick $m-1$ locations in the route where these vertices are visited once. The $(m-1)!$ term follows from the number of ways to order the $m-1$ unique vertices over the chosen locations. □

Proposition 2.2.2. Let the number of possible strategies of fixed length n in K_m, $n > m$, be denoted by $\mathcal{S}_{m,n}$. Then the following inequality holds:

$$\mathcal{S}_{m,n} \geq \binom{n-2}{m-1} \cdot (m-1)! \cdot 2^{n-2}.$$

Proof. This follows immediately from Proposition 2.2.1 and the fact that there are 2^{n-2} possible send strategies for a route of length n. □

2.2.3 Computation of the Number of Feasible Solutions

Proposition 2.2.2 shows that the number of feasible solutions increases exponentially. Thus the number of possible solutions for K_{10} is way larger than the number of possible solutions for K_6.

- For K_6 the genetic program consisted of 60 generations of 500 strategies, so at most 30000 strategies were explored. Proposition 2.2.2 gives a lower bound of 40 638 720 possible strategies in K_6. This implies that at most a fraction of

$$\frac{30000}{40638720} \approx 7.382122 \cdot 10^{-4}$$

of the total solution space is searched.

Length of route	# routes	# send strategies	Total
7	$5!$	2^5	3840
8	$\binom{6}{5} \cdot 5!$	2^6	46 080
9	$\binom{7}{5} \cdot 5!$	2^7	161 280
10	$\binom{8}{5} \cdot 5!$	2^8	1 720 320
11	$\binom{9}{5} \cdot 5!$	2^9	7 741 440
12	$\binom{10}{5} \cdot 5!$	2^{10}	30 965 760
Sum of all strategies:			40 638 720

Table 2.1: Computations to find a lower bound on the number of possible strategies in K_6. For every length of the route the lower bound given by Proposition 2.2.1 is given in the second column. The exact value of the total number of send strategies is given in the third column. The product of both that makes up the total number of strategies of this length is given in the last column.

- For K_{10}, where the genetic algorithm computed 200 generations of 1000 strategies, the same analysis can be made. The fraction of solution space that is explored by the genetic algoirthm is upper bounded by

$$\frac{200000}{6127308575539200} \approx 3.264076 \cdot 10^{-11}.$$

Length of route	# routes	# send strategies	Total
11	$9!$	2^9	185 794 560
12	$\binom{10}{9} \cdot 9! \cdot 8$	2^{10}	3 715 891 200
13	$\binom{11}{9} \cdot 9! \cdot 8^2$	2^{11}	40 874 803 200
14	$\binom{12}{9} \cdot 9! \cdot 8^3$	2^{12}	326 998 425 600
15	$\binom{13}{9} \cdot 9! \cdot 8^4$	2^{13}	2 125 489 766 400
16	$\binom{14}{9} \cdot 9! \cdot 8^5$	2^{14}	11 902 742 691 840
17	$\binom{15}{9} \cdot 9! \cdot 8^6$	2^{15}	59 513 713 459 200
18	$\binom{16}{9} \cdot 9! \cdot 8^7$	2^{16}	272 062 690 099 200
19	$\binom{17}{9} \cdot 9! \cdot 8^8$	2^{17}	1 156 266 432 921 600
20	$\binom{18}{9} \cdot 9! \cdot 8^9$	2^{18}	4 625 065 731 686 400
Sum of all strategies:			6 127 308 575 539 200

Table 2.2: Computations to find a lower bound on the number of possible strategies in K_{10}. For every length of the route the lower bound given by Proposition 2.2.1 is given in the second column. The exact value of the total number of send strategies is given in the third column. The product of both that makes up the total number of strategies of this length is given in the last column.

The above mentioned fractions show that it is stunning that the genetic algorithm manages to repeatedly find the same strategy that maximises the expected value. The large gap between the K_6 and K_{10} fractions also explain why it is not that surprising that the algorithm doesn't perform equally well.

2.3 Improving the Genetic Algorithm

2.3.1 Variation through Crossovers

It is clear that the complete graph on ten vertices requires some more creativity to increase variation throughout the algorithm. A first method is to include longer routes in the first generation. Even though these longer routes are not necessarily better than the shorter ones – at some point all information has been transmitted and making an additional loop doesn't change the expected value – the final part of a route with useless and unnecessary walks can become useful after a crossover. Moreover, after running some experiments, it has become clear that it could be better not to have only optimal send strategies. For example, send strategies are added that are not completely filled with zeroes after all information has been transmitted. The send strategy can then be included in the crossing process of the genetic program. Then the parts that didn't count towards the expected value of some route can suddenly become relevant for the child strategy formed with this route.
To implement this, the first generation is filled with longer random routes. Moreover, to find the send strategy, a local search algorithm is used instead of the dynamic program. In such a local search, a random send strategy is constructed firstly. This is a vector filled with 0's and 1's of

the same length as the route. Next, all possible send strategies that can be obtained by flipping one 0 to a 1 or vice versa are considered. For all these new send strategies the expected value can be computed. The next considered send strategy is the one with the highest expected value out of those new strategies. This process is repeated until there is no more improvement possible. Note that this method does not guarantee a global optimum. It is possible that the search gets stuck in a local maximum while there is a different send strategy with an even better expected value. But this increases the variation of the algorithm.
To further increase the variation and to make the algorithm faster, these send strategies are only computed in the first generation. After this generation, the send strategies are also added to the crossing mechanism of the algorithm. So the send strategy of the child strategy is the combination of the send strategies of the parents. Note that this again does not imply that the send strategy for the kid is optimal. Yet the routes can be ranked based on these send strategies and the idea is that the genetic algorithm would eventually generate the best route with the best send strategy combined while exploring more solutions in the process.

Example 2.3.1. To construct the next generation, two routes are again picked at random. But this time, their respective send strategies are also considered. Suppose we pick two strategies

Strategy 1	Route:	015412342050
	Send:	001010010011
Strategy 2	Route:	030120250410
	Send:	011011011001

and say the crossover is made at vertex 2. With the send strategies added, this becomes:

Strategy 1	Route:	015412342050
	Send:	001010010011
Strategy 2	Route:	030120250410
	Send:	011011011001

Combining the first strategy up until and including vertex 2 with the second strategy after vertex 2 gives the child strategy:

Route: 0154120250410
Send: 0010101011001.

2.3.2 Variation through Mutations

While refraining from running the dynamic program in each iteration allows some gains in both variation and speed of the genetic algorithm, the greatest increase in variation is obtained by adding mutations. In this case, a route has a certain probability to mutate after the crossover. This mutation changes the route in some form and then the changed route is added to the new generation. Three mutations have been chosen to increase the number of considered strategies:

- *Added random walk.* Uniformly at random, one point that is not the last one is chosen in the route. After this point, a random walk of random length is added to the route. This mutation increases the length of the route.

- *Vertex flip.* One random vertex in the route is changed to some other vertex that wasn't a vertex directly before or after the flipped vertex.

- *Send flip.* Uniformly at random one element in the send strategy is flipped: a 0 becomes 1 or vice versa.

These mutations are not always performed and for each strategy, there is at most one mutation performed. The *added random walk* mutation is actualised with a probability of 0.01. Since this mutation increases the length of the route, a new send strategy has to be constructed as well. This is done by performing a local search for this new route. The initialisation of this local search on the send strategy is a random vector where an entry is zero with probability 2/3 and one with probability 1/3. The *vertex flip* mutation is performed with a probability of 0.2 and the *send flip* with a probability of 0.1. Since the length of the route doesn't change in either one, the send strategy is not changed after these mutations (this wouldn't make any sense for the *send flip* either).
Having implemented these mutations, the new version of the genetic algorithm can be tested. This was again done for Example 2.2.2 with 200 generations of 1000 routes, where the best 50 and the worst 40 routes are isolated. For four of these tests the plots of the evolution of the best expected value is given in Figure 2.6. The top two plots depict a run of the genetic algorithm where only a local maximum is found rather than the optimal value. In the bottom two plots, the optimal value is found. These plots are especially promising, as they show improvement in later generations too. In these runs the algorithm was able to explore enough other routes to get out of local maxima.

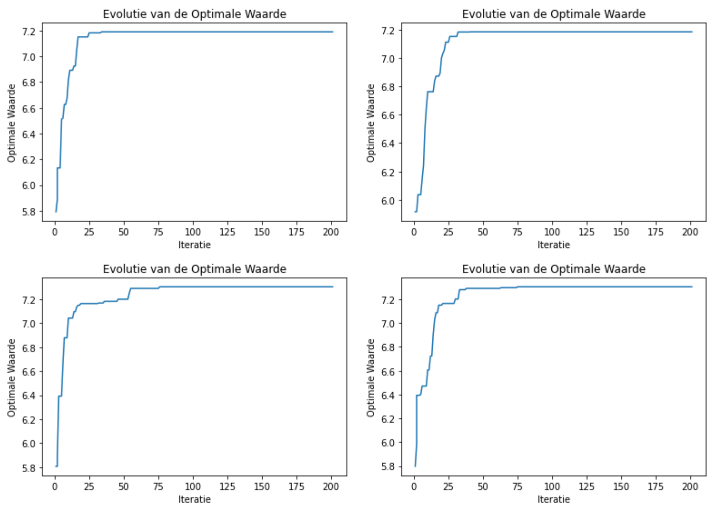

Figure 2.6: Four runs of the genetic algorithm with mutation. The x-axis shows the generation, the y-axis the best expected value. The bottom two runs find the optimal route and the optimal send strategy, the top two do not.

2.3.3 Comparison of the Different Versions

Figure 2.6 shows that the mutations can enable the genetic algorithm to get out of local optima and eventually find the best planning. But how often does this actually happen? To test this, the genetic algorithm was run a hundred times for Example 2.2.2. Out of these 100 runs, the optimal value was found 61 times. Given the fact that one run takes less than a minute, it is feasible to run the genetic algorithm ten times. Assuming that the optimal value is found with

a probability of 61/100 per run, the probability to find the the optimal value after ten runs is 0.9999.
To put this result into perspective, the comparison of four different genetic algorithms can be made. The first considered algorithm does not contain any mutations, the second one has a *vertex flip* mutation with a probability of 0.25. Then there is the genetic algorithm with only the *added random walk* mutation with probability of 0.01 and lastly there is the genetic algorithm with all mutations combined as explained above.

Genetic algorithm	# of successful runs
No mutations	19/100
Vertex flip	32/100
Added random walk	55/100
Combination	61/100

Table 2.3: Performance of different versions of the genetic algorithm on hundred runs for Example 2.2.2.

There is a clear difference in the performance between the different versions. Adding a mutation increases the probability of finding the optimal value. The vertex flip on its own does perform better, but 32/100 is still not amazing. Adding the random walk increases the odds of success more. In more than half of the runs the optimal planning was found. The genetic algorithm where three types of mutations are combined is still a little bit better, but since the genetic algorithm includes a lot of randomness, it is hard to determine whether this is actually better than only adding a random walk. Since both algorithms have no distinguishable speed difference, they seem interchangeable.
In any case, the version of the algorithm with the combination of the mutations provides a tool to quickly and accurately find the optimal planning for larger graphs.

Chapter 3

Multiple Drone Missions

Now that the genetic program has been tested and has proven its use for a model where one drone moves over the graph, the logical extension is to increase the number of drones. As already noted in subsection 1.1.2, the models for different numbers of multiple drones are very similar. Therefore this chapter will focus on two drones. This does lead to some new modelling choices and of course an adaptation of the existing algorithm. First of all the following modelling choices have been made:

- Both drones start from the safe base camp, where there is no information to be retrieved. The base camp is also the last vertex in the routes of both drones.
- Routes and send strategies are determined before the mission, so there is no way modifying the mission if a drone is intercepted by the enemy. Also, if this happens, it is assumed that the other drone can continue its mission.
- Both drones are allowed to retrieve the same information. This means that information is more considered to be a picture taken at a location rather than a package that is retrieved.
- If information is successfully transmitted by a drone, the other drone can still submit the same information. But t his information should not be double counted. The expected value of transmitted information per vertex does increase if its information is sent twice, but can never exceed the total amount of information available at that specific vertex.

The last assumption was the reason that the formula for the expected value in subsection 1.1.2 has been constructed. In the case of multiple drones the expected value is always computed as the sum of the expected amounts of transmitted information from each vertex.

3.1 Adapting the Genetic Algorithm for Multiple Drones

Having the formula for the expected value when deploying multiple drones from subsection 1.1.2, the different strategies can be ranked in a genetic program. Therefore the existing algorithm can be adapted such that it considers strategies for the wanted number of drones.

Initialisation

The main idea behind the initialisation is the same as before. The algorithm randomly construct walks through the graph of a random length between the number of vertices minus four and the number of vertices plus 100. Again, if the walk does not end back at the base camp, Dijkstra's algorithm is used to return. The send strategy is found using a local search algorithm based on the total expected value combining both drones. This local search is started with a random vector where an entry is zero with probability 2/3 and one with probability 1/3. The obvious difference now is that two routes and two send strategies are needed to form one gene. That means that for 1000 genes in the first generation, 2000 routes should be created. Since this takes longer in comparison with the time of performing the crossovers, the genetic algorithm has been adapted to enable differentiation between the number of genes in the first generation and all following generations.

Crossover

The crossover is very similar to the crossover in the single drone case. Two strategies are picked to be the parents of a new strategy. Every parent now consists of two routes and send strategies, so the first step is to randomly order the routes belonging the father and accordingly sort the corresponding send strategies. Now we pairwise cross the first routes and send strategies of the parents and the second routes and send strategies to obtain the crossing.

Mutations

The mutations have the same flavour as in the single drone case. First of all there is the *Added random walk*. This is the same mutation as described in subsection 2.3.2: one of the two routes is picked and a random walk is added somewhere in the walk. This continues to be a useful mutation to increase the variation in the genetic program. But it slows the algorithm down, because it requires a local search for the optimal send strategy in terms of the expected value for two drones. Therefore this mutation is only actualised with a low probability. If a generation consists of n strategies, the mutation probability is set around $\frac{2}{n}$.
A second important remark that can be made is that with the formulation of the expected value as it is now, both drones are incentivized to still travel to all vertices of the graph and try to gather information everywhere. However, it does make sense that the drones fly more or less in 'opposite directions' through the graph. Then more vertices have a high probability of their information being successfully transmitted by at least one of the drones. This inspired the *reversed* mutation. In this mutation, exactly one of the two routes is completely reversed. The same holds for the corresponding send strategy. This mutation is way cheaper than the other one so this is actualised with 20% chance.

3.2 Results

To test the performance of the algorithm for two drones, the same approach as for one drone is followed. The considered graphs will get larger step by step, increasing the number of possible strategies. First, the K_4 example is considered, then the K_6 example, followed by the sparse graph on ten vertices and finally K_{10}.

3.2.1 Three Surveillance Locations

Returning to Example 1.1.1, this time it is assumed that two drones are deployed to gather information. This might seem a bit overkill, as there are only three locations to be surveyed by our drones. But it forms a small and suitable first test for the genetic program. For this smaller graph, the parameters for the genetic program don't have to be large either: all 30 generations consist of 200 strategies, of which the best 15 are directly put in the next generation and the worst ten are not taken into consideration. The optimal strategy given by this approach is the following:

Drone 1	Route:	0232010
	Send:	0000111
Drone 2	Route:	0232010
	Send:	0000111

The expected value of this strategy is 2.346529. This is quite a large increase compared to the expected value of 1.666494 with one drone. However, from a practical point of view this strategy raises some questions. Both drones follow the same route and the same send strategy, namely the optimal strategy for one drone. This doesn't seem to be practically feasible while still assuming independence in the crossing probabilities for both drones. Nevertheless, under these modelling assumptions the genetic program manages to find this optimal solution ten out of ten runs. So even though the practical applications for two drones to survey three locations are rather limited, the genetic program does perform well.

3.2.2 Five Surveillance Locations

The model becomes more interesting in the case of five surveillance locations. For such a mission, there are way more routes to be considered and it makes sense to not let the two drones follow the same strategy. As there are more options, the parameters of the genetic algorithm have to be increased too. For the test on this K_6 described in Example 2.1.1, all 100 generations have 1000 strategies. The best 50 and the worst 40 are separated. This results in the following optimal strategy of expected value 4.859738:

Drone 1	Route:	030504210
	Send:	001010101
Drone 2	Route:	010425030
	Send:	001010101

This strategy is also found in ten out of ten runs. The genetic program performs well in finding the optimal strategy. There are a couple of things to note from this strategy. First of all, the two drones do not follow the same strategy. This makes intuitive sense, as it seems more reasonable to divide and conquer. However, there is no penalty for the drones for surveying all locations in their route, because the objective is solely to maximise the expected value of retrieved information. Therefore the two routes indeed go past all surveillance locations, but the order seems almost reversed. This seems logical, both drones first focus on gathering information at different vertices. Hence these first vertices have a higher probability of getting reached by at least one drone and thus contribute more to the expected value. After having visited their first vertices, the drones take the optimal route to the group of vertices already visited by the other drone, further increasing the total expected value. Secondly, the optimal strategy for one drone from subsection 2.1.1 is not copied by one of the two drones. Rather, both drones use a different strategy to optimise their joint efforts. Lastly, it is worth noting the expected value is 4.859838, while there are 5 possible units to be retrieved by the drones. This shows that using more than two drones to survey five locations is not an efficient use of resources. These two drones are already expected to retrieve almost all wanted information. Considering the expected value for one drone of 4.115090, the question could be asked whether two drones are really needed. The answer to this question seems to be specific to the objectives and resources for each distinct mission.

3.2.3 Sparse Graph on 10 Vertices

With the new insights from the previous section, looking at Figure 2.4 leads to the natural prediction that both drones start of in a different direction and first focus on a different cluster of vertices, before crossing to the other cluster. Using the same parameters as for K_6 (1000 strategies per generation, separate the 50 best and 40 worst and repeat for 100 generations) shows that this is indeed true. The optimal strategy is again found in ten out of ten runs and is given by:

Drone 1	Route:	013245678790
	Send:	001000010101
Drone 2	Route:	09787654231310
	Send:	00000100110101

The expected value of this strategy is 7.981840. In comparison to the expected value of the one drone scenario, 6.113745, this is a large value. This type of graph is an example of a mission that is very suitable for two drone being used. Curiously, the strategy for drone 1 from subsection 2.2.1 is equal to the optimal strategy for the one drone case. The fact that the second drone increases the probability of also exploring vertices 6-9 yields this large improvement.

3.2.4 Nine Surveillance Locations

Up until now, all optimal strategies were quickly and consistently found by the genetic algorithm. This changes drastically when considering Example 2.2.2. For this graph the success rate of the genetic program decreases. The best found strategy is

Drone 1	Route:	0526787625931040
	Send:	0000001000010101
Drone 2	Route:	0401395267876240
	Send:	0010100001010001

with expected value 8.653276. The genetic algorithm was run over three hundred times with different parameters and this was the best strategy that was found. Moreover, it was found

more than once, which leads to believe that this is the optimal strategy. However, just as in the single drone case, the success rate dropped significantly. Furthermore, the genetic algorithm is slow because it needs to explore more and larger strategies. Hence a balance between speed and reliability has to be found. The parameters that were chosen take into account this trade-off. The first generation consist of 700 strategies, all following generations consist of 1200 strategies. Each time a new generation is computed, the best 50 strategies are copied and the worst 30 are not taken into account. In total, 100 generations are computed. With these parameters, the best route is found in around two out of ten times.
To investigate whether the performance of the genetic algorithm on K_{10} can be improved, the evolution of the best found value throughout the algorithm is again plotted. This gives insight in the capability of the algorithm to get out of local maxima. Four of these plots are shown in Figure 3.1.

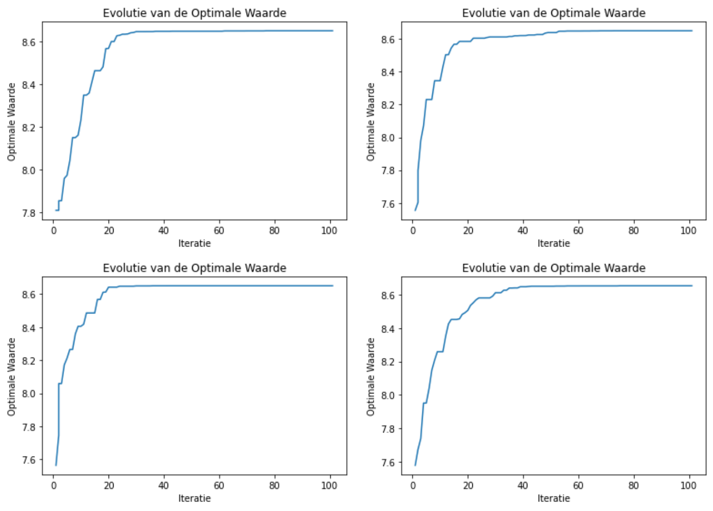

Figure 3.1: Four runs of the genetic algorithm for K_{10} The x-axis shows the generation, the y-axis the best found expected value. The only run that was able to find the global maximum is at the bottom right.

Looking at these evolution plots, it is hard to distinguish which run actually found the optimal value and which didn't. Even if the optimal value isn't found, the local maximum that is obtained instead comes very close to the best possible expected value. This is great from a practical viewpoint, as the proposed strategy will always have a high expected value. But it would be more satisfactory to find the optimum more often. It can be noted that the four plots have the same behaviour: a steep increase in the first thirty iterations, less change afterwards. Comparing this with the algorithm for one drone, where the steep increase lasted approximately eighty iterations, it seems that the current algorithm is not able to explore enough different routes to get out of local maxima. The most important tool so far for exploring other routes is the added random walk mutation. However, this mutation takes a while to compute as this random walk has to be generated and then the send strategy has to be recomputed as well. In order to keep the program quick - at this point the program can be run in a couple of minutes -, the probability of this mutation has to be kept low (around 0.001). Increasing this value increases the exploration, but also the run time.

Since additional exploration is mostly required in the later generations and a higher mutation probability implies a higher running time, it could be beneficial to use an adjustable probability based on the considered generation. So for example, the mutation probability can be computed with the following function:

$$f(x) = \frac{\tanh(0.1 \cdot (x - 50))}{500} + \frac{1}{500},$$

where the variable x denotes the generation. This implies the domain is given by $[0, 100]$. The codomain is then approximately $[0, 0.004]$ with a steep increase around $x = 50$. See also Figure 3.2 for a plot of the function. Hence the second half of generations has a way larger probability of mutation that the first half. This is exactly what is wanted. However, the results are not as hoped. Instead of a higher success rate to find the optimal value, this rate stays at $2/10$ successful runs. When looking at the evolution plots in Figure 3.3, it becomes clear what is happening.

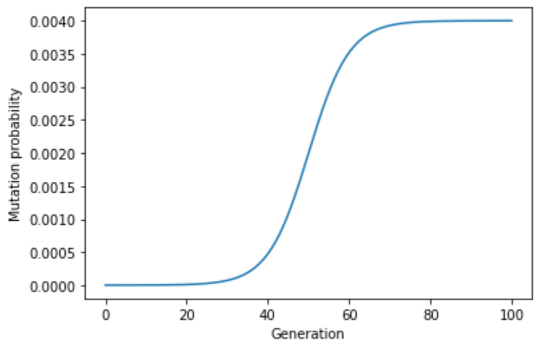

Figure 3.2: Function that relates the adaptable mutation probability with the generations.

The initial increase, which was very steep in the case of a fixed mutation probability, is now way more spread out. This means that even though the later generations have a higher probability of still finding a better strategy than the previous one, the optimal value is not found more often. The only solution seems to be to let the mutation probability stay one fixed value, but a higher one that the one that is used now. To make significant progress, it seems that the probability has to be increased to a value that makes the algorithm so slow that it is not feasible to test whether it actually performs better. Since this alternative mutation probability did not increase the performance of the genetic algorithm, the best code remains to have a success rate of around 20%. In practical applications this is sufficient. But if it is desired to increase this, it seems that the entire algorithm needs to be rewritten in such a way that it becomes faster. This would be a good first step in future research.

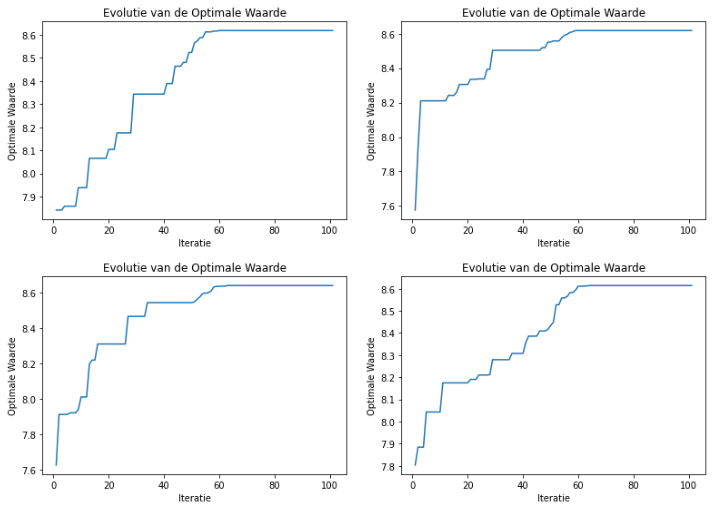

Figure 3.3: Four runs of the genetic program with adjustable mutation probability. More progress is made in the later generations compared to the runs with fixed mutation probability. However, the initial increase is way less steep.

Chapter 4

Summary and Further Work

The ongoing wars in Ukraine and Gaza have shown the importance of unmanned vehicles in modern warfare. These drones and carts are not only used in offensive missions, but also in survey missions. There are multiple advantages of using them instead of manned missions for surveillance. First of all, surveillance missions are dangerous and the use of autonomous systems reduces the risk for our troops. And there are practical applications too. Because of its speed, a drone can cover more distance in less time and it is cheaper to deploy a lot of drones instead of a lot of manned missions. Therefore the covered ground increases. In a reality where unmanned vehicles are gaining importance on the battlefield and a lot of gains can be obtained by smartly deploying them, it is important that the Dutch military does not fall behind and explores all viable automated possibilities.
In that regard, this reports investigated how to find the optimal strategy for an autonomous reconnaissance mission. Specifically, two questions have to be considered to obtain the optimal strategy.

- In which order should the different locations be travelled to?
- Where is it beneficial to make a transmission and where is it better to hold on to the gathered information?

To figure out the answer to the above research questions, a mathematical model is proposed in chapter 1. The enemy terrain can be represented by a weighted graph where the vertices are given by the base camp and surveillance locations. The different detection probabilities and the information to be gathered become weights on the graphs. The objective then becomes to find a walk through the graph together with a send strategy that maximises the expected value of retrieved information.
Using this mathematical formulation, it was hoped that the problem could be solved as an Integer Linear Program. However, it became clear that such a program would be complex and seems to require a lot of decision variables. Therefore, the approach of a genetic algorithm was preferred. This algorithm has been studied in great detail in chapter 2, where it is assumed that one drone is deployed. A first and more simple version of the algorithm worked neatly on smaller graphs, but when the number of feasible strategies was increased, it had to be improved. The number of searched solutions was augmented by implementing a local search algorithm and by adding mutations. The new algorithm performed satisfactorily on larger graphs as well, providing a quick and reliable method to propose the optimal strategy for single drone missions.

Having obtained a solver for one autonomous system, the scenario was made even harder by considering multiple systems during one mission in chapter 3. Instead of one route and one send strategy, two of each are wanted, that jointly maximise the expected value of retrieved information. This squares the number of possible strategies, making it less likely to find the optimal one. The existing algorithm was adapted to this scenario and continued to perform extremely well on smaller graphs. But for K_{10}, the algorithm finally met its limits. Even with further improvements, the optimal strategy is only found in around 20% of the runs. From a practical point of view this is not a disaster - the returned strategies come very close to the optimal one in terms of expectation -, but from a mathematical viewpoint this leads to some open research questions.

First of all, it is clear that there is still room for improvement regarding the genetic program. In the two drone case on K_{10}, the mutation probability is kept low, because other run time becomes too high. If the algorithm is further optimised, perhaps this probability can be increased. In the single drone case, a smart choice of mutations tripled the success rate of the algorithm. A logical next research step would be to explore whether this is also possible in the two drone case.

If it is discovered that the solution space has just become too large for a fast genetic program, another method could be researched. Maybe this research was not profound enough and there does exist an efficient formulation of an ILP to solve this problem. Also the possibility of a stochastic optimization program could be an interesting research topic [6]. In such a program, the objective function is allowed to be an expected value. Furthermore, the constraints can be probabilities. Finding a way to rewrite this as a linear program is an entire research field in itself and this optimal planning of reconnaissance missions would be an interesting case study for those methods.

Bibliography

[1] *2022/2023 World Civil Unmanned Aerial Systems Market Profile & Forecast.* Fairfax, Virginia, USA, Nov. 2023.

[2] Abe E. Boswinkel. "Het risico van informatie zenden in een vijandige omgeving beperken". Netherlands Defence Academy, 2023.

[3] Arthur P. Brill Jr. "Dragon drone deployment tests corps' UAV capabilities". In: *Sea Power* (1998).

[4] Tom Engelhardt. "Remotely Piloted War". In: *Foreign Policy in Focus* (2012). URL: `https://www.proquest.com/docview/954766118/fulltext`.

[5] Frank G. Hoffman. "Hybrid Threats: Reconceptualizing the Evolving Character of Modern Conflict". In: *Strategic Forum* (2009).

[6] Alexander Kogan and Miguel A. Lejeune. "Threshold Boolean form for joint probabilistic constraints with random technology matrix". In: *Mathematical Programming* (2013). DOI: `https://doi.org/10.1007/s10107-013-0728-y`.

[7] Sarah J. Lohman et al. *What Ukraine Taught NATO about Hybrid Warfare.* Carlisle Barracks, Pennsylvania, USA: Strategic Studies Institute and US Army War College Press, 2022.

[8] Frank Christian Sprengel. "Drones in hybrid warfare: Lessons from current battlefields (Working Paper)". In: *Hybrid CoE* (2021).

[9] Paul H. Williams. *Model Building in Mathematical Programming.* Wiley, 2013.