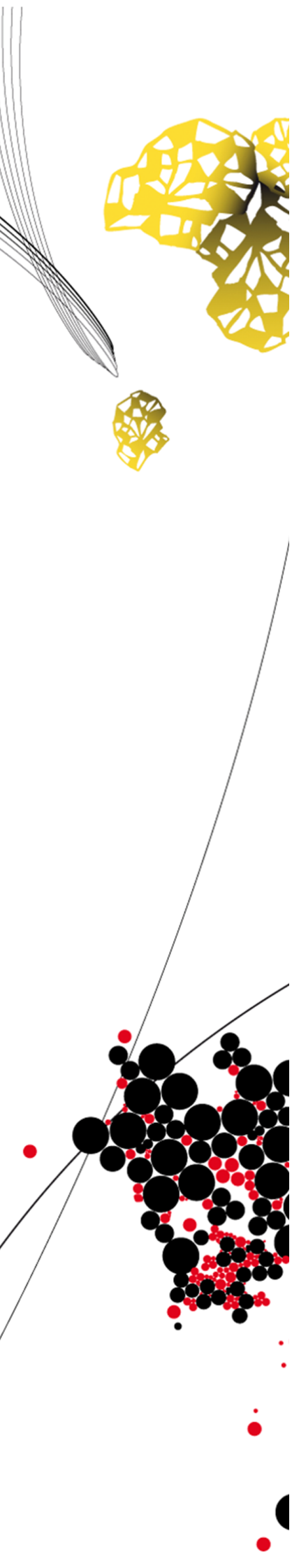

UNIVERSITY OF TWENTE.

Applied Mathematics

Solving The Multi-Sensor Assignment Problem Efficiently

Leander Christiaan van der Bijl

M.Sc. Thesis

September 2021

Supervisors University
prof.dr. A.A. Stoorvogel
dr. P.K. Mandal

Supervisors Thales Hengelo
D. Swart
P. Verveld

Preface

This graduation project is part of the Applied Mathematics program at the University of Twente. It was done in collaboration with Thales Hengelo.

I would like to thank my University supervisors P.K.Mandal and prof. dr. A.A. Stoorvogel. I also want to thank my supervisors from Thales Hengelo namely Dennis Swart and Perry Verveld. Lastly I would like to thank my friends and family for their support.

Summary

In this report, it is investigated whether the computational cost of solving the NP-hard Multi-Sensor Assignment Problem (MSAP) can be decreased. Currently, this problem is solved by the Kruger method, which first solves the problem as an assignment problem and then applies a Branch and Bound algorithm.

In this report, several other methods were created that use this same solving approach but try to make it more efficient and experiment with different strategies. Most of these methods gave significantly better performance than the Kruger method.

However, it was also investigated whether another approach could also be used, namely solving the MSAP as a Mixed Integer Problem (MIP). The commercial program Gurobi was used to test the performance of this approach. It turned out that Gurobi performed much better on most problems than all of the earlier tried methods. Because of this it is recommended to investigate the MIP approach further and see whether it can be fine-tuned specifically for an MSAP such that it also gives better performance for small problems.

Contents

Chapter 1

Introduction

This graduation project will be about the Multi-Sensor Assignment Problem, a problem that will be explained in the first part of this report. The problem is often encountered when working with radars, especially when considering the company Thales (specifically Thales Hengelo) where this graduation project took place. This section will therefore discuss the basic principles of a radar to give some intuition of the application of this research project.

The Multi-Sensor Assignment Problem has been investigated before by Kruger [1] and Sibma [2]. This project can therefore be seen as a continuation of these papers. Furthermore, inspiration from these papers will be taken with regards to structure and terminology to preserve continuity. Though some definitions have been slightly altered when this was deemed necessary.

The report is structured in the following way: this introduction will start by describing the practical problem that gave motive for this research, while simultaneously giving the mathematical foundations such that this problem can then be formally defined. After this, the research problem will be formally defined in section 2. Section 3 will then attempt to solve the research problem by adjusting the method that is currently used to solve this problem. Section 4 will consider another approach to solve the research problem without making use of the current method. The conclusion and recommendations will then be given in Section 5. Finally, a table with symbols and a table with abbreviations are given in Section 6. In this report, it is assumed that the reader has basic knowledge of statistics and is familiar with several types of optimization problems and corresponding solving techniques (such as linear programs and mixed integer programs).

This introduction will now start with explaining the basiscs of the radar. It will then proceed with explaining a simpler version of the Multi-Sensor Assignment Problem and how this can be solved.

1.1 The radar

As stated in the name of the problem, research will be related to sensors. This section will start by briefly describing the technical details behind the radar, which is a specific type of sensor, to give the reader an idea of the applications and relevance of this research. During the explanation, related definitions and theorems will be given.

In practice, radars are used to detect objects. Objects are entities that have a location within a certain time interval. In practice, the time interval is equal to the time that these objects are within a detectable region. The objects can be flying objects, such as airplanes, but also objects on land or sea such as boats or cars. However, it is also possible for radars to detect objects that are usually classified as uninteresting. Examples of these are clouds in the sky or windmills on land. Objects that are of interest are called targets. The formal definition of a target will now be given.

Definition 1. (Target) A **target** is an entity that can be represented by a state function $\mathcal{F}(t) : \mathcal{T} \rightarrow \mathbb{R}^d$, where $\mathcal{T}$ denotes a finite set of points in time, namely all points in time at which the target is detected by a radar.

A radar detects objects by transmitting electromagnetic waves. When an object is hit by these waves, the waves are sent back to the radar which then receives this and processes the information that is contained within these waves. This information can be represented by the following vector: (R, A, E, D), where the range R is equal to the distance between the detected object and the radar, the elevation E is the angle between the radar and the object in the vertical direction (from the ground to the sky), the azimuth A is the angle in the horizontal direction and D represents the Doppler speed which can be used to find the speed of the object in the direction of the radar. A sketch of this is shown in Figure 1.1. In real life applications, these measurements are always the best estimate of the true state of the measured object. It must therefore be noted that there is always some error margin and an estimate of this error is usually given.

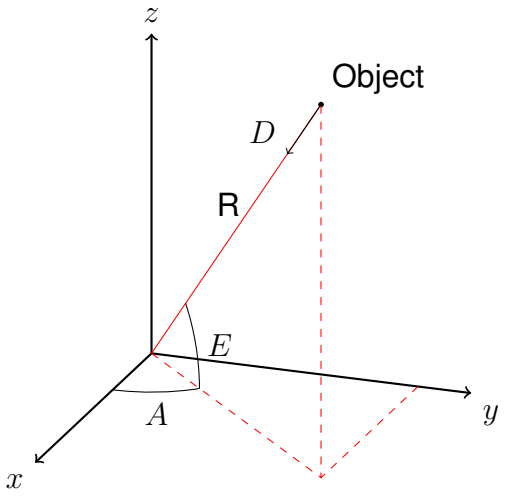

Figure 1.1: The information that is obtained from a detected object (polar coordinates).

The information in these measurements can then be mapped to the Cartesian coordinate system. A new set of data points (x, y, z) is therefore obtained, together with estimates of the velocities $(\dot{x}, \dot{y}, \dot{z})$ for which the Doppler speed is used. However, as the Doppler speed only returns information in one dimension, multiple measurements over time are required to obtain a good velocity estimate which has three dimensions.

Definition 2. (State vector) We consider the **state vector** $s(k)$ of an object to be the value of the state function for some discrete point in time k. Hence, $s(k) := \mathcal{F}(k) = (x, y, z, \dot{x}, \dot{y}, \dot{z})$. This represents the location and velocity of that target at discrete time point k.

With the help of Definition 2, the goal of the radar can now be rephrased as estimating the state function of targets that pass through the scanning area of that radar.

Definition 3. (Plot) A **plot** z is equal to a radar measurement that is originating from a target. The relation to the state vector of that target will later be specified.

Most radars periodically scan an area, which means that they start scanning at a certain sub-area a_0, then proceed to move to other instructed sub-areas until after some time t_s they are back at sub-area a_0. This process then repeats, a single cycle of this process is called a scan.

As enough time progresses, a radar will likely detect multiple plots. Now the question is, do these plots all belong to the same target? Or can they belong to different targets, and how to determine which plot belongs to which target?

Plots that originate from the same target need to be somehow grouped together. This will be accomplished by means of tracks, which will now be defined.

Definition 4. (Track) A track T_i is a set of plots.

Tracks are used to create a set of plots that are all thought to originate from the same target. Forming these tracks will be the main subject of this report. To generalize this further, the problem that will be dealt with in this report can be extended to the field of sensors. Because of this, the remaining part of this report will no longer mention radars but refer to sensors instead. All previously mentioned definitions remain unchanged, the term 'radar' is simply replaced with 'sensor'.

1.2 The Single-Sensor Assignment Problem

As briefly mentioned, the main problem of this report is heavily related to creating tracks. The problem is now explained intuitively before the mathematical definition will be given.

Each time that new plots are detected by a sensor, each of these new plots needs to be added to some track. This must be done in such a way that the likelihood that each track consists of plots that correspond with a single target is maximized. Furthermore, there should only be a single track for each target that passes through the scanning area of the sensor. In practice, multiple sensors can work together to do this with higher accuracy. However, this section will intuitively explain the problem when there is only a single sensor. From now on, this problem will be referred to as the Single-Sensor Assignment Problem (a formal definition will follow later in this chapter).

In order to formally define this problem and the required likelihoods, it is helpful to start with a simple case. When only one target is passing through the scanning area, creating a track is often straightforward but not as straightforward as one might think. It is not always correct to simply create a single track and add all plots to that track. This is because, in practice, it might occur that a plot has incorrectly been classified as originating from a target. Because of this, a new type of track must be created. This track will be referred to as a false alarm track (FA). Hence, when only one target is crossing the scanning area, each plot can either be originated from that target or it may be a false alarm.

An example will now be given. In this example, plots will be visually represented on a 2-dimensional grid. As this example is meant to give some intuition, it will be explained how a human might approach this problem and where the difficulties

would arise. The example can be seen in Figures 1.2 and 1.3. In these figures, the squares represent plots. The blue and yellow plots represent measurements that are assumed to be from the same target. The gray plots represent plots of which the origin is still unknown. Furthermore, the black plots represent plots that are assumed to be false alarms. Each of the figures consist of four scans. The scans are labeled from A to H, these labels are chronologically ordered. It can be seen that in each scan, new gray plots arrive. These grey plots will then be classified to either a target or a false alarm in the next scan. It can be seen in scan A that the first plot is found and this plot is then considered to be a false alarm in scan B, as it is just a single plot that has no other plots near it. However, the new plot in scan B seems to lie so close to the plot in scan A that it seems logical that they belong to the same target. This process then continues in the next couple of scans. It can be seen that a plot in scan D is again considered to be a false alarm and this time there is no reason to change this assumption. However, as the target that was marked by the color blue contiues its path, a new yellow target arises in the upper left. Deciding which plot belongs to which of these two targets seems straightforward until scan G. During this scan, the targets get so close that it becomes not so trivial to decide which plot belongs to which target. Newer scans barely give new information to clear this up as in scan H it is still not trivial to assign the plots to targets and this will probably continue to be the case for newer scans. It can therefore be concluded that it is not trivial to determine which plot corresponds to which target in general.

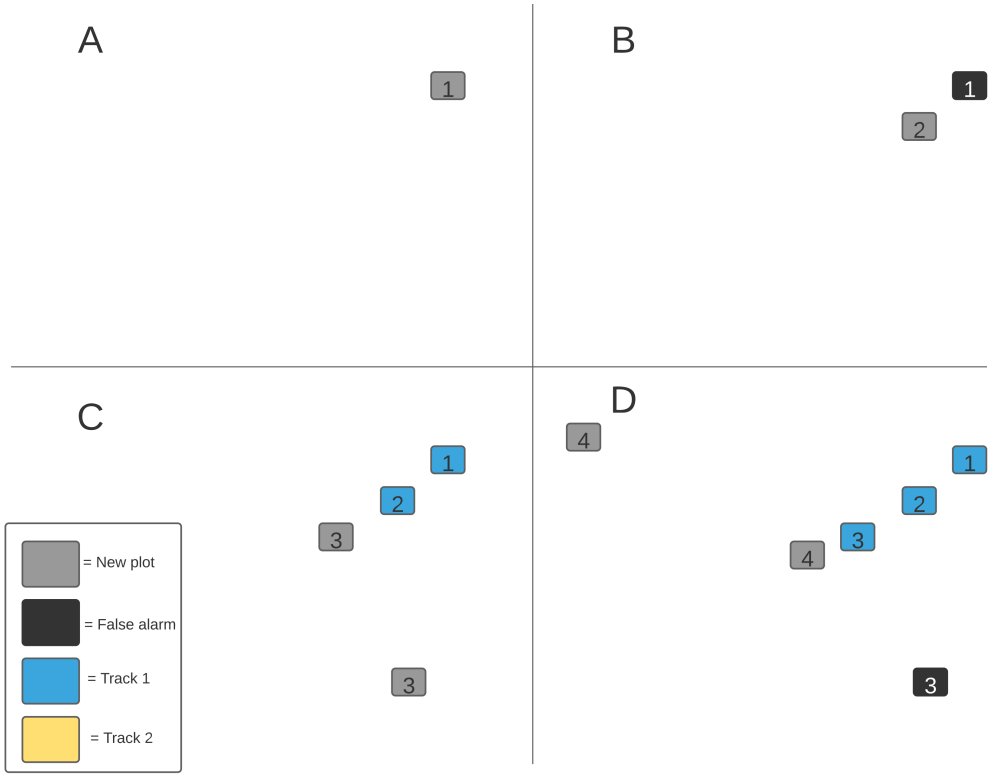

Figure 1.2: The first four scans of an assignment problem example.

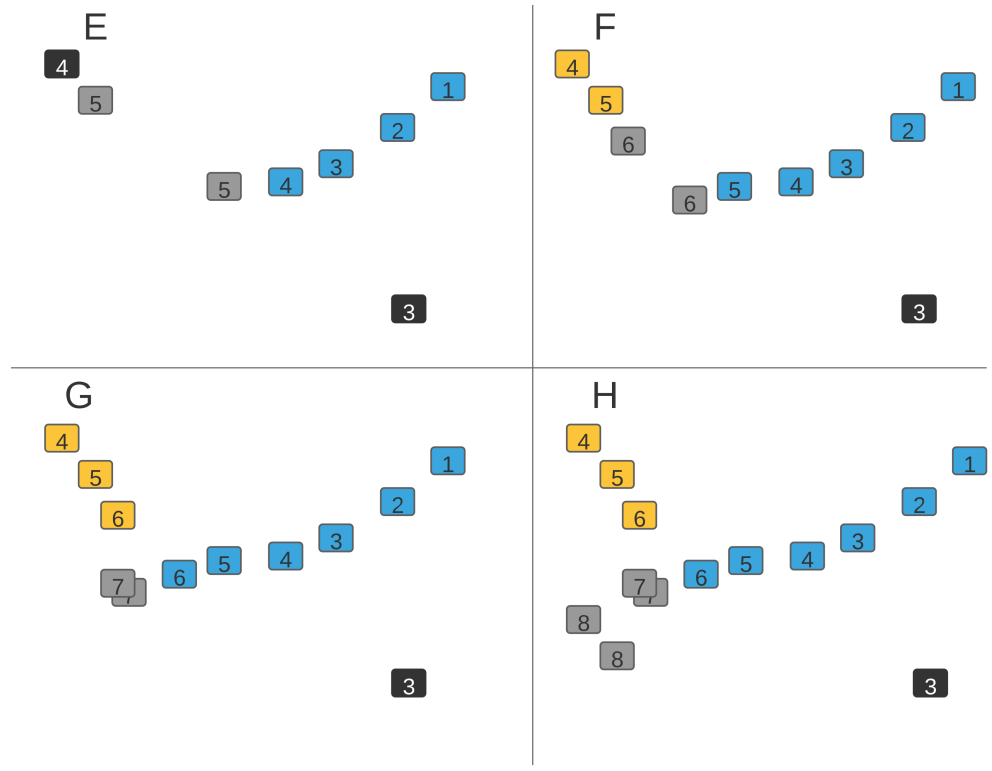

Figure 1.3: The last four scans of an assignment problem example.

This problem will now be formally defined. Let

$$Z(k) = \{z_m(k), m = 1, \ldots, M_k\}$$

be the k^{th} data set (i.e. the set of all plots that are detected during the k^{th} scan) where $z_m(k)$ is the m^{th} plot created in this data set and M_k is the number of plots in the k^{th} data set. Furthermore, let

$$Z^l = \{Z(1), Z(2), \ldots, Z(l)\}$$

be the cumulative set of the first l data sets. To solve the Single-Sensor Assignment Problem efficiently, several assumptions need to be made.

Assumption 1. The data sets will be pre-determined. This means that adjusting the time interval in which plots are recieved will not be considered in this report, as this is beyond the scope of this project.

Assumption 2. Each plot in a data set $Z(k)$ can only be produced by a single source. This means that at most one track can be assigned to each plot.

Assumption 3. Each target will produce at most one plot per radar in a data set $Z(k)$, which means that at most one of the plots can be assigned to each track.

The amount of hypotheses that this can produce will now be considered for a set of plots $Z(k)$ after the k^{th} data set. To start with a simple example, it is assumed that $k = 1$ and there are three plots, ie. $M_1 = 3$. Also, $k = 1$ implies that there are no existing tracks. This means that the first plot $z_1(1)$ can be either be a false alarm (FA) or a new track (NT). This gives two hypotheses. It was assumed that each source can only produce one plot at a time, hence no other plots can be part of this track. If the possibilities of $z_2(1)$ are then considered, it once again can only be a false alarm or a new track. This will therefore give four hypotheses in total. The resulting hypotheses tree after all three plots can be seen in Figure 1.4.

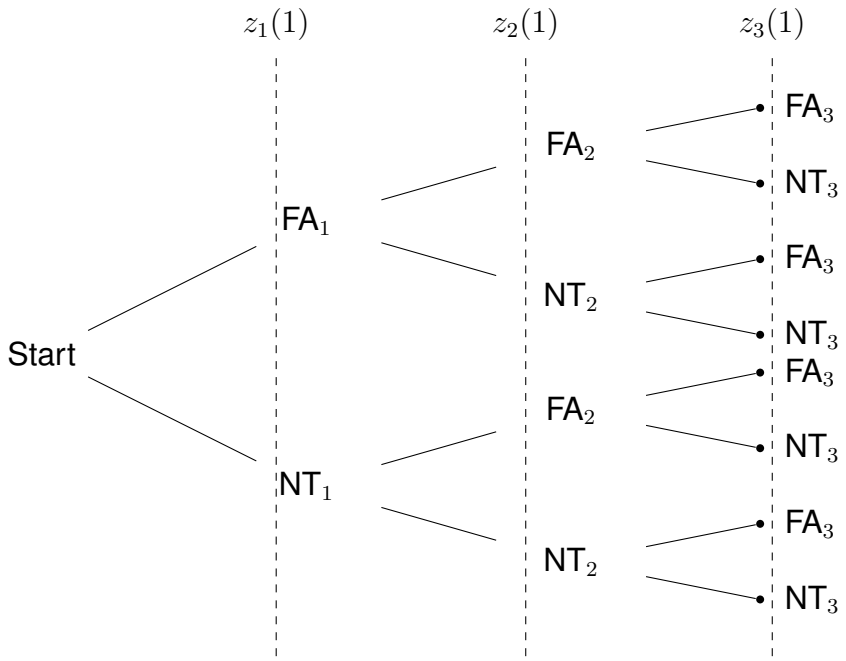

Figure 1.4: A hypothesis tree for three plots of the first scan.

The number of hypotheses will now be calculated. Let

$$\Omega^k = \{\Omega_i^k, i = 1, \ldots, I_k\} \tag{1.2.1}$$

denote the set of all possible hypotheses after the k^{th} data set where Ω_i^k refers to the i^{th} hypothesis of this set and where I_k is the total number of hypotheses at that point. If the logic of the previous hypotheses examples is applied further, it can be seen that the size of the hypotheses will follow the formula

$$|\Omega^1| = 2^{M_1},$$

and the maximum number of tracks at that time will be equal to M_1 (this happens if all plots produce a new track). This gets even worse for later moments in time as then plots can also be part of tracks that are created in previous time points.

1.3 The likelihoods

The Single-Sensor Assignment problem is still not formally defined at this point. This is because each of the hypotheses discussed in the last section somehow needs to have a value assigned to them that represents the likelihood of that hypothesis.

Otherwise it is not possible to create tracks in such a way that the likelihood is maximized. These likelihoods were efficiently found by Reid in [3], whose Multiple Hypothesis Tracking (MHT) method is still used today.

A short description of the MHT method will now be given. First it must be noted that there are two ways of applying the MHT method, namely the 'measurement oriented' and the 'track oriented' approach. At the moment Thales makes use of the measurement oriented approach. Both approaches solve the problem equivalently. However, it is still an open debate which approach is best. As finding out which approach is optimal is beyond the scope of this project, the measurement oriented approach will be assumed from now on.

For each target, a linear state model is used, which means that each target i will move according to

$$s_i(k+1) = \Phi s_i(k) + w_i(k),$$

where $s_i(k)$ is the target's state vector during scan k at time of detection $t_d(i,k)$. The state transition matrix Φ is used to calculate the dynamical behavior of the target. Furthermore, $w_i(k)$ denotes a white noise sequence with covariance matrix Q. These states are then related to the plots z in the following way:

$$z_i(k) = Hs_i(k) + v_i(k), \tag{1.3.1}$$

where the matrix H maps the target's actual state to the plot space, and again white noise is added by means of $v_i(k)$ with covariance matrix R. Furthermore, z_i is used to denote the plot instead of z to emphesize that this plot belongs to target i.

If there are no current tracks and a plot is found at estimated location $[x, y, z]$, a method is needed to compute the first predicted state. For simplicity, assume that the Doppler speed is not known. Such a prediction could then be equal to

$$\hat{s}_1(k) = [x, y, z, 0, 0, 0],$$

where also a covariance matrix $P_1(k)$ would be calculated for this approximation. When at least one active track is detected after scan k, a predicted state for each track is made for scan $k+1$ before receiving the actual result. This is done by means of a Kalman Filter. When scan $k+1$ is received, each of the plots will be compared to the estimated predicted states by means of a gating procedure. This means that plots are only able to be assigned to a track if the plot is located within a certain area based on the prediction of that track.

If $\hat{s}_j(k+1)$ and $\hat{P}_j(k+1)$ are the mean and covariance of the estimated state for track j with prior hypothesis Ω_i^k, then the plot $z_m(k+1)$ lies within an η-sigma validation region if

$$(z_m(k+1) - H\hat{s}_j(k+1))^T B_j^{-1}(k+1)(z_m(k+1) - H\hat{s_j(k+1)}) \leq \eta^2. \quad (1.3.2)$$

Inequality 1.3.2 is called the gating-criterion as stated by Reid in [3]. In this inequality, $B_j(k+1) = H\hat{P}_j(k+1)H^T + R$ and the value of η can be found by using a χ^2 distribution with q degrees of freedom where q is the size of the predicted state vector $\hat{s}$. A visual representation of this gating-criterion can also be seen in Figure 1.5.

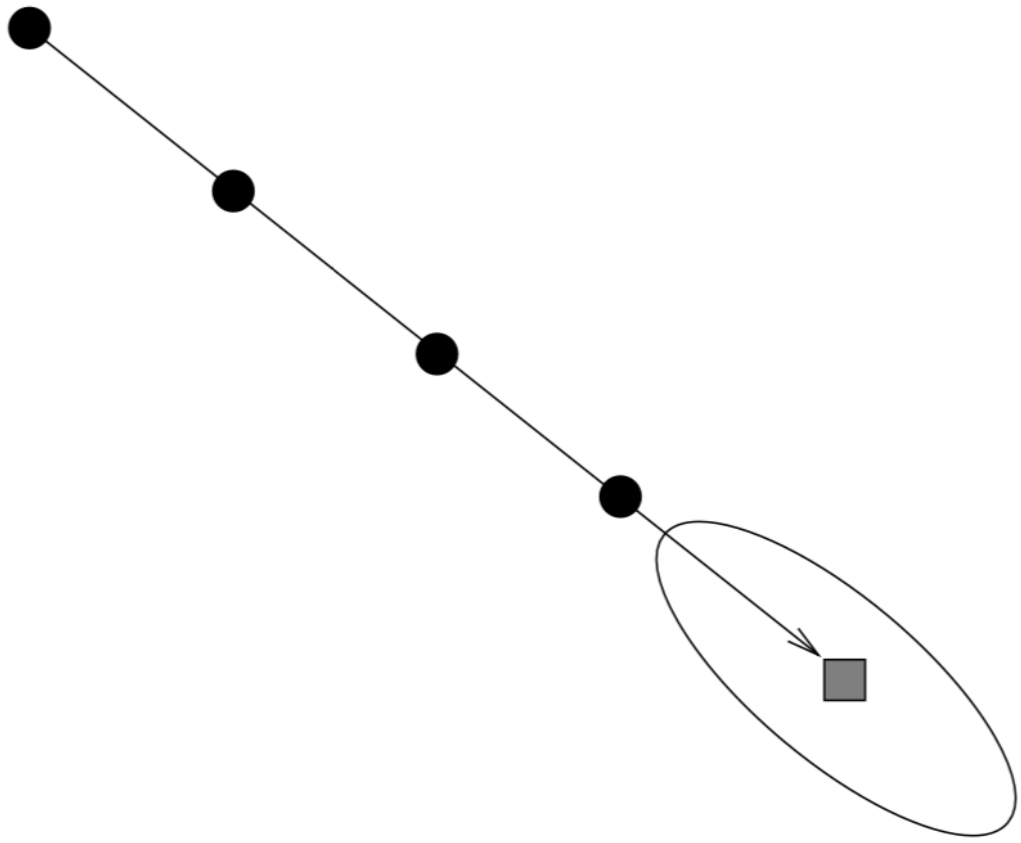

Figure 1.5: An example of gating during tracking as in [1], only plots in the ellipse wil be considered.

This gating-criterion will be used to find the area that was just discussed. A normal distribution can now be used to find the likelihood that a plot originated from a known target (as then it should be part of an existing track). The details of this can be found in section IV of [3].

1.4 The optimization problem

Now that it is possible to find the likelihood of a plot corresponding to a specific track, the Single-Sensor Assignment Problem can finally be described.

Definition 5. A **Single-Sensor Assignment Problem** is defined as the problem where the goal is to assign tracks to each plot $z_m(i)$ for all $m \in \{1, \ldots, M_i\}$ and $i \in \{1, \ldots k\}$ such that the likelihood of this assignment is maximized. These likelihoods are found by applying the MHT method. It is assumed that Assumptions 1, 2 and 3 hold.

Now that the problem is defined, some way of solving it must also be found. In order to do this, correlation matrices will be defined.

Definition 6. (Correlation Matrix)
A **correlation matrix** C is a matrix where each column represents a unique track and each row represents a plot. Each c_{ij} will be equal to the negative logarithm of the likelihood that plot i is assigned to track j, which will be called the correlation score of this assignment. If a plot does not fall within the η-sigma validation region of a track the likelihood will be set to 0, and therefore the corresponding correlation score will be equal to ∞. Not only existing tracks, but also false alarms and new tracks should be in this matrix for each individual plot. This means that each plot produces at least two columns in the correlation matrix: one new track and one false alarm.

Correlation matrices are used to store all likelihoods in one place such that further calculations can be performed. Figure 1.6 shows how such a correlation matrix may look like, it can be seen that each plot gets its own false alarm and new track option, this is because Assumption 1 would otherwise prevent to assign multiple plots to a new track. In practice, there are scenarios where all plots are a new track. This is also intuitively explainable because new tracks do not actually refer to the same physical track, they just refer to a previously unknown track which has to be labeled with a unique ID from that moment on.

$$\begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} & c_{1,6} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} & c_{2,6} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} & c_{3,6} \end{pmatrix}$$

Figure 1.6: An example of a correlation matrix for 3 plots. In this case, there are no existing tracks: the first three columns represent the new tracks of each plot, the last three columns represent the false alarms for each plot. This therefore results in 6 tracks total.

The matrix is chosen this way because it can now be written as an Assignment Problem, which is a well-known optimization problem in mathematical literature. Because any plot can only be part of one track, and each track can only produce one plot, the tracking problem with one sensor can be described as an Assignment Problem in the following way:

Solving the Single-Sensor Assignment Problem

Given a Correlation Matrix C, the corresponding Single-Sensor Assignment Problem can be formulated as the following optimization problem

$$\begin{aligned} \underset{x_{ij}}{\text{minimize}} \quad & \sum_i \sum_j c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = 1 \quad (i = 1, \ldots, n), \\ & \sum_{i=1}^{n} x_{ij} \leq 1 \quad (j = 1, \ldots, m), \\ & x_{ij} \in \{0, 1\}. \end{aligned} \tag{1.4.1}$$

Where the matrix $X = ((x_{ij}))$ will represent the optimal assignment, meaning that plot i is assigned to track j if $x_{ij} = 1$.

Argumentation. As the correlation matrix C consists of the negative logarithms of the assignment likelihoods, the problem should now be minimized to find the assignment with maximum likelihood. Furthermore, because of the logarithm and independence of the assignments, the likelihood of each individual assignment can be summed to obtain the likelihood of the total assignment. When using matrix notation for the assignment (by means of the matrix X) and combining this with Assumptions 2 and 3, optimization problem 1.4.1 is obtained. This same approach is used in [1].

The question now arises how quick the correlation matrices grow with respect to the amount of plots that are detected. For each plot, a new track and a false alarm column are required. Furthermore, each existing track requires its own column. Let A_T be the number of tracks and M be the number of plots during a scan. Then the size of the correlation matrix that corresponds to the Single-Sensor Assignment Problem of that scan can be found by using the following formula:

$$\begin{aligned} m &= M, \\ n &= A_T + 2M, \end{aligned} \tag{1.4.2}$$

where m is the amount of rows and n is the amount of columns of the correlation matrix.

This is not the typical assignment problem as there more columns than rows and therefore not all columns can be assigned, as then the row constraint would not hold by basic linear algebra laws. However, according to [4] this problem can be solved in $O(m^2n+m^2\log\ n)$. If the number of columns is much larger than the number of rows, this complexity reduces to $O(m^2n)$. This means that the Single-Sensor Assignment Problem can be solved in polynomial time.

1.5 Murty's k-best Method

A drawback of solving this assignment problem only for the best solution after each scan is that the best solution after scan i might no longer be the best solution after scan $i+1$, the best solution after scan $i+1$ might require some other assignment in the earlier scans. However, it is desirable to use a recursive algorithm as otherwise it will drastically increase the computational complexity with each new scan. Therefore it is decided to find not only the optimal solution but the k best solutions and consider all of them during the next scan's assignment problem. The k best solutions can be

found by using [5]. This method is also given in pseudo code below.

Data: An assignment problem P_0
Result: The k-best solutions of this assignment problem
1. Start by creating a list L with problem solution pairs $[\{P_0, S_0\}]$ where S_0 is the optimal solution of P_0 (can be found by applying Munkres).
2. Create a list R that is returning the k-best solutions when the algorithm finishes.
3. **for** $l = 1$ *to* k **do**
  if L *is empty* **then**
    Terminate the algorithm, no more solutions are possible.
  end
  3.1 Search through L and find the pair $\{P, S\}$ with the best solution value.
  3.2 Remove $\{P, S\}$ from L.
  3.3 Add S to R.
  3.4 **for** *each assignment* i, j *such that* x_{ij} *= 1 in* S **do**
    3.4.1 Let $P' = P$.
    3.4.2 Set c_{ij} to ∞ in P'.
    3.4.3 Find the best solution S' to P'.
    3.4.4 If S' exists, add $\{P', S'\}$ to L
    3.4.5 Remove row i and column j from the matrices C and X in the problem P. This reduces the dimension of the problem by one.
  end
end
4. Return S.

Algorithm 1: Murty's k-best method

Chapter 2

The Multi-Sensor Assignment Problem

This section describes the problem that is similar to the tracking problem in the previous section, except now there are S sensors, which means it is a more general case. In order to discuss this problem more efficiently, the next section introduces a new way to define a tracking problem.

2.1 Assignment tables

Solving a tracking problem requires the knowledge of the amount of tracks, the amount of plots, the possible combinations between those tracks and plots and also the negative logarithm of the likelihoods for these combinations including the likelihoods of false alarms and new tracks. All this information can be represented in an assignment table. An example of such a table is given in Table 2.1.

	T_1	NT_1	NT_2	FA	FA
z_1	$c_{1,1}$	$c_{1,2}$	∞	$c_{1,4}$	∞
z_2	$c_{2,1}$	∞	$c_{2,3}$	∞	$c_{2,5}$

Table 2.1: An example of a tracking matrix with two plots from one sensor.

Such an assignment table contains the same information as acorrelation matrix, which means that the rows represent the plots and the columns represent the tracks to which the plots can be assigned. Furthermore, the entries are filled with the negative logarithms of the likelihood of these combinations. However, in addition to this information, an assignment table also specifies the plot names and the track names such that the problem is easier to interpret.

The Assignment Table 2.1 represents the assignment problem where there is a single sensor, one existing track T_1 and two plots. Each of these plots can therefore be assigned to the track T_1. However, the plots can also be a new track denoted by NT_i where i denotes the corresponding plot. Furthermore, the plots can also be a false alarm FA. Because of the assignment format, each plot should have its own false alarm option. These are not indexed, since there is no point in keeping them in memory as no other plot will be assigned to that same track during later scans. Once a new track is assigned to a plot, that new track should be denoted as a track T_i (for an appropriate index i) in the next scan.

2.2 Correlated tracks

The multi-sensor tracking problem will now be described. In this problem, the retrieved data after scan k is denoted by

$$Z(k) = \{z_m^s(k), s \in \{1, \dots, S\}, m = 1, \dots, M_k^s\},$$

where s denotes which sensors the plots $z_m^s(k)$ originated from. It is still assumed that Assumption 1 holds. This means that several plots that are detected in a predefined interval will be given from multiple sensors and the goal is to maximize the likelihoods when assigning plots to tracks just like before. It is also assumed that assumptions 2 and 3 hold. This raises a problem as it is desirable to formulate the problem similarly to 1.4.1. Furthermore, if several plots from different sensors are assigned to the same track, the correlation scores should be updated accordingly as these assignments are not independent. A way to work around this is by doing the following: whenever a plot is assigned to a track where at least one plot from a different sensor can also be assigned to, create a new track for this case and update the correlation scores accordingly. These tracks are named correlated tracks and will now be formally defined.

Definition 7. (Correlated Tracks) If two plots from different sensors z^1 and z^2 are possibly originating from the same target, a **correlated track** (CT) will be used. In the case that both plots should be assigned to the same track T, the plot that was detected earliest, i.e., that appears in the highest row of the matrix, (assume that this is z^1) will be assigned to T. Plot z^2 will then be assigned to the correlated track $CT := T + z^1$ which is equal to track T updated by plot z^1. If more than two plots are able to be originated from the same target, this process will be iterated where in each iteration the earliest detected plots (that are assigned to the same track) will create a new correlated track to which the next plot will be assigned.

Differentiating plots by time of detection is needed in this case in order for the Kalman Filter to work correctly when applying the MHT method.

Correlated tracks can be seen in the assignment table 2.2, which is an example of two plots from two different sensors and one track. The correlated tracks are denoted by the $CT_{i,j}$ where the indices represent the assignment on which this correlated track is based. In the example, $CT_{1,1}$ is based on the hypothesis that the plot from sensor 1 is assigned to track T_1. Combining the plot from sensor 2 with T_1 is only possible if the plot from sensor 1 is not assigned to it, otherwise it must be assigned to the correlated track $T_1 + z_1^1 \equiv CT_{1,1}$. Because of this it is still possible to assume that the sum of the rows and columns of the assignment matrix does not exceed 1. It is important to note that it is not necessary for the plots to be ordered chronologically. However, doing so keeps the correlation matrices structured and easier to interpret. This is why the correlation matrices in this report will always have the plots chronologically ordered.

	T_1	NT_1	NT_2	FA	FA	$CT_{1,1}$	$CT_{1,2}$
z_1^1	$c_{1,1}$	$c_{1,2}$	∞	$c_{1,4}$	∞	∞	∞
z_1^2	$c_{2,1}$	∞	$c_{2,3}$	∞	$c_{2,5}$	$c_{2,6}$	$c_{2,7}$

Table 2.2: An assignment table for two sensors with two plots.

A drawback of these correlated tracks is that the number of columns is much larger when compared to the Sinlge-Sensor Assigment Problem. Table 2.3 shows a tracking matrix for four plots, each detected by a different sensor, and two existing tracks. The matrix is too large to fit on the page so it is split up into three sub-matrices. The amount of correlated tracks that are needed in this case is 31. The reason for this is that correlated tracks often create new correlated tracks by the iterative process found in the definition of correlated tracks, which means that the amount of of correlated tracks needed will grow quickly when either the amount of plots or the amount of sensors increases. This effect is not too surprising though, as the amount of hypotheses is much higher when multiple sensors are used.

Because of this, the old formula for the correlation matrix that was given in Equation 1.4.2 no longer holds. It is therefore necessary to find a new formula. In this report, Multi-Sensor Assignment Problems will be in the following form: $(M_1, M_2, \ldots, M_S)$, where the M_i denote the amount of plots that sensor i detects and $i < j$ implies that sensor i detects its plots chronologically earlier than sensor j. This is a slight deviation from reality as in the real world a sensor will not always detect all of its plots before another sensor detects one as well. However, solving both problems is exactly the same but this model will make things easier to calculate with while

still giving good approximations. Another reason to define MSAPs like this is that it maximizes the total amount of hypotheses, which means that the results based on this definition can be seen as upper bounds for real life problems. In the remaining part of this report, an MSAP will be abbreviated to a number that represents the vector. For example, the number 1113 will represent the MSAP problem $(1, 1, 1, 3)$ which is a problem where there are four sensors in total, three of which detect one plot and the last one detects three plots.

When this form is assumed and the number of existing tracks is denoted by A_T, the number of rows and columns of the corresponding correlation matrix can be found with the following formula:

$$\begin{aligned} m &= \sum_{i=1}^{S} M_i \\ n &= A_T + 2m + \sum_{i=1}^{S-1} n_i, \end{aligned} \tag{2.2.1}$$

where m again denotes the number of rows, n the number of columns and $n_i = M_i((\sum_{k=1}^{i-1} n_k) + A_T + 1)$ with $n_0 := 0$.

This formula can be found in the following way. For each plot, a row must be created, which is why the number of rows can be found by simply summing the number of plots for each sensor. The number of columns is trickier though. Each existing track should get its own track and each plot produces 2 tracks (one false alarm and one new track). After this, all that remains is to count the amount of correlated tracks. To obtain this, it will be counted per sensor how many correlated tracks this sensor produces. During this process, sensors are ordered by the time that they detect their plots. The first sensor produces a correlated track for each existing track and one for the new track for each of its plots, the second sensor does so as well but it will produce additional correlated tracks for each correlated track that was created by the previous sensor for each of its plots. By iterating this process for each sensor (except for the last one, because this one produces no correlated tracks) the Formula 2.2.1 was found.

Checking these values for the example in Table 2.3 it is obtained that

$$m = 4$$
$$n_1 = 3$$
$$n_2 = 6$$
$$n_3 = 12$$
$$n = 2 + 8 + 3 + 6 + 12 = 31.$$

This is the result that was expected and therefore it can be seen that the formula works for the example that was given. It can be seen that the amount of columns grows much quicker even with small number of rows. Because of this, it is assumed that n is much bigger than m in general.

	T_1	T_2	NT_1	NT_2	NT_3	NT_4	FA	FA	FA	FA	$CT_{1,1}$	$CT_{2,1}$
z_1^1	$c_{1,1}$	$c_{1,2}$	$c_{1,3}$	∞	∞	∞	$c_{1,7}$	∞	∞	∞	∞	∞
z_1^2	$c_{2,1}$	$c_{2,2}$	∞	$c_{2,4}$	∞	∞	∞	$c_{2,8}$	∞	∞	$c_{2,11}$	∞
z_1^3	$c_{3,1}$	$c_{3,2}$	∞	∞	$c_{3,5}$	∞	∞	∞	$c_{3,9}$	∞	$c_{3,11}$	$c_{3,12}$
z_1^4	$c_{4,1}$	$c_{4,2}$	∞	∞	∞	$c_{4,6}$	∞	∞	∞	$c_{4,10}$	$c_{4,11}$	$c_{4,12}$

	$CT_{3,1}$	$CT_{1,2}$	$CT_{2,2}$	$CT_{3,2}$	$CT_{2,11}$	$CT_{3,11}$	$CT_{3,12}$	$CT_{2,14}$	$CT_{3,14}$	$CT_{3,15}$
z_1^1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
z_1^2	∞	$c_{2,14}$	∞	∞	∞	∞	∞	∞	∞	∞
z_1^3	∞	$c_{3,14}$	$c_{3,15}$	∞	$c_{3,17}$	∞	∞	$c_{3,20}$	∞	∞
z_1^4	$c_{4,13}$	$c_{4,14}$	$c_{4,15}$	$c_{4,16}$	$c_{4,17}$	$c_{4,18}$	$c_{4,19}$	$c_{4,20}$	$c_{4,21}$	$c_{4,22}$

	$CT_{3,17}$	$CT_{3,20}$	$CT_{1,3}$	$CT_{2,4}$	$CT_{3,5}$	$CT_{2,25}$	$CT_{3,25}$	$CT_{3,26}$	$CT_{3,28}$
z_1^1	∞	∞	∞	∞	∞	∞	∞	∞	∞
z_1^2	∞	∞	$c_{2,25}$	∞	∞	∞	∞	∞	∞
z_1^3	∞	∞	$c_{3,25}$	$c_{3,26}$	∞	$c_{3,28}$	∞	∞	∞
z_1^4	$c_{4,23}$	$c_{4,24}$	$c_{4,25}$	$c_{4,26}$	$c_{4,27}$	$c_{4,28}$	$c_{4,29}$	$c_{4,30}$	$c_{4,31}$

Table 2.3: An assignment table for four sensors with each one plot.

2.3 Parent-child constraints

Now that correlated tracks are defined, it is possible to create correlation matrices for the Multi-Sensor Assignment Problem. However, it is still not possible to define the Multi-Sensor Assignment Problem just like the Single-Sensor Assignment Problem. This is because using the correlated tracks comes with additional constraints. Namely, whenever a plot is assigned to a correlated track, the corresponding (previous) track-plot combination must also be assigned. To clarify, if Table 2.3 is consid-

ered, it can be seen that plot z_1^2 can be assigned to correlated track $CT_{1,1}$. But it was explained earlier that $CT_{1,1} = T_1 + z_1^1$. It therefore only makes sense to assign z_1^2 to $CT_{1,1}$ if z_1^1 has also been assigned to T_1. This relation will be called a parent-child relation, where the assignment of z_1^1 to T_1 will be called the direct parent of the track $CT_{1,1}$.

This notation can get confusing, therefore it is slightly altered. Given a correlation matrix, a pair of indices (i, j) will be used to refer to the potential assignment of plot i to track j. This pair of indices will be called a potential assignment.

Often, chains of parent-child relations will exist when considering an assignment table. Consider again Table 2.3 and note that if plot z_1^4 were to be assigned to correlated track $CT_{3,28}$ (i.e. potential assignment $(4, 31)$) then z_1^3 would have to be assigned to correlated track $CT_{2,25}$. However, this last track is also a correlated track, and in order to make use of it z_1^2 would have to be assigned to NT_2. This last assignment is therefore also considered to be a parent of the potential pair $(4, 31)$ but it is not considered to be a direct parent. Each potential assignment can therefore have multiple parents, but only one direct parent.

To make it easier to refer to parents and childs of potential assignments, the following functions are created:

Definition 8. (The first order parent function) The first order parent function is the function $P^1 : A \to A$, where A is the space of sets of potential assignments. P^1 returns a set of the parent potential assignments $(\hat{i}, \hat{j})$ for each potential assignment (i, j) in its input. If an assignment does not have a parent, the function will return the empty set. Furthermore, $P^1(\emptyset) := \emptyset$.

Definition 9. (The first order child function) The first order child function is the function $\mathcal{C}^1 : A \to A$, where A is the space of sets of potential assignments. $\mathcal{C}^1$ returns a set of the child potential assignments (i, j) for each potential assignment $(\hat{i}, \hat{j})$ in its input. If an assignment does not have a child, the function will return the empty set. Furthermore, $\mathcal{C}^1(\emptyset) := \emptyset$.

These definitions can then be extended.

Definition 10. (The k^{th} order parent function) The k^{th} order parent function for $k \in \mathbb{N}$ is the function P^k which is equal to applying the first order parent function k times to a potential assignment.

Definition 11. (The k^{th} order child function) The k^{th} order child function for $k \in \mathbb{N}$ is the function $\mathcal{C}^k$ which is equal to applying the first order child function k times to a set of assignment indices.

Finally, the following functions are defined.

Definition 12. (The parent set function) The parent set function is a function P that can be defined in the following way: $P(i,j) := \cup_{n\in\mathbb{N}} P^n(i,j)$.

Definition 13. (The child set function) The child set function is a function $\mathcal{C}$ that can be defined in the following way: $\mathcal{C}(i,j) := \cup_{n\in\mathbb{N}} \mathcal{C}^n(i,j)$.

These functions can be used to formally define the Multi-Sensor Assingnment Problem:

Definition 14. The **Multi-Sensor Assignment Problem** is defined as the problem where the goal is to assign tracks to each plot such that the likelihood of this assignment is maximized. These likelihoods are found by applying the MHT method. It is assumed that Assumptions 1, 2 and 3 hold. Furthermore, each potential assignment can only be assigned if its direct parent is also assigned.

The corresponding optimization problem can be seen in Problem 2.3.1.

$$
\begin{aligned}
\underset{x_{ij}}{\text{minimize}} \quad & \sum_i \sum_j c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = 1 \quad (i = 1, \ldots, n), \\
& \sum_{i=1}^{n} x_{ij} \leq 1 \quad (j = 1, \ldots, m), \\
& x_{ij} \in \{0, 1\}, \\
& x_{i_\rho j_\rho} - x_{P^1(i_\rho j_\rho)} \leq 0, \quad \rho \in \{1, \ldots, N_{CT}\}.
\end{aligned}
\tag{2.3.1}
$$

In this optimization problem, N_{CT} is the number of correlated tracks and $(i_\rho j_\rho), \rho \in \{1, \ldots, N_{CT}\}$ denote the possible assignments that correspond to a correlated track. The row and column constraint of the matrix X are the same as in Problem 1.4.1 due to the way that correlated tracks are defined. The last constraint makes sure that a correlated track can only be assigned if its parent is also assigned.

This optimization problem is known to be NP-hard, this is because in [6] an NP-hard problem is described of which the SMAP is a more general version. Namely, in [6] the problem can be seen as an MSAP where all likelihoods are equal and the goal is simply to assign as much plots/tracks as possible.

2.4 The Kruger Branch and Bound method

Because of the additional constraint in Equation 2.3.1, the Munkres algorithm might return a solution that satisfies some of these constraints. This can be overcome by the introduction of a new method: the Branch and Bound method. This method was created in 1960 by Ailsa Land and Alison Doig [7]. The general idea of this Branch and Bound Algorithm will now be described. The terminology will be the same as in [9] to preserve contuinity later in this report.

The general idea of Branch and bound
The goal is to solve an optimization problem P, which is a problem for which an efficient solver is not known.

1. Solve a relaxed version of P that can be solved efficiently, denote this problem by P_0. The solution of P_0 will be denoted by S_0. P_0 is called root node.

2. Check whether S_0 is also a solution of P. If this is the case, the algorithm is finished (this scenario usually only happens very rarely).

3. If some variable x in S_0 does not satisfy the constraints in P, create 'nodes' (i.e. sub problems of P_0) that consist of P_0 with the additional constraint that x no longer violates that constraint. The variable x will then be called a branching variable. When branching, it is important that only infeasible solutions of P are removed from the solution space. Specifically, it should still be possible to reach any feasible solution of P in some node. More elaboration on this can be found in [7].

4. Now solve the problem P_i of one of these nodes, which will again be some relaxed version of problem P. If all constraints of P are satisfied by S_i, the node can be fathomed, the best feasible solution will be denoted as the incumbent. If there are no open branches left, the algorithm is finished. If not all constraints are satisfied, the node will produce new nodes such that again only infeasible solutions are removed.

5. If there are some open nodes left, repeat step four until all nodes are fathomed. Non-feasible nodes can also be fathomed when its solution value is worse than that of the incumbent. When all nodes are fathomed, the incumbent will be the optimal solution to P.

Each Branch and Bound method follows this structure. However, there are some aspects here that can be changed. These are:

- Choosing the relaxed problem P_0.

- Defining the way that the branching is defined.
- Defining the order in which nodes are processed.
- Applying heuristics.

A description of the Branch and Bound method that was implemented by Tanja Kruger [1] will now be described.

The relaxed problem
In the Kruger method, P_0 is created by removing the parent-child constraints. Due to this, the relaxed problem can be solved in polynomial time by applying the Munkres algorithm.

The branching method
As mentioned before, if applying Munkres to P_0 would give a feasible solution to the MSAP as well, the algorithm is finished. However, if a potential assignment (i, j) is assigned (i.e. $x_{i,j} = 1$) but its parent $P^1(i, j)$ is not (i.e. $x_{P^1(i,j)} = 0$) then somehow branches must be created to proceed the Branch and Bound method. Table 2.4 shows a truth table in which all possible combinations of the parent and child potential assignments can be seen and which ones are valid. It shows that the only infeasible situation occurs when the child is assigned but the parent is not. Therefore, when the parent constraint is not satisfied for some solution of a branch problem S_k, three branches will be created. Each of these branches will satisfy this particular constraint according to one of the feasible possibilities in table 2.4. This means that in each branch, potential assignments will be either forced or forbidden. The following strategy is therefore obtained:

Assume that $x_{ij} - x_{P^1(i,j)} > 0$ for some indices i and j in a branch P_k. Then three branches are created in the following order:

1. A branch where $x_{ij} = x_{P^1(i,j)} = 1$, also referred to as the 11-branch.
2. A branch where $x_{ij} = 0$ and $x_{P^1(i,j)} = 1$, also referred to as the 01-branch.
3. A branch where $x_{ij} = x_{P^1(i,j)} = 0$, also referred to as the 00-branch.

Note that the terms ‘forced’ and ‘forbidden’ are used now as opposed to assigned and not assigned. This is because there needs to be a differentiation between assignments that are made during branching and assignments that are made during the Munkres algorithm. An example of this branching strategy is seen in Figure 2.1. This figure will further in this section be explained in more detail.

During this process, some nodes may create an infeasible problem (i.e. there is no feasible solution possible). The Branch and Bound algorithm can be made more

efficient by preventing these branches from occurring as much as possible. This is why the following algorithm is used:

Whenever a certain potential assignment (i, j) is forced, add the indices to a set A_b. If the potential assignment is forbidden instead, add the indices to a set F_b. Whenever a new branch b is created, A_b and F_b are copied from the current branch. Assume that $x_{ij} - x_{P^1(i,j)} > 0$ for some potential assignment (i, j) in a branch b. Then the following is done:

1. Check whether there exists a potential assignment $(k, l) \in P(i, j)$ where $(k, l) \in F_b$ or $(k, \hat{l}) \in A_b$ or $(\hat{k}, l) \in A_b$ for $\hat{k} \neq k$ and $\hat{l} \neq l$. If this is the case, then $x_{i,j}$ can not be 1 so it will be set to 0 together with $x_{P^1_{(i,j)}}$.
2. If case 1 did not happen, check whether there exists a potential assignment $(k, l) \in \mathcal{C}(i, j)$ where $(k, l) \in A_b$. In this case, $x_{i,j}$ can not be 0 and therefore it will be set to 1 together with $x_{C^1_{(i,j)}}$.
3. If neither of those are the case, the branching will be done in three ways as described before.

x_{ij}	$x_{P^1(i,j)}$	valid
1	1	yes
0	1	yes
0	0	yes
1	0	no

Table 2.4: A truth table for all possible combinations of a child assignment and its parent assignment.

Defining the order in which branches are processed

In the Kruger method, branches are processed according to the Last In First Out (LIFO) principle. The idea behind this principle is that some (hopefully good) feasible solution will quickly be found, after which nodes can be fathomed much quicker as there is now a lower bound.

This Branch and Bound method can now replace Munkres in the single sensor case to create a solution to the multiple sensor assignment problem. Note that Munkres is still used in the Branch and Bound though.

An example

An example where the Kruger method is applied can be found in Figure 2.1. In this figure, the squares represent nodes in the branch and bound tree and the lines represent the constraints on which is branched. When a line has 11 directly above

it, this denotes the 11-branch which leads to the node in which this constraint has been added to the problem. The lines with 00 or 01 above them denote the same for the corresponding branches. In each node, the negative value of the relaxed problem is denoted by Z (which means that higher is better in this case) and the assignment is denoted by X, where $X_i = y$ implies that plot i has been assigned to target y (this is not equal to the matrix X as defined in the rest of this report, it uses a shortened notation for simplicity). Furthermore, in the bottom of each square is either explained why a node is fathomed or on which constraints it branches to new nodes. The corresponding assignment table for this example can be seen in Table 2.5.

It can be seen that the algorithm finds the optimal solution after eleven branches, and the algorithm completely terminates after twelve branches.

	T_1	$CT_{1,1}$	$CT_{2,1}$	$CT_{3,1}$	$CT_{2,2}$	$CT_{3,2}$	$CT_{3,3}$	$CT_{3,5}$
z_1^1	-0.9743	∞	∞	∞	∞	∞	∞	∞
z_1^2	-0.1032	-0.9526	∞	∞	∞	∞	∞	∞
z_1^3	-0.6112	-0.8132	-0.9035	∞	-0.2155	∞	∞	∞
z_1^4	-0.0350	-0.2716	-0.0060	-0.5385	-0.7794	-0.5245	-0.8542	-0.7845

	$CT_{1,16}$	$CT_{2,17}$	$CT_{3,18}$	$CT_{2,9}$	$CT_{3,9}$	$CT_{3,10}$	$CT_{3,12}$	NT_1
z_1^1	∞	∞	∞	∞	∞	∞	∞	-0.1739
z_1^2	-0.2194	∞	∞	∞	∞	∞	∞	∞
z_1^3	-0.2519	-0.1225	∞	-0.6074	∞	∞	∞	∞
z_1^4	-0.5174	-0.0513	-0.6309	-0.0771	-0.4572	-0.4971	-0.6214	∞

	NT_2	NT_3	NT_4	FA	FA	FA	FA
z_1^1	∞	∞	∞	-0.2473	∞	∞	∞
z_1^2	-0.0561	∞	∞	∞	-0.7663	∞	∞
z_1^3	∞	-0.1089	∞	∞	∞	-0.0041	∞
z_1^4	∞	∞	-0.6655	∞	∞	∞	-0.5415

Table 2.5: An assignment table for four sensors with each one plot and one existing track as (slightly adjusted) found in [1]. This is the problem that corresponds to Figure 2.1

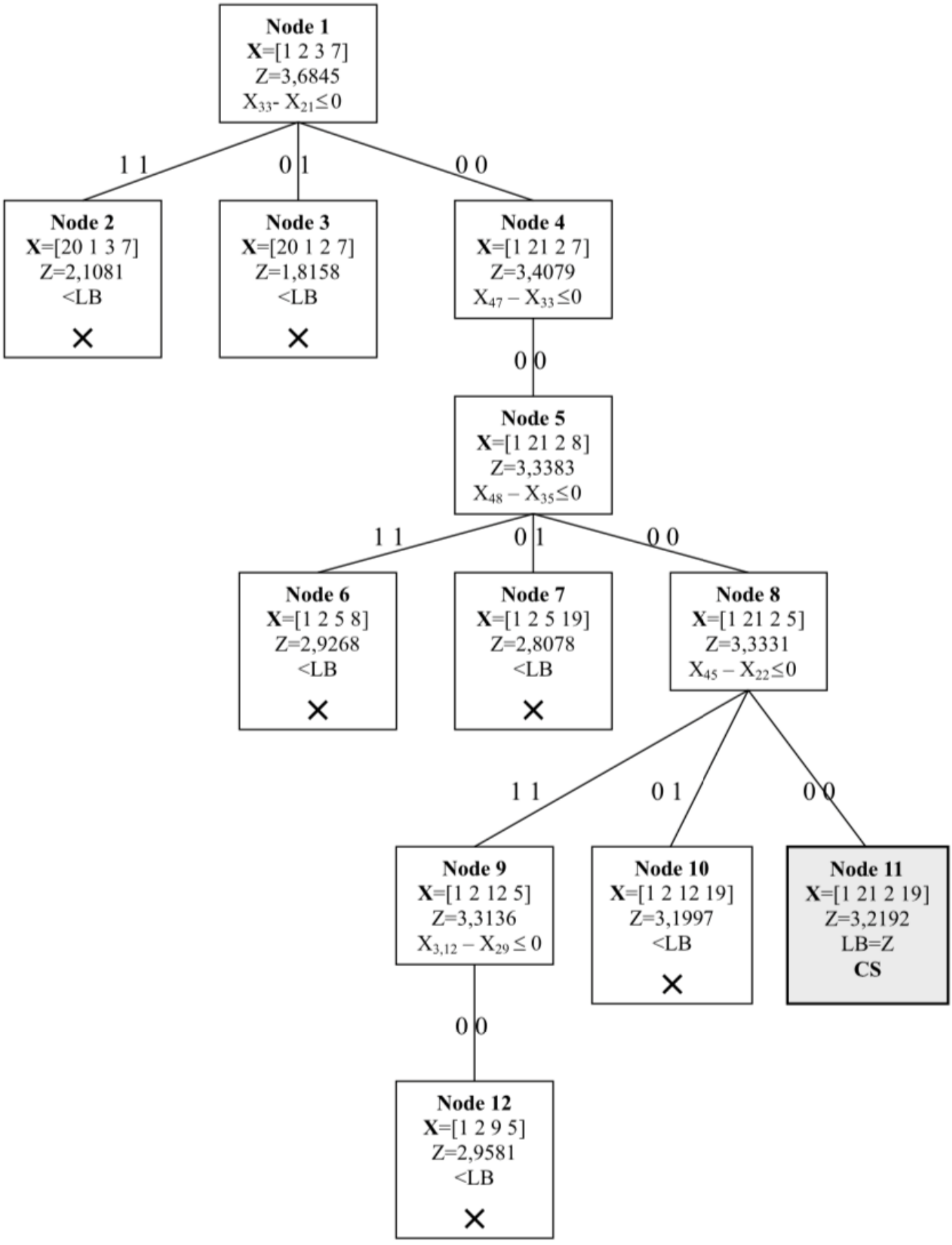

Figure 2.1: An example for the Kruger method as in [1]

2.5 The research problem

Now that the the Multi-Sensor Assignment has been defined and the Kruger method is explained, the research question can be formulated: the research problem is to investigate whether the Multi-Sensor Assignment Problems can be solved more efficiently than the Kruger method is currently doing.

In this case, more efficiently refers to decreasing the amount of computations necessary as much as possible.

Chapter 3

Improving the Branch and Bound algorithm

This section will be about reducing the complexity of the solving algorithm by changing the variables of the current Branch and Bound method. This is done because the current method seems to contain some straightforward possible improvements which will be explained in this section. Furthermore, in [1] it does not seem like several Branch and Bound strategies were tested and therefore it might be worthwhile to test some completely different ones.

As described before, there are four aspects when applying Branch and Bound. It is now investigated whether the solution algorithm can be improved by changing three of those. Namely, by adjusting the way branching is defined, adjusting the order in which nodes are processed and applying heuristics. Section 4 will investigate a solution in which the relaxed problem is altered as well. But in this section, the relaxed problem will stay the same as in the Kruger method.

It will now be explained what Branch and Bound strategies will be tested and how they are different from the Kruger method.

3.1 Adjusting the way each branch is created

Whenever a potential assignment (i, j) is forced, the potential assignments in the set $P(i, j)$ are also forced indirectly as otherwise it can only lead to more infeasible solutions. In order to save time later on, this could be done immediately.

Similarly, whenever a potential assignment (i, j) is forbidden, each of the potential assignments in the set $\mathcal{C}(i, j)$ are forbidden as well indirectly.

Additionally, some logical flaw has been found in the Kruger method that can still produce infeasible problems occasionally. Consider a node which does not contain a feasible solution. According to the Branch and Bound algorithm, branches must now be created. However, in the case that all three branches would lead to an infeasible node, then the Kruger method would still produce one infeasible node in its first branching step. This can be fixed by checking the feasibility of both branches individually before creating the corresponding nodes.

Furthermore, in the old method, the 01- and 00-branches can be replaced by a single 0 branch (as the parent can only be 0 or 1 anyway) and therefore only the child needs to be forced. It is efficient to create as few branches as possible as this reduces the amount of times one has to apply the Munkres algorithm.

The new branching algorithm will therefore be as follows:

Assume that $x_{ij} - x_{P^1(i,j)} > 0$ for some potential assignment (i, j) in a branch b. Then the following is done

1. If there is no (k, l) in $A_b \cap \mathcal{C}(i, j)$, create a branch where $x_{i,j} = 0$ and $x_{k,l} = 0$ for (k, l) in $\mathcal{C}(i, j)$.
2. If there is no (k, l) in $F_b \cap P(i, j)$, create a branch where $x_{i,} = 1$ and $x_{k,l} = 1$ for (k, l) in $P(i, j)$.
3. Proceed with the general Branch and Bound algorithm.

This new method therefore creates at most two branches for each node.

3.2 The order of processing branches

If the situation is considered where two branches have just been created according to the algorithm in the previous section, then the algorithm must proceed with a new branch. It is not yet clear which branch would be best to continue on.

Figure 3.1 shows an example with several nodes of a branch and bound tree. The optimal solution in this case has been given a yellow color. Blue nodes are nodes that have to be passed in order to obtain the optimal solution and white nodes represent nodes that do not have to be passed. The arrows have either a 0 or a 1 in them, where a 0 means that that arrow represents the branch that forbids the potential assignment(s) and the 1 means that that arrow represents the branch that forces the potential assignment(s). Each has a combination of ones and zeros in it, which represents the branching path that must be taken to reach that node. It can be seen that in this case, it is optimal to choose the 0-path first, then the 1-

path twice and then the 0-path once again. The optimal solution can therefore be reached quite quickly in this case. However, this path can only be known once the optimal solution has already been found, else there is no way to know where it is located. Because of this, a branching strategy needs to be used. In order to choose a branching strategy, several strategies will be tested. These different strategies will now be described.

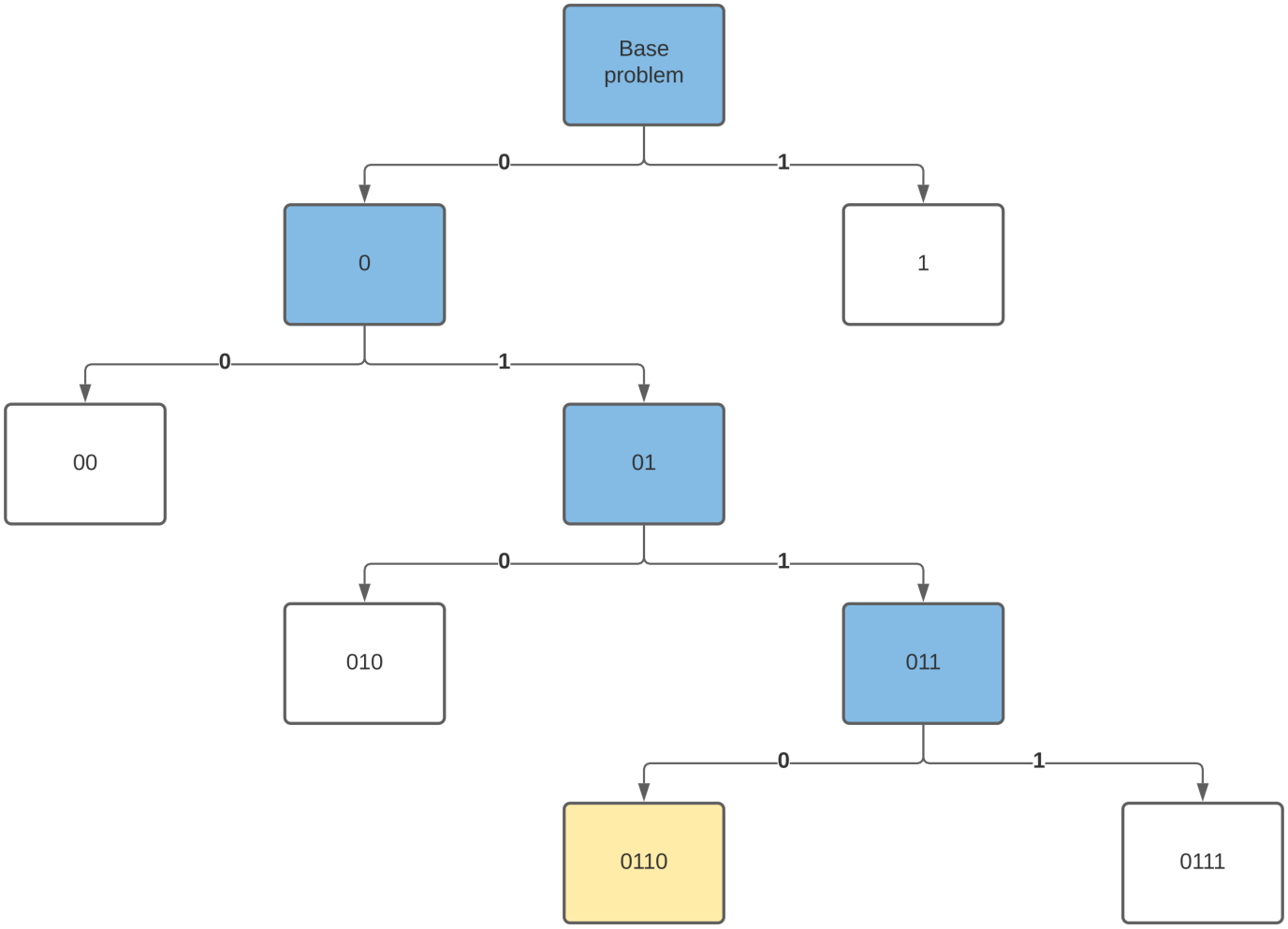

Figure 3.1: An example of the new branching strategy. In this figure, the location of the optimal solution is shown.

3.2.1 The 1-strategy

The 1-strategy is quite straightforward: it uses the Last In First Out (LIFO) principle just like the Kruger strategy. However, if two nodes are created simultaneously, it

will always choose the 1-path. An example of this strategy applied can be seen in Figure 3.2. In this case, each square represents a node and each arrow represents a branch just like in Figure 3.1. However, in Figure 3.2, the nodes are numbered according to the order they are fathomed in. Blue nodes represent nodes that did not give a feasible solution and could not be fathomed. Red nodes represent nodes that could be fathomed without being the incumbent. Green nodes represent nodes that did become the incumbent and the yellow node represents the optimal solution to the problem. It can be seen that it now takes 14 nodes to find the optimal solution and 16 nodes for the algorithm to be finished.

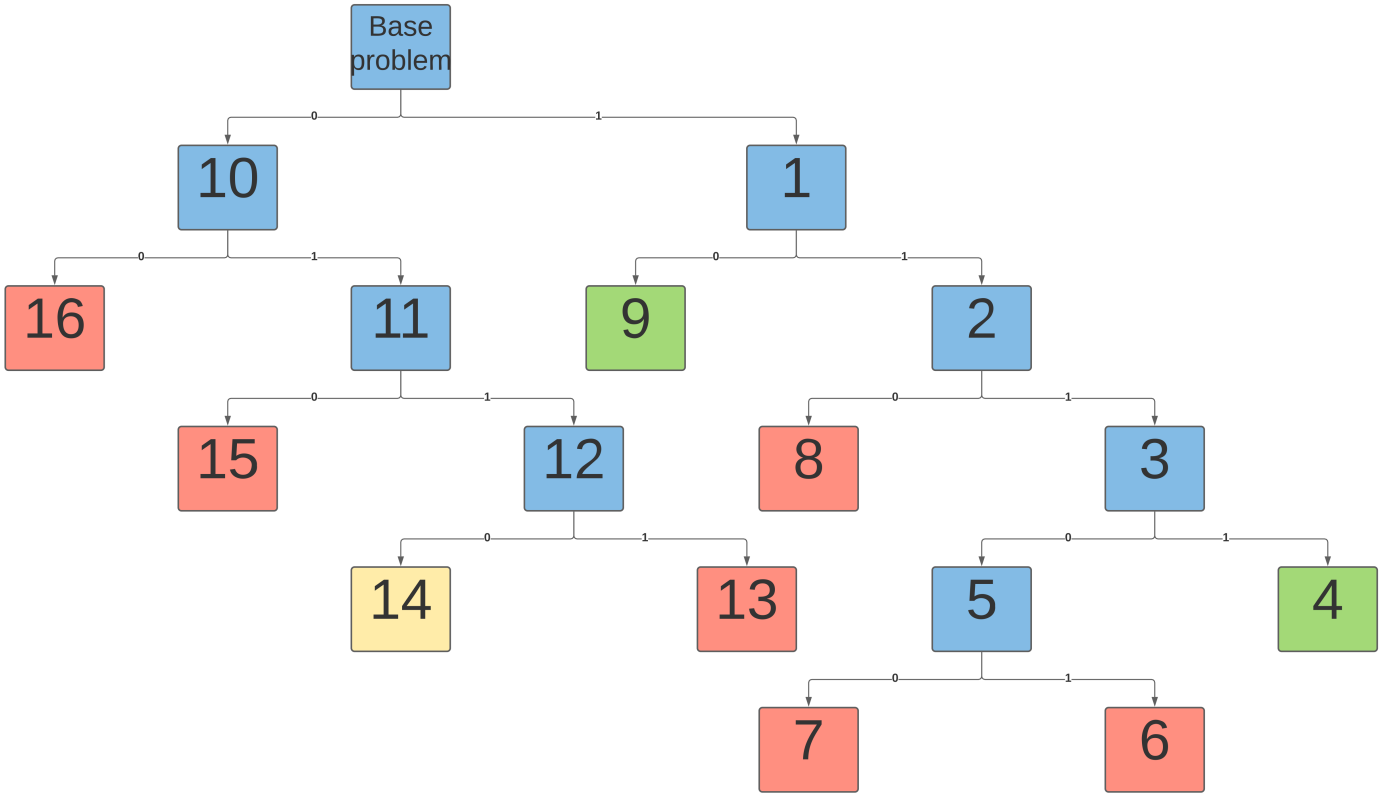

Figure 3.2: An example for the new branch and bound 1-strategy

3.2.2 The 0-strategy

The 0-strategy is similar to the 1-strategy. It also consideres nodes according to the LIFO principle. However, when two nodes are created simultaneously, it will always choose the 0-path.

3.2.3 The 05-strategy and 05R-strategy

The 05-strategy and 05R-strategy also consider nodes according to the LIFO principle. When two nodes are created simultaneously, the 05-strategy will switch between choosing the 1-path and the 0-path. It will do so by starting with the 1-path if possible. After this, if one path is chosen the other path will be chosen next time if possible. This will repeat until the algorithm is finished.

The 05R-strategy is very similar to the 05-strategy. It will also switch between both paths except it will do so randomly.

3.2.4 The Ordered-strategy

So far, the strategies have been using the LIFO principle. This raises the question whether it is also possible to use more advanced strategies. One of these will now be explained.

The ordered-strategy will make sure that for all open nodes the relaxed problem has been solved before branching. It will then proceed branching with the node that has the best solution. If this node is feasible, an optimal solution has been found and the algorithm can terminate. This is because all other nodes have an upperbound that is lower than this solution per definition. If the node is not feasible, the branches will be created like normal and the algorithm will repeate the same process.

3.2.5 Finding an upper bound of the solution value

A drawback of the Branch and Bound algorithm is that, before finding the first feasible solution, a node will almost never be fathomed. A solution for this could be to use some heuristic before applying Branch and Bound to find a feasible solution. This feasible solution should be as good as possible, as the closer it is to the optimal solution, the higher the probability that nodes can be fathomed.

It is chosen to use a greedy algorithm as this heuristic. Given an MSAP, this algorithm will choose the highest value $c_{\hat{i},\hat{j}}$ in the correlation matrix C. It will then proceed to assign the potential assignment that corresponds to this likelihood, i.e. $x_{\hat{i},\hat{j}} = 1$. After this, it will assign all parents of this potential assignment as well, i.e. $x_{i,j} = 1$ for $(i,j) \in P(\hat{i},\hat{j})$. Whenever a potential assignment is assigned, the row and column that correspond to this assignment will be removed from the correlation matrix C. After this, the algorithm will repeat until C is empty.

This greedy algorithm can be applied to the 1-, 0-, 05- and 05R-strategies, it will not be applied to the ordered-strategy as this strategy will only fathom a node if it is the

optimal solution and therefore adding this heuristic will not make a difference in this case. Four new strategies are therefore obtained. These will be denoted by 1G, 0G, 05G and 05RG, where the G stands for greedy and the numbers before it refer to the branching strategy.

3.3 Comparing to the Kruger method

To compare these new MSAP solving methods to the Kruger method, a performance benchmark needs to be created. Choosing a measure like running time would depend highly on the implementation quality of these methods. Therefore, instead of computation time, a computation score is created as a performance benchmark. This computation score is calculated in the following way: each time that Munkres is used to solve the assignment in a node, the worst complexity of this problem is equal to n^2m, where n is equal to the amount of rows and m is equal to the amount of columns. The sum of these worst complexities over all nodes will be the computation score of a method for that problem.

This computation score can therefore be seen as the 'total worst complexity'. It gives an indication of how many computations are at most necessary for a specific MSAP to be solved by a solving method. The computation score therefore depends on both the MSAP and the solving method. A lower computation score implies a more efficient solving method for that MSAP.

Next it is also important to address how the size of an MSAP is defined. MSAP's can grow in two ways: namely by increasing the number of plots and by increasing the number of sensors. To see how well each method performs when the problem size increases in either of those ways, two tables are created. Each of these tables start with a simple base case, namely three sensors that each detect one plot, after which one table shows the results after increasing the number of plots and the other table shows the results after increasing the number of sensors.

To fill these tables, 1000 randomly generated MSAPs are generated for each problem structure. Each strategy will then be test on these same problems. When generating an MSAP, the values in the correlation matrix that correspond to false alarms will be constant. The values that correspond to new tracks will be constant as well, but this constant will be different. Furthermore, the remaining values will be filled with either infinity when the assignment is not valid, or random numbers that have a uniform distribution.

As the computation score grows exponentially when increasing the size of the MSAP, the relative computation score with respect to the Kruger method will be considered

instead. This relative computation score is equal to the average computation score of the new method, divided by the average compuation score of the Kruger method. The results of comparing the Kruger and the new MSAP solving methods can be seen in Tables 3.1 and 3.2. In these Tables, the problem structure is the same as defined in Section 2.2. This means that problem structure 11112 refers to an MSAP where there are five sensors of which four detect only a single plot and the remaining sensor detects two plots.

Problem	1	1G	0	0G	05	05G	05R	05RG	Ordered
111	0.849	0.845	0.844	0.808	0.827	0.802	0.823	0.810	1.288
1111	0.659	0.645	0.698	0.669	0.671	0.647	0.664	0.641	1.053
11112	0.470	0.462	0.497	0.486	0.482	0.472	0.482	0.470	0.736
111122	0.276	0.273	0.335	0.327	0.292	0.287	0.292	0.292	0.418
1111222	0.160	0.159	0.239	0.235	0.176	0.176	0.178	0.176	0.224

Table 3.1: Comparing the old method to the new methods when the number of sensors increases. The values in this table are equal to the relative average computation score with respect to the Kruger method.

Problem	1	1G	0	0G	05	05G	05R	05RG	Ordered
111	0.849	0.845	0.844	0.808	0.827	0.802	0.823	0.810	1.288
222	0.619	0.602	0.641	0.623	0.645	0.624	0.633	0.612	0.909
333	0.548	0.541	0.563	0.555	0.565	0.555	0.560	0.548	0.627
444	0.535	0.533	0.578	0.572	0.556	0.553	0.553	0.553	0.479

Table 3.2: Comparing the old method to the new methods when the number of plots increases. The values in this table are equal to the relative average computation score with respect to the Kruger method.

It can be seen that different solving methods are best for different problem structures. The ordered-strategy performs poorly in the simple cases or when the number of sensors is increased. However, it suddenly performs best when the number of plots is increased. It should be noted though that except for the ordered-strategy the difference in performance per strategy is relatively small for each of the new methods. Overall, the 1-strategy is the best branching strategy in most categories as it has the lowest computation score for most problem structures and combining this with finding a greedy upper bound at the start slightly decreases the amount of computations even more. However, the effect of the greedy upper bound is not significant for any method as this only decreases the computation score slightly.

It is also interesting to see that the 05R-strategy performs slightly better than the 05-strategy in most cases.

Overall it can be stated that the computation score of the Kruger method can be significantly decreased by choosing another method and the 1G-strategy is the best candidate out of the ones that have been tested. If the 1G-strategy is chosen, the computational score decreases by around 15% for the smallest problems, around 80% for the problems with the most sensors and around 50% for the problems with the most plots. For the intermediate problems, it can be seen that the relative computational cost lies between these values and decreases when the problem size increases.

3.4 Attributes of the solving technique

It is also interesting to check how much the computation score differs per generated MSAP for each strategy and problem structure. If some strategy has a very inconsistent computation score, this could be a reason to not choose it as it will be inefficient in practice. Figure 3.3 shows a histogram of all computation scores for each strategy for the 444 problem structure. These histograms will have the (bucketed) amount of computations on the horizontal axis and the frequency of this amount on the vertical axis.

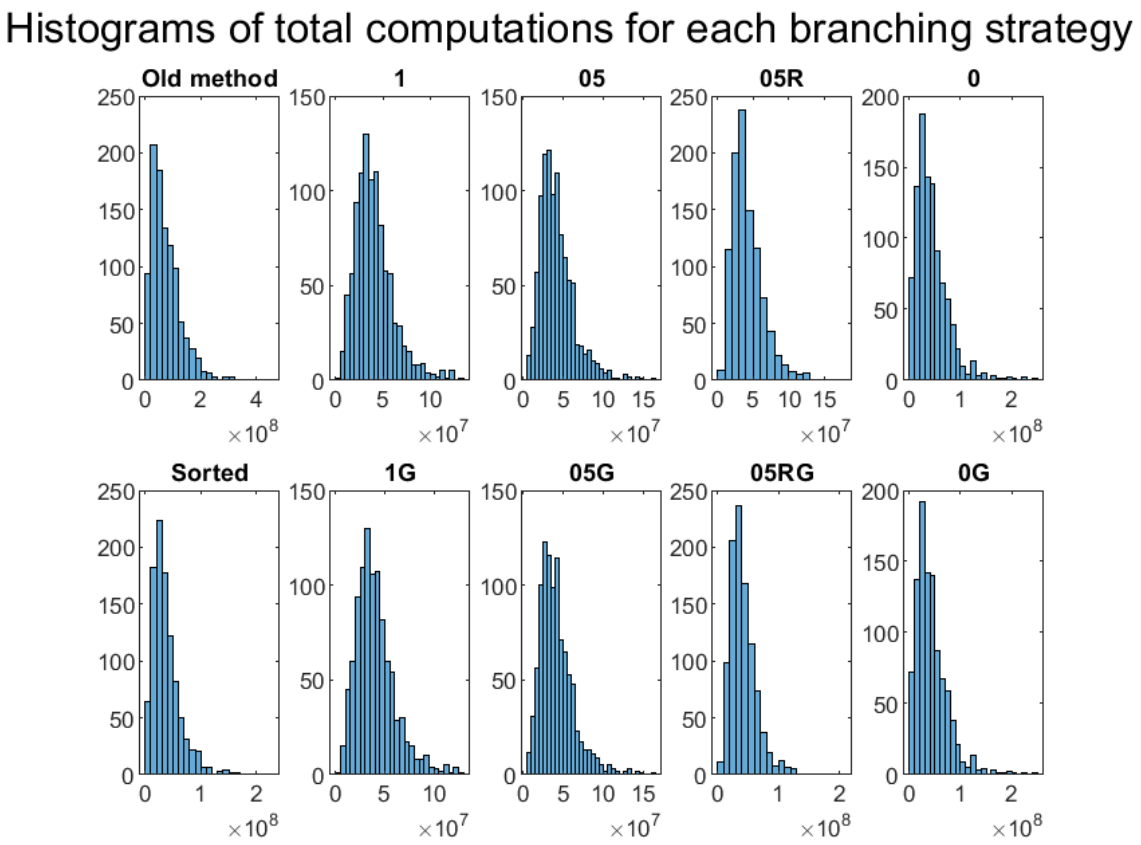

Figure 3.3: Histograms for the computation score of each strategy. The 444 problem structure is used in this case.

It can be seen from Figure 3.3 that the amount of computations are distributed around the mean where it has a peak in frequency. This implies that the computation score of a single run should be close to the data from Tables 3.1 and 3.2 and that the computation score should not vary too much in general.

These figures are similar for the other problems structures, figures of these can be found in the Appendix A. Therefore, it can be concluded that the previous statements hold for all MSAPs.

From now on, only the methods with the greedy heuristic are considered as its results are very similar to its non greedy counterpart and it has already been shown that adding this heuristic contributes to a better computation score for each method.

To give some insight into how each method solves the MSAP, Figure 3.4 shows an example of an MSAP with problem structure 222 where for each method the number of open nodes is plotted against the iterations. This figure has been turned 90 degrees as it would not fit onto the page otherwise. The horizontal axis (after rotation) shows the number of open nodes and the vertical axis shows the iterations.

The red line represents that a new incumbent is found during that iteration.

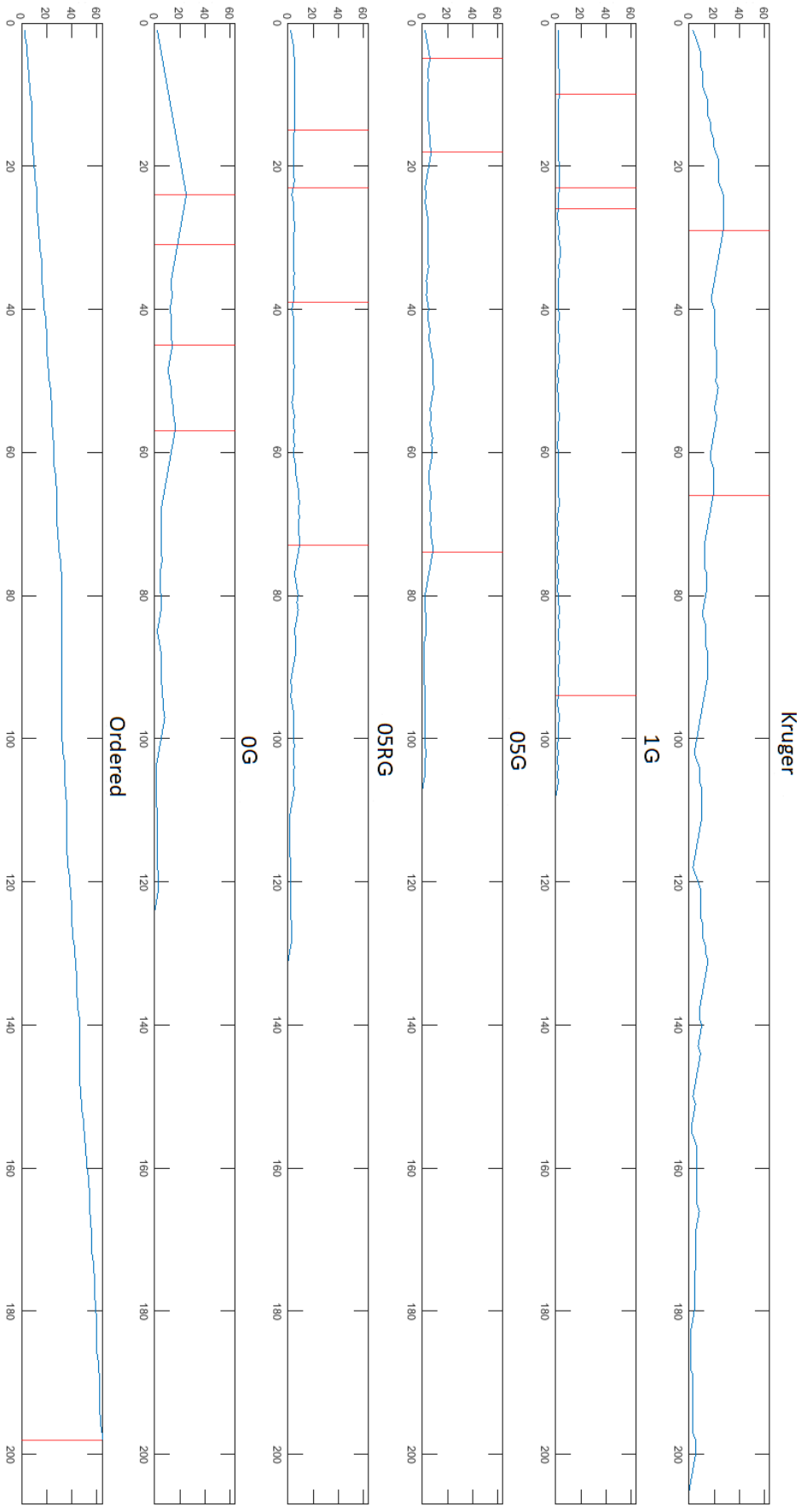

Figure 3.4: A Figure in which the branches against iterations are shown for each method, the problem structure in this case is 222.

It can be seen in Figure 3.4 that the ordered-strategy only finds one incumbent during its final iteration. This makes sense as the ordered method should only find a single feasible solution which should also be the optimal solution to the MSAP. Furthermore, it can be seen that the $1G$-strategy never has a lot of open nodes at the same time in comparison with the other strategies. This might be useful in practice as it means that this method uses less memory when implementing. This is something that was also noticable in this plot for other problem structures and for several other MSAPs with the same structure, which means that it is considered to be an attribute of the 1G-strategy. More of these figures are not included in this report as it would take up too much space.

Something else that is is considered is the average number of nodes that a strategy needs to find the optimal solution. This can be seen in Tables 3.3 and 3.4.

Problem	Kruger	1G	0G	05G	05RG	Ordered
111	9.3	5.8	5.9	5.8	5.9	10.3
1111	28.1	17.0	16.7	17.2	17.0	26.9
11112	265	144	140	145	145	199
111122	2148	858	1010	901	919	1120
1111222	15335	4290	6708	4801	4779	4980

Table 3.3: Comparing the average amount of nodes necessary when the number of sensors increases. For each problem structure, 1000 problems were created.

Problem	Kruger	1G	0G	05G	05RG	Ordered
111	9.3	5.8	5.9	5.8	5.9	10.3
222	112	79	70	80	78	98
333	1105	838	713	832	816	740
444	7826	6360	5795	6482	6447	4507

Table 3.4: Comparing the average amount of nodes necessary when the number of plots increases. For each problem structure, 1000 problems were created.

In Tables 3.3 and 3.4, it can be seen that the $1G$-strategy often requires less nodes than other strategies when the number of sensors increases. However, when the number of plots increases, the $1G$-strategy does not require the least nodes. In this case, the $0G$- or ordered-strategies require the least amount of nodes. Note that this is not necessarily contradicting Table 3.2 as in Table 3.2 the problem size in each node is also considered which influences the performance score.

It can be concluded that the $1G$-strategy does not always require the least amount of nodes, but it does have the lowest computation score for most problem structures. Furthermore, the $1G$-strategy is the most memory-efficient out of all the strategies.

Chapter 4

Solving the MSAP as a mixed integer program

So far the MSAP has been solved by first relaxing it to a regular assignment problem and then applying Branch and Bound to solve it for the additional constraints. However, this is not the only way to do this. Another method would be to leave out the binary constraints and solve it as a Linear Program (LP). The relaxed problem will then be in the form of 4.0.1, which can be solved efficiently by an LP solver like the Simplex method. After this, Branch and Bound can be applied to solve it for the binary constraints. When doing this, the MSAP can be seen as a mixed integer program.

$$\begin{aligned} \underset{x_{ij}}{\text{maximize}} \quad & \sum_i \sum_j c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = 1 \quad (i = 1, \ldots, n), \\ & \sum_{i=1}^{n} x_{ij} \leq 1 \quad (j = 1, \ldots, m), \\ & x_{i_\rho j_\rho} - x_{P^1(i_\rho j_\rho)} \leq 0, \quad \rho \in \{1, \ldots, C\}. \end{aligned} \tag{4.0.1}$$

4.1 Gurobi

Gurobi [8] is commercial software that is very efficient in solving these mixed integer programs. Gurobi not only uses an efficient Linear Program and Branch and Bound solver, but also makes use of various other techniques. The four main techniques are: presolving, cutting planes, heuristics and parallelism. Each of these will now shortly be explained, the information in these explanations is taken from the Gurobi

webpage [9].

4.1.1 Presolvers

Before applying the branch and bound method, presolvers can be used to tighten the constraints such that fewer computation are needed to find a solution. Presolvers are therefore classified as size-reducing transformations.

An example of this is the following. Consider the following set of inequalities:

$$\begin{aligned} &x_1 + x_2 + x_3 \geq 7, \\ &x_1 \leq 4, \\ &x_2 \leq 2, \\ &x_3 \leq 1. \end{aligned}$$

By making use of basic mathematical logic, these inequalities are only satisfied when $x_1 = 4, x_2 = 2$, and $x_3 = 1$. The equalties can therefore be removed and the variables can be given a fixed value. A presolver could therefore completely remove these variables from an optimization problem, which would reduce the size of this problem.

There are also size reductions specifically for MIPs. An example of such a reduction will now be given. Consider x_1 and x_2 and assume that both have to be binairy (i.e. they can only be one or zero). Next, consider the following inequality:

$4x_1 + 4x_2 \leq 1.$

Which can be reduced to the following form by dividing by four:

$x_1 + x_2 \leq \frac{1}{4}.$

As the sum of two binairy variables must always be integer, this inequality can again be reduced to:

$x_1 + x_2 \leq 0.$

Considering that x_1 and x_2 should be binairy and therefore can not take on a negative value, the only solution to this inequaltity is $x_1 = x_2 = 0$. This once again removes these variables from the problem and therefore the problem size is reduced.

4.1.2 Cutting planes

Gurobi also makes use of the concept of cutting planes when solving a MIP. According to their webpage, the theory of cutting planes is 'deep and extensive' but also 'generally accepted to be the single most important contributor to the computational advances that have been made in integer programming over the last several years'.

The goal of cutting planes is also to make inequalities more tight such that the solution space of the relaxed problem will include less feasible solutions, which is similar to presolving. However, cutting planes differ in the sense that they are applied during the branch and bound process and they do not create additional (undesirable) constraints.

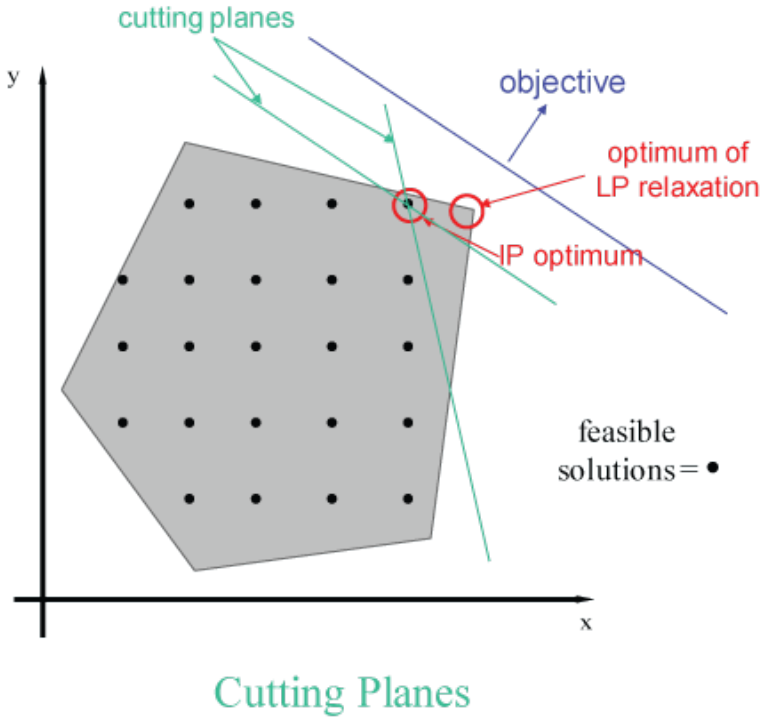

Figure 4.1: A schematic representation of a cutting plane as in [9].

A simple example of a cutting plane from [9] will now be given. Suppose that the following inequality is part of a MIP:

$$6x_1 + 5x_2 + 7x_3 + 4x_4 + 5x_5 \leq 15,$$

where x_i, for $i = 1 \ldots 5$ are required to be binary. Furthermore, suppose that the following solution of the relaxed version of this optimization problem has been obtained:

$$x_1 = 0,$$
$$x_2 = 1,$$
$$x_3 = x_4 = x_5 = \frac{3}{4}.$$

Normally, a new node would have to be created as this solution is not binairy. However, this node can be removed by applying a cutting plane. First, observe that $7+4+5 = 16 > 15$ which implies that $x_3 = x_4 = x_5 = 1$ is not allowed. Because of this, a new constraint can be added to the optimization problem: $x_3+x_4+x_5 \leq 2$. This will now eliminate this node from the branch and bround algorithm as $\frac{3}{4} + \frac{3}{4} + \frac{3}{4} = \frac{9}{4} > 2$ and therefore it is no longer a solution to the relaxed problem.

One might wonder at this point why this new constraint was not added at the beginning. Is it not possible to find all cutting planes and simply add them before branching? There are two main reasons to not do this. One is that there often are a lot of cutting planes possible and adding them all would take too much time and therefore make the algorithm less efficient. The second reason is that additional constraints can also make an LP harder to solve. New constraints must only be added if it is known that it will remove infeasible nodes, else it is just add computations and creates a new constraint that then also must be considered by the LP-solver (which also requires more computations).

4.1.3 Heuristics

An example of a heuristic has already been mentioned in Section 3.2.5. In this case, it was tried to find a good incumbent as quick as possible such that nodes could be fathomed faster. Gurobi also applies such heuristics. However, because the software is not open source, it is unknown what these heuristics look like and how they differ from the greedy method.

According to the Gurobi webpage for MIPs: 'Gurobi includes multiple such heuristics of many different flavors.' Which implies that multiple heuristics are used and that they are probably more advanced than just a single greedy algorithm before applying branch and bound. It is therefore recommended to investigate what more heuristics could be applied to improve the $1G$-strategy.

4.1.4 Parallelism

When Gurobi solves an MIP, it runs in parallel. This is done because the Branch and Bound nodes can partly be processed independently. When only the root node

is created there is no possiblity for nodes to be processed in parallel. However, as more and more branching occurs and open nodes are created, it is possible to process nodes independent of each other. It was shown in Section 3.4 that the $1G$-strategy often does not have a lot of open nodes at the same time when compared to other methods. It can therefore be decided to choose another strategy such that paralellism can be applied more efficiently.

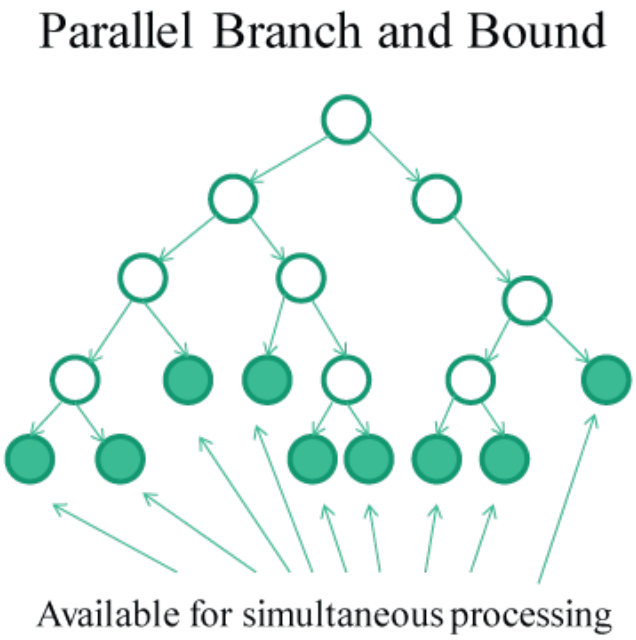

Figure 4.2: A schematic representation of parallel branching as in [9].

This idea of parallelism somewhat contradicts the previous measure of performance. Namely, valuing the method with the least amount of total computations as best. This is because parallelism does not reduce the total amount of computations but does in practice lead to saving time and making more efficient use of the tools available. However, this aspect is neglected as optimizing the capacity of CPU/GPU cores is beyond the scope of this research project. It is a recommendation to look into this when implementing an MSAP solver.

4.1.5 Additional methods

The Gurobi MIP-solver webpage also mentiones some other techniques that are not specified further. These are:'sophisticated branch variable selection techniques, node presolving, symmetry detection, and disjoint subtree detection'. These methods are used to limit the size of the branch and bound tree.

4.2 Comparing the mixed integer approach to the assignment approach

Gurobi is now used to solve the MSAP problem by applying it to the same problems as were used in Tables 3.1 and 3.2.

Comparing the results from Gurobi with the old method is not as straightforward as before. This is because the nodes for both approaches are not comparable as each node requires a different type of problem to be solved (assignment problem versus a linear program). Furthermore, the internal implementation may be different (Gurobi is programmed in Python and the current approach in MATLAB). It is therefore chosen to simply compare running times. The results of this can be found in Tables 4.1 and 4.2.

It must be noted that because of practical limitations, each method was performed by a different computer. Which means that, even if the implementation were similar, comparing running times will be unfair. However, as can be seen in Tables 4.1 and 4.2, the running times differ quite significantly. These differences seem too high to be caused by any of the unfair circumstances. Because of this it can be concluded that Gurobi performs better for large problems, but worse on very small problems.

Problem structure	Kruger	1G	05RG	Ordered	Gurobi
111	1.16	0.755	0.770	1.09	2.22
222	43.55	28.9	29.6	41.4	5.42
333	$1.51 \cdot 10^3$	966	988	$1.07 \cdot 10^3$	15.9
444	$4.29 \cdot 10^4$	$2.64 \cdot 10^4$	$2.75 \cdot 10^4$	$2.48 \cdot 10^4$	29.5

Table 4.1: This table shows the time in seconds it takes each method to solve 1000 MSAPs with the given structure.

Problem structure	Kruger	1G	05RG	Ordered	Gurobi
111	1.16	0.755	0.770	1.09	2.22
1111	4.48	2.79	2.81	4.30	3.31
11112	89.2	41.7	42.3	64.7	9.68
111122	$3.17 \cdot 10^3$	737	774	$1.11 \cdot 10^3$	37.6
1111222	$2.50 \cdot 10^5$	$2.35 \cdot 10^4$	$2.46 \cdot 10^4$	$3.16 \cdot 10^4$	261

Table 4.2: This table shows the time in seconds it takes each method to solve 1000 MSAPs with the given structure.

As mentioned before, the time it takes for Gurobi to finish grows much slower than

the other methods. It is possible to solve an MSAP with structure 444444 with Gurobi within two minutes where it would probably take months for the other methods to do so (it did not succeed to do so within several weeks). Because Gurobi is not open source, further research is needed to know for sure why this is but the global strategy of Gurobi is known and therefore some theoretical comparisons can be made.

When comparing both approaches, the assignment approach will always have less constraints on which need to be branched. This is because the MIP approach will have a binary constraints for each potential assignment, which leads to mn binairy constraints. The assignment approach only has a parent-child constraint for each potential assignment that is part of a correlated track. This number does come very close to mn for large matrices as then the fast majority of columns are correlated tracks. As having less constraints makes problems easier to solve when applying Branch and Bound, this is considered to be an advantage of the assignment approach.

It could be tried whether using the Gurobi methods (presolvers, cutting planes, heuristics, parallelism and the additional methods) can be used to improve the assignment approach. Some heuristics have already been tried (but there many more that could be used) and there is no straightforward method to apply presolvers and cutting planes to the assignment approach, but parallelism and the additional methods can definitely be applied. However, because of all the different methods that have been specifically developed for MIPs, it can also be the case that the MIP approach will always be better than the assignment approach even though is has more constraints. It is therefore a future recommendation to see whether the MIP approach can be refined specifically for MSAPs to make it even faster, perhaps also for small problems. It is also a future recommendation to see whether (some of) the Gurobi methods can be applied to the assignment approach to make it better than the MIP approach.

Chapter 5

Conclusion and recommendations

To briefly recap, the research question was whether the Multi-Sensor Assignment Problem could be solved more efficiently than the Kruger method was doing. Which means that the computational cost should be lowered.

The computational cost of solving a Multi-Sensor Assignment Problem can be significantly decreased by using any of the new strategies introduced in section 3. These new strategies improve the Kruger method by branching more efficiently and by eliminating infeasible nodes earlier. The 1G-strategy performs the best in general. However, Gurobi is able to solve large MSAPs much faster than the 1G-strategy because of the Mixed Integer Problem approach and the additional methods that are available for these types of problems.

A future recommendation is therefore to see whether the MIP can be adjusted specifically for MSAPs to get even better performance, specifically for the smaller problems. Another future recommendation is to see whether the assignment approach can be better than the MIP approach after applying the Gurobi methods to it.

Chapter 6

Symbols and abbreviations

Table 6.1 shows all abbreviations that were used in this report and the full names next to it. This Table can therefore be used to search a full name of an abbreviation as it might be difficult to find it back in the report.

Abbreviation	Full name
SSAP	Sinlge-Sensor Assignment Problem
MSAP	Multi-Sensor Assignment Problem
LP	Linear Program
MIP	Mixed Integer Program
MHT	Multiple Hypothesis Tracking
LIFO	Last In First Out

Table 6.1: A table that includes all abbreviations used in this report.

Table 6.2 shows the most important symbols that were used in this report and its definition or an explanation.

Symbol	Definition/explanation
$\mathcal{F}$	A state function.
$s_i(k)$	A state vector.
$z_k^{(i)}$	A plot.
T_i	A track.
$Z(k)$	A data set.
Z^l	The cumulative set of data sets.
M_k	The number of plots produced during scan k.
Ω^k	A set of hypotheses.
$t_d(i,k)$	Time of detection of target i during scan k.
$\hat{s}_i(k)$	The predicted state vector.
C	Correlation matrix
X	Assignment matrix
m	The number of rows in a matrix
n	The number of columns in a matrix
A_T	The number of tracks
NT	A new track
FA	A false alarm track
CT	A correlated track
P^1	The first order parent function
$\mathcal{C}^1$	The first order child funtion
P^k	The k^{th} order parent function
$\mathcal{C}^k$	The k^{th} order child funtion
P	The parent set function
$\mathcal{C}$	The child set function
N_{CT}	The number of correlated tracks
(i,j)	Potential assignment
F_b	Set of forbidden potential assignments
A_b	Set of forced potential assignments

Table 6.2: A table that includes all symbols used in this report.

Bibliography

[1] T. Kruger, “The multiple sensor multiple target data association problem with additional constraints,” Master’s thesis, University of Twente, Enschede, 06 2003.

[2] B. Sibma, “Hypotheses for the multiple sensor multiple target data association: k-best assignment problem with additional constraints,” Master’s thesis, University of Groningen, Groningen, 05 2004.

[3] D. Reid, “An algorithm for tracking multiple targets,” *IEEE Transactions on Automatic Control*, vol. 24, no. 6, pp. 843–854, 1979.

[4] L. Ramshaw and R. E. Tarjan, “A weight-scaling algorithm for min-cost imperfect matchings in bipartite graphs,” in *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, 2012, pp. 581–590.

[5] K. G. Murty, “Letter to the editor—an algorithm for ranking all the assignments in order of increasing cost,” *Operations Research*, vol. 16, no. 3, pp. 682–687, 1968. [Online]. Available: https://doi.org/10.1287/opre.16.3.682

[6] N. Kakimura and M. Takamatsu, “Matching problems with delta-matroid constraints,” *SIAM Journal on Discrete Mathematics*, vol. 28, no. 2, pp. 942–961, 2014.

[7] A. H. Land and A. G. Doig, “An automatic method of solving discrete programming problems,” *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960. [Online]. Available: http://www.jstor.org/stable/1910129

[8] https://www.gurobi.com, “The official gurobi website,” 08 2021.

[9] https://www.gurobi.com/resource/mip-basics, “The mip solving technique from the official gurobi website,” 08 2021.

[10] R. Danchick and G. Newnam, “Reformulating reid’s mht method with generalised murty k-best ranked linear assignment algorithm,” *Radar, Sonar and Navigation, IEEE Proceedings* -, vol. 153, pp. 13 – 22, 03 2006.

[11] S. Oh, “A scalable multi-target tracking algorithm for wireless sensor networks,” *International Journal of Distributed Sensor Networks*, vol. 8, no. 9, p. 938521, 2012. [Online]. Available: https://doi.org/10.1155/2012/938521

[12] S. Mori, C.-Y. Chong, and K. Chang, “Distributed multiple hypothesis tracking in finite point process formalism: A simple two station case,” in *2019 22th International Conference on Information Fusion (FUSION)*, Ottawa, Canada, 2019, pp. 1–8.

[13] R. Jonker and T. Volgenant, “A shortest augmenting path algorithm for dense and sparse linear assignment problems,” in *DGOR/NSOR*, H. Schellhaas, P. van Beek, H. Isermann, R. Schmidt, and M. Zijlstra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 622–622.

[14] C.-Y. Chong, S. Mori, and D. B. Reid, “Forty years of multiple hypothesis tracking - a review of key developments,” in *2018 21st International Conference on Information Fusion (FUSION)*, Cambridge, United Kingdom, 2018, pp. 452–459.

[15] S. Oh, S. Russell, and S. Sastry, “Markov chain monte carlo data association for multi-target tracking,” *IEEE Transactions on Automatic Control*, vol. 54, no. 3, pp. 481–497, 2009.

[16] G. Castñnón and L. Finn, “Multi-target tracklet stitching through network flows,” in *2011 Aerospace Conference*, Big Sky, Montana, USA, 2011, pp. 1–7.

[17] S. Mori and C.-Y. Chong, “Performance analysis of graph-based track stitching,” in *Proceedings of the 16th International Conference on Information Fusion*, Istanbul, Turkey, 2013, pp. 196–203.

Appendix A

Histograms of the computation scores

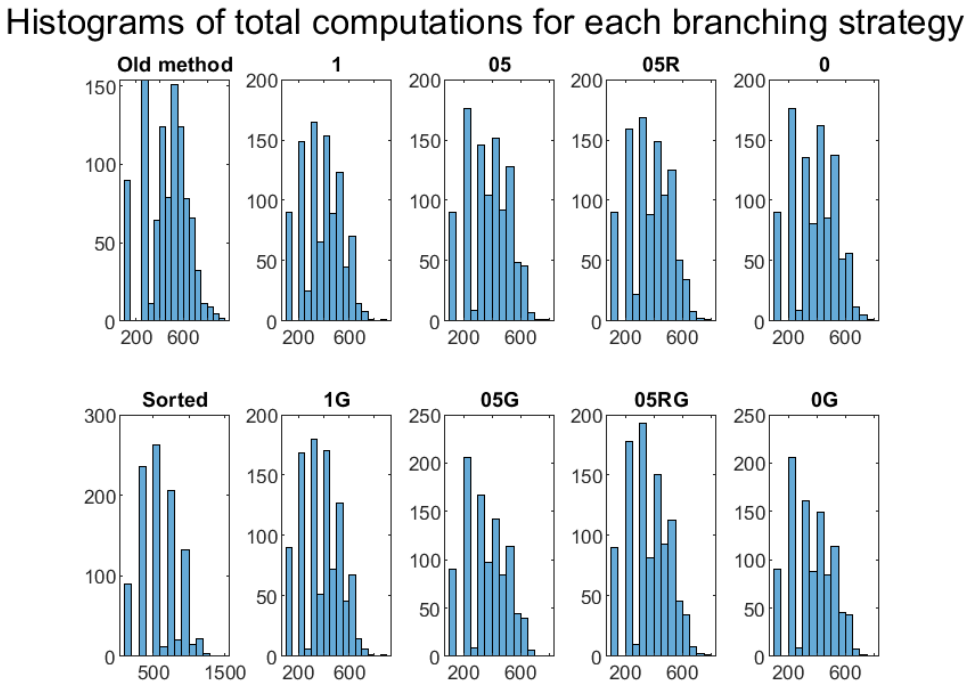

Figure A.1: Histograms for the computation score of each strategy. The 111 problem structure is used in this case.

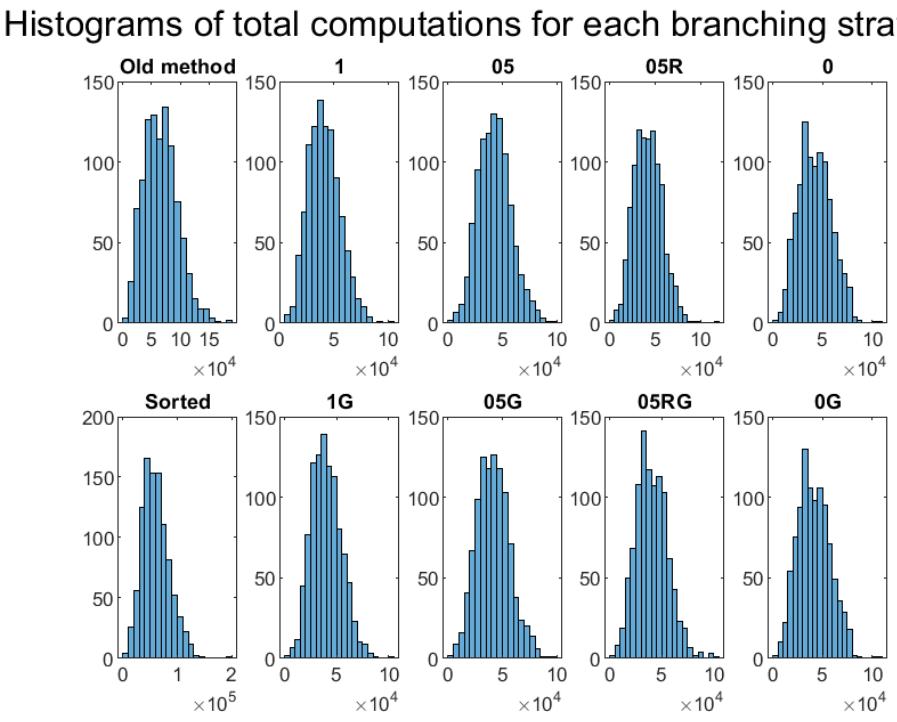

Figure A.2: Histograms for the computation score of each strategy. The 222 problem structure is used in this case.

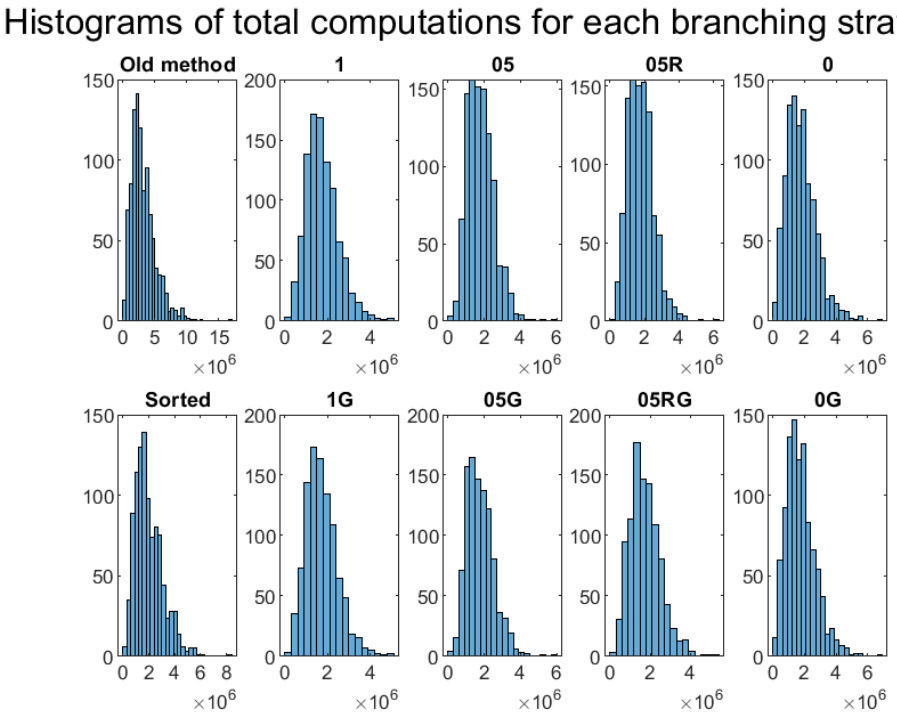

Figure A.3: Histograms for the computation score of each strategy. The 333 problem structure is used in this case.

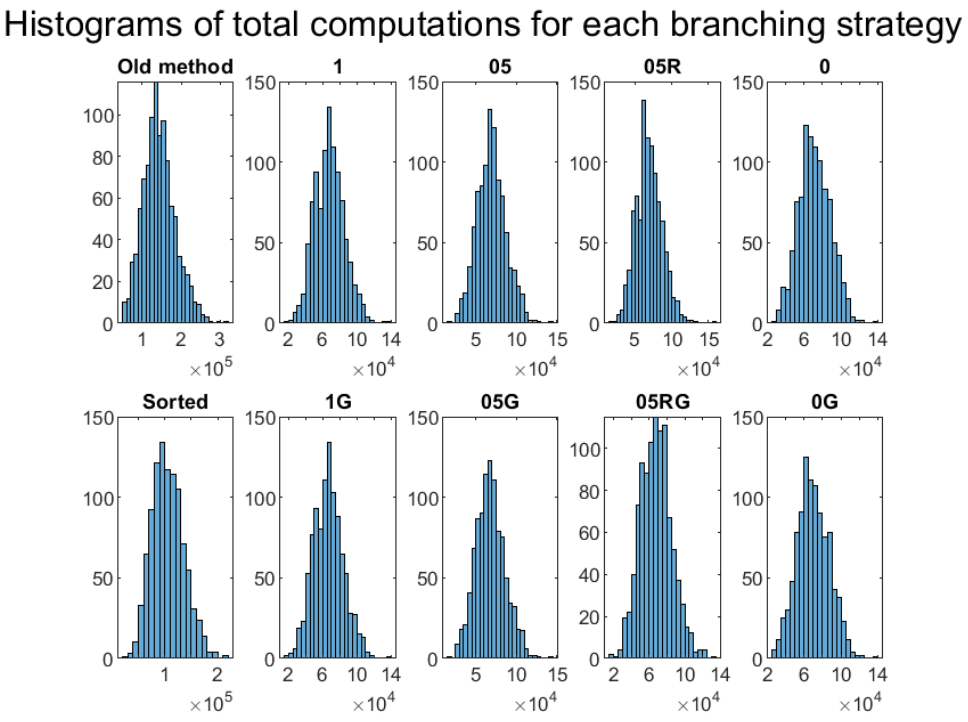

Figure A.4: Histograms for the computation score of each strategy. The 11112 problem structure is used in this case.

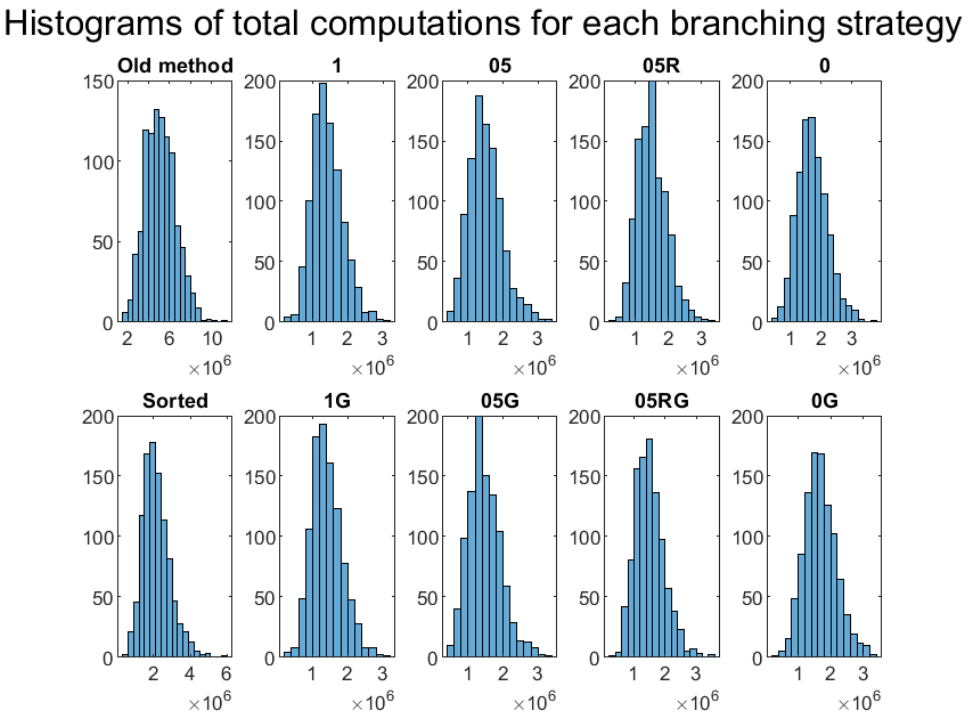

Figure A.5: Histograms for the computation score of each strategy. The 111122 problem structure is used in this case.

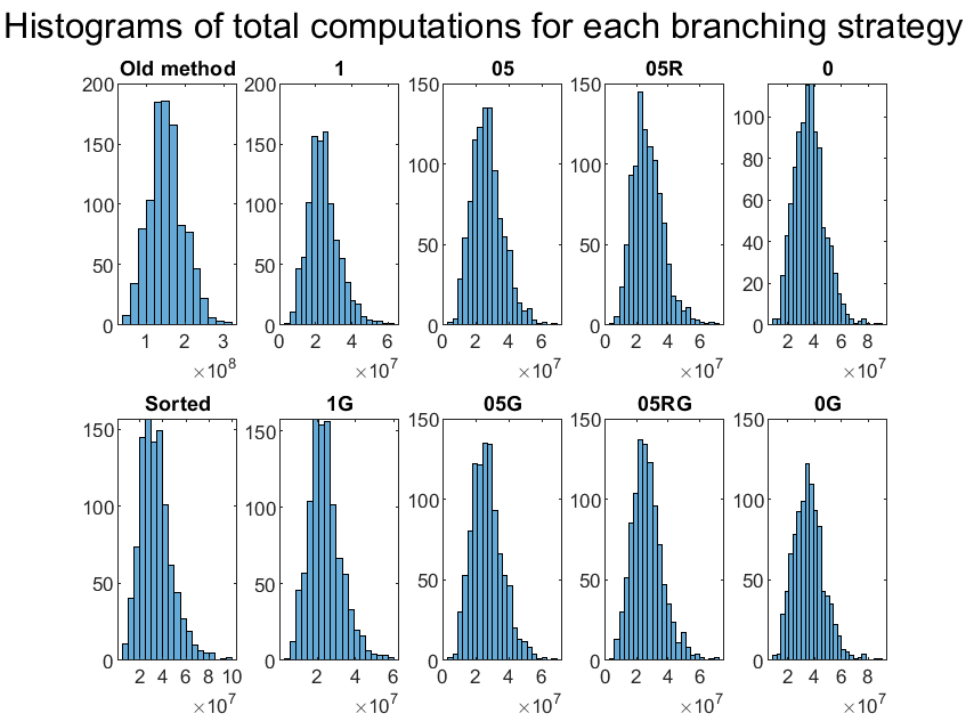

Figure A.6: Histograms for the computation score of each strategy. The 1111222 problem structure is used in this case.