UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Dipartimento di Ingegneria dell'Informazione
Laurea Magistrale in Computer Engineering

Matching of RFID car tags to train axles at MERMEC

Relatore: prof. Francesco Silvestri

Tutor aziendale: ing. Stefano Totaro

Laureando: Marco Gasparini

Anno accademico 2023-2024

Data di laurea 15/10/2024

Abstract

A crucial step in railway diagnostics is the capability of correctly matching data coming from different sensors. For example, in some measurement systems, cars are identified and detected using RFID tags, while the information about axles are collected by different tools installed on the rails. It is then essential to be able to determine which axles have to be associated to each car, and this is a non-trivial goal because of some flaws of the involved sensors. The goal of this thesis is to develop an optimized algorithm to solve this matching problem.

A first chapter introduces the problem from a high-level point of view. Then, after introducing some useful preliminaries, it is discussed in a more detailed way, including an explanation of the available input data, which the cost function measuring the quality of a solution is based on. The following chapters present the heuristic and exact optimization strategies that have been adopted to minimize this function, comparing then their performances and discussing their strengths and weaknesses.

Contents

List of Algorithms

List of Figures

List of Symbols

symbol	name	unit
N	number of axles in a train transit	
C	number of detected tags in a transit (after filtering)	
H	holes: N - total axles in the C cars	
t_j	detection time of axle j, $0 \leq j < N$	s
v_j	detected speed for axle j	m/s
$v_{j,j+1}$	average inter-axle speed between axles j and $j+1$	m/s
t_c	detection time of tag/car c, $0 \leq c < C$	s
l_c	number of axles in car c	
$d_{cl,c}$	tag/closest axle distance in car c	m
$d_{i,i+1}$	distance between axles $i, i+1$ in a car, $0 \leq i < l_c$	m
$d_{W,T}$	distance between wheel sensor and tag reader	m
$start_c$	first axle assigned to car c in solution	

Chapter 1

Introduction

Wheel measurements play a significant role in train diagnostics: they are essential to analyze the conditions of the rolling stock and therefore to perform trend analysis and to understand the correct maintenance timings. The collected measures allow to monitor wheel deterioration, for example in terms of diameter reduction or anomalies in the profile. Figure 1.1 shows an example of an integrated system installed on the railway with this monitoring purpose [1].

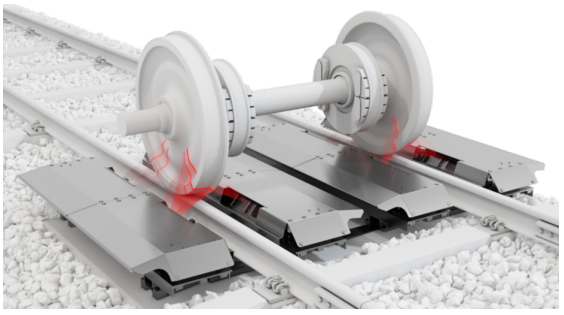

FIGURE 1.1: Wheel profile measurement system.

The detection and measurement system considered in this paper requires the use of RFID tags and readers. Train cars should have on their sides two different tags which uniquely identify them (and their sides); the corresponding reader antennas are placed near the rails (as in figure 1.2), and they provide tag readings along with timestamps. Axles are detected by a *wheel sensor* attached to the rails (figure 1.3), which outputs time and speed for each of them.

This is where a matching problem arises: there are different sensors collecting data which have a meaning only if combined, because there is the obvious but crucial need of assigning the diagnostic measures coming from the wheel system to the right wheel in the right car. To understand why this is essential, it is enough to think about a maintenance technician who has to operate on a specific deteriorated wheel.

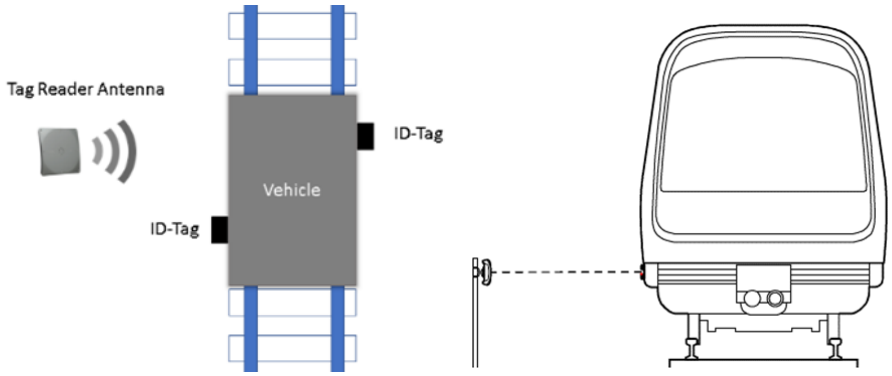

FIGURE 1.2: Positions of RFID tags and corresponding reader.

FIGURE 1.3: Wheel sensor.

Some limitations of the involved sensors, which will be discussed in the following, introduce complications in the problem: for example, it is quite likely that not all RFID tags are detected, or that some of them are physically missing on the cars.

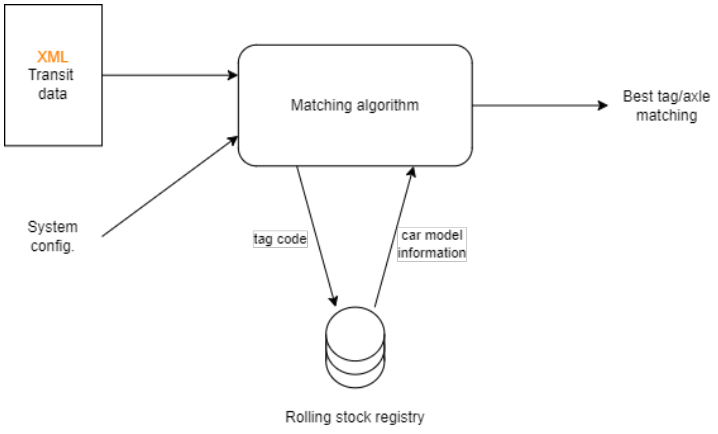

FIGURE 1.4: Expected architecture of the matching algorithm.

The schema of figure 1.4 briefly summarizes what a matching algorithm is then supposed to do.

An automatically generated XML file contains all data which have been collected about a train transit, including axle detection times with corresponding speeds and tag codes associated with timestamps. Some parameters describing the on-field configuration of the system are also available for each physical deployment of it. Then, using this input

information and querying a database with data describing the models in the rolling stock, the solver should compute and output the most reliable matching between axles and tags.

An important non-functional requirement is about efficiency. It must be considered that this algorithm is intended to be integrated in a complex software, collecting and elaborating diagnostic data from the measurement system; if this system is installed right out of a maintenance facility, all the outputs must be available for the operators working in it in a short time. Therefore, running times of the matching algorithm should be analyzed and minimized; a rough estimation of the maximum allowed time is 2-3 seconds.
Another requirement is about the programming language for the solver. Python might work for a first sketch, but the final version should be coded in C# to be easily integrated with the company's software, which is designed to run on Windows machines.

This thesis describes the work that has been carried out to solve the presented problem, considering the related requirements. The following is a summary of the main goals of this contribution:

- development of high-level strategies to tackle the matching problem, considering known approaches from the fields of algorithmics and optimization;
- implementation of these strategies in an efficient way, using C# for the final version;
- analysis and comparison of the performances of the solving algorithms, to understand which one better fits with the requirements.

Chapter 2

Preliminaries

The goal of this chapter is to describe some elements and aspects which will then be used in a black-box way, taking them for granted and allowing to keep the focus on the implementation of the solving strategies. They include theoretical insights, mathematical frameworks and more technical details such as specific libraries which have been used while developing the solving algorithms.

2.1 Optimization Strategies

This section introduces from a general and theoretical point of view the heuristic and exact strategies to solve minimization problems which have been used as an inspiration or directly implemented for this matching problem.
Heuristic strategies are the ones sacrificing the optimality guarantees in order to provide an efficient algorithm, which outputs solutions of reasonable quality in a very short computing time. This paper discusses the implementation of a *greedy* algorithm, of its *GRASP* variation and of a *genetic algorithm*.
An exact strategy, which ensures to reach the optimal solution, will be also presented in the following: it is based on the use of an *Integer Linear Programming* (ILP) model.

2.1.1 Greedy Algorithm

When tackling an optimization problem, one possibility is to develop an algorithm consisting of a series of steps implementing just a *local optimality* policy which, depending on the specific problem, may or may not guarantee global optimality of the final solution. This is the very general pattern for *greedy algorithms* [2].
More formally, let an *independence system* be a pair (E, S), with E being a set and S a subset of the power set of E (so $S \in P(E)$), closed under inclusion (which means that $A \in S, B \subset A \Rightarrow B \in S$). The elements of S are named *independent sets*. The

optimization problem on (E, S) is then the following: for a given weight/cost function w, associating to each item of E a non-negative real value ($w : E \to \mathbb{R}_0^+$), the goal is to find an independent set A whose weight $w(A) := \sum_{a \in A} w(a)$ is minimal.
Then, the actual procedure is quite simple: starting from $T \leftarrow \varnothing$, elements of E must be scanned by increasing cost; if for a certain $e \in E$ which is met it holds that $T \cup \{e\} \in S$, then e is added to T ($T \leftarrow T \cup \{e\}$). After the full scan, T is the output of the greedy algorithm.

Algorithm 1: Greedy algorithm

Input: $(E, S) \mid S \in P(E),\ A \in S \wedge B \subset A \Rightarrow B \in S;\ w : E \to \mathbb{R}_0^+$
Output: $T \in S$ (no global optimality proof)
$T \leftarrow \varnothing$
$E_s \leftarrow$ /* E sorted by ascending $w(e)$ */
for $i \leftarrow 0$ *to* $|E_s| - 1$ **do**
  if $T \cup \{E_s[i]\} \in S$ **then**
    $T \leftarrow T \cup \{E_s[i]\}$
  end
end

Algorithm 1 is the pseudocode description of a general greedy algorithm.

2.1.1.1 GRASP Algorithm

The locally optimal choice performed in the described greedy procedure by picking the available element with minimal cost does not guarantee the final output to be globally optimal in the whole solution space. Therefore, a possible variation on the scheme might be to add some randomization to element choice, allowing some sub-optimal local decisions. This operation becomes meaningful if solution construction is performed multiple times, allowing to explore a broader portion of feasible solutions space. In the end, the delivered output is the best solution obtained in the process.
This pattern goes under the name of *Greedy Randomized Adaptive Search Procedure* (GRASP) [3]. Several variations on this general scheme are possible: one common choice is to select at each step with a large probability p the least expensive element, and with probability $p_{sec} = 1 - p$ the second best one. This is the policy implemented in algorithm 2, where $U([0, 1])$ is a random value uniformly sampled in range $[0, 1]$.

2.1.2 Genetic Algorithm

While the greedy procedure described above is a *constructive heuristic*, in the sense that the output is built through a series of steps based on locally optimal choices, a different option is to work with a set of feasible solutions, trying to improve their quality by combining them. This is the very general idea behind the class of *genetic algorithms* (GAs).

Algorithm 2: GRASP greedy algorithm

Input: $(E, S) \mid S \in P(E),\ A \in S \wedge B \subset A \Rightarrow B \in S;\ w : E \to \mathbb{R}_0^+$
Output: $T^* \in S$ (no global optimality proof)

```
T* ← ∅
while /* time lim. not expired*/ do
    T ← ∅
    E_s ← /* E sorted by ascending w(e) */
    for i ← 0 to |E_s| − 1 do
        if i < |E_s| − 1 ∧ U([0,1]) < p_sec then
            swap(E_s[i], E_s[i + 1])
        end
        if T ∪ {E_s[i]} ∈ S then
            T ← T ∪ {E_s[i]}
        end
    end
    if T* == ∅ or w(T) < w(T*) then
        T* ← T
    end
end
```

They are *metaheuristics* [4], which are problem-agnostic optimization frameworks consisting of an iterative procedure to reach the final solution; this procedure usually relies internally on a problem-specific heuristic. As a consequence, it is not trivial to define GAs in general, but they share some common important features which are heavily inspired by Darwin's Theory of Evolution.
After Ingo Rechenberg was the first to propose *evolution strategies* for optimization problems in the 1960s, an important milestone was set by John Henry Holland in 1975, who provided some theoretical and mathematical basis for the field. A further contribution was the one of Melanie Mitchell [5], who described some common features of GAs in the late 90s. Here are the main ones:

- a *population*, namely, a set of individuals (identified by their encodings, the *chromosomes*) which are feasible solutions of the optimization problem;
- a *crossover* policy, which is used to generate new individuals from existing ones, chosen randomly or accordingly to some *selection* procedure;
- *mutations*, introducing some random variability in chromosomes.

A positive aspect of genetic algorithms is that they provide a very general and customizable procedure to solve various problems, even with a quite agnostic approach. On the other hand, this lack of specificity might lead to a loss in terms of performances. These are also the reasons for which no pseudocode is provided here, since it is easier to directly describe the specific implementation for this problem, and this will be done in the following.

2.1.3 Integer Linear Programming

The other way is to *exactly* solve the considered minimization problem, which means to provide a solution with global optimality guarantees (with respect to its cost) in the whole solution space. The mathematical framework which comes to rescue is the one of *Linear Programming*: this paper exploits in particular an *Integer Linear Programming* (ILP) problem.
In general, an ILP problem consists of:

- a set of integer variables;
- a cost function which is linear with respect to the variables and which must be minimized;
- a set of constraints (linear inequalities) involving the variables.

This can be easily summarized using matrixes, in *canonical form*:

$$\begin{cases} min\ \mathbf{c^T x}\ s.t. \\ \mathbf{Ax} \leq \mathbf{b} \\ \mathbf{x} \geq \mathbf{0},\ \mathbf{x} \in \mathbb{Z}^n \end{cases}$$

where $\mathbf{c} \in \mathbb{R}^n,\ \mathbf{b} \in \mathbb{R}^m,\ \mathbf{A} \in \mathbb{R}^{m \times n}$: $m, n \in \mathbb{N}$ represent respectively the number of constraints and the number of variables.
Standard form exploits instead a further vector $\mathbf{s}$ of non-negative *slack variables*:

$$\begin{cases} min\ \mathbf{c^T x}\ s.t. \\ \mathbf{Ax} + \mathbf{s} = \mathbf{b} \\ \mathbf{s} \geq \mathbf{0} \\ \mathbf{x} \geq \mathbf{0},\ \mathbf{x} \in \mathbb{Z}^n \end{cases}$$

These two formulations can be proved to be equivalent [6]: in fact, each inequality can be transformed into an equality, with a non-negative slack variable representing the difference between the values of the left-hand side and the right-hand side.
An important variant of ILP is *Binary Integer Programming*, or *0-1 Linear Programming*, in which the last constraint is more specific and states that $x \in \{0, 1\}^n$. In the most common applications of this variant, each variable represents a decision, namely, if to include or not the corresponding element in the solution.
ILP is NP-hard, and this can be proven for example by reduction from vertex cover problem (which simply consists of writing its ILP formulation). This results in an exponential upper-bound (with respect to the number of variables n) for worst-case complexity of a solving algorithm. However, it must be accounted that ILP problems can be solved through different iterative strategies (e.g. *branch-and-bound* method)

which rely on solving the *LP relaxation* of such problems (i.e. discarding integrality constraints), and that the resulting LP problems can be solved in pseudo-polynomial time [7]. Furthermore, state-of-the-art LP solvers use internal heuristics to speed up the computation; as a consequence, it is not easy to precisely estimate the time complexity associated with the solution of an ILP problem, but in a lot of cases the exponential upper bound is in practice very loose.

2.1.3.1 Lazy Constraints

If the number m of constraints of the ILP problem is considerably large, solving it might turn out to be a computationally expensive operation. This is the reason for the introduction of *lazy constraints* [8]: their simple but powerful idea is that it might be more efficient to initially leave some inequalities out of the problem, solving it in this simplified version and then checking if these constraints are satisfied: if not, they are re-introduced in the model and the problem is solved again. This process is repeated until no inequalities are violated: at that point, the feasibility and optimality of the solution are guaranteed.

Algorithm 3: ILP solution with lazy constraints

Input: $\mathbf{A} \in \mathbb{R}^{m \times n},\ \mathbf{b} \in \mathbb{R}^{m},\ \mathbf{c} \in \mathbb{R}^{n}$
Output: $x* \in \mathbb{Z}^{n}$ (global optimal sol.)
$L \leftarrow$ /* indexes of lazy constr. rows in A */
$\mathbf{A_l},\ \mathbf{b_l} \leftarrow [\mathbf{a_i}],\ [b_i]\ \forall i \in L$ //lazy constr. rows of $\mathbf{A}$ and corresp. elems. of $\mathbf{b}$
$\mathbf{A_{nl}},\ \mathbf{b_{nl}} \leftarrow [\mathbf{a_i}],\ [b_i]\ \forall i \in \{1...m\} - L$ //the other rows
while *True* **do**
  $prob \leftarrow \{min\ \mathbf{c^T x}\ s.t.\ \mathbf{A_{nl}x} \leq \mathbf{b_{nl}},\ \mathbf{x} \geq \mathbf{0},\ \mathbf{x} \in \mathbb{Z}^{n}\}$
  $x* \leftarrow$ solve($prob$)
  if $\mathbf{A_l x} \leq \mathbf{b_l}$ **then**
    break //lazy constr. not violated: final solution reached
  end
  for $i \in L$ **do**
    if *not* $\mathbf{a_i} x* \leq b_i$ **then**
      $L \leftarrow L - \{i\}$ //activate constr. i
    end
  end
end

Algorithm 3 summarizes the lazy constraint approach to solve an ILP problem; `solve()` is an implementation-dependent black-box function to solve an ILP model.

2.1.3.2 ILP Solvers

This section briefly introduces the specific libraries which have been used to solve with a black-box approach the ILP problem which has been written for this paper, and which

will be presented in the following.
For what concerns the Python implementation, the choice has been to use *CVXOPT* package, and more specifically its `cvxopt.glpk` module allowing the use of *GLPK* (GNU Linear Programming Kit) optimization library. In particular, the API of this library provides the `ilp(...)` function to solve mixed ILP problems (MILPs), where it is not required that all variables are integer. It solves problems in the form:

$$\begin{cases} min\ \mathbf{c^T x}\ s.t. \\ \mathbf{Gx} \leq \mathbf{h} \\ \mathbf{Ax} = \mathbf{b} \\ x_k \in \mathbb{Z},\ \forall k \in I \\ x_k \in \{0,1\},\ \forall k \in B \end{cases}$$

so the mandatory parameters of the function are $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^p$, $\mathbf{A} \in \mathbb{R}^{p \times n}$, $\mathbf{h} \in \mathbb{R}^m$, $\mathbf{G} \in \mathbb{R}^{m \times n}$ and $I, B \subset \{1...n\}$. The output consists of a status string paired with an optimal, feasible or undefined value for $\mathbf{x}$ depending on the status.
In the C# implementation, the *LPsolve* library was used. It provides a clear, well-documented and versatile API to the underlying MILP solver. Its features are similar to the ones provided by *CVXOPT* package; linear constraints of any type ($\leq$, $\geq$ or $=$) can be added one-by-one by using an array to store the coefficients for every variable.

Chapter 3

Problem Formulation

This chapter is intended to discuss the details of the problem, also providing a useful glossary for the following analysis. All the input data will be introduced and described, and also the expected features of the output of a solver will be clarified.
Then, a cost function expressing the quality of a certain matching solution will be introduced, motivating the underlying choices and discussing its limits. It will be presented here because it is based on the input data and because it is independent from the actual solving strategy which will be applied.

3.1 Input Data

This section provides some details on input data and on their sources, also discussing the most relevant issues related to them.
These input data include:

- transit data, which in turn consist of axles and tag detection data;
- system configuration measures, describing the physical deployment of the detection system and its sensors;
- some items from the rolling stock database, which describe car models from a geometrical point of view.

Although one might claim that proper input data are only transit data, system configuration and rolling stock registry are also included in this description, since they are supposed to be used along transit data in the computation of solution cost performed by each solver.

3.1.1 Transit Data

This section provides some information about the data collected by the system on train transit. First of all, it is worth to introduce some notation that will be used in the following:

- N is the total number of detected axles;
- C is the number of detected tags which will be used to label the axles (some filtering might be applied to exclude spurious detections; this will be discussed later in this paper);
- H is the difference between N and total number of axles of the C cars (namely, the *holes*): it is then the number of unlabeled axles in final solution.

3.1.1.1 Axles Detection

For the purposes of this work, the inner implementation of the wheel measurement system can be considered transparent: it is enough to know that a timestamp is generated for the detection of each of the N train axles, along with a reliable estimate of its speed. Time 0 corresponds to the instant when the train enters the measuring system.

$$\text{axles data: } (t_j, v_j),\ 0 \leq j \leq N$$

3.1.1.2 Tag Detection

Tag information is collected by a reader which is an antenna installed on one side of the railways. It outputs the codes associated with the RFID tags it detects, which are strings of fixed length, along with a timestamp. This time measure is synchronized with the one of the wheel system, and it is affected by an uncertainty which has been estimated as 300 ms.

$$\text{tag/car data: } (code_c, t_c),\ 0 \leq c \leq C$$

Besides the uncertainty, tag detection might suffer of different problems:

- some tags from a train transiting on parallel rails might be read;
- a train might have some cars with no tags or with deteriorated and thus unreadable ones;
- the tag reader might miss some readings.

These issues represent a first and very impactful reason why car-axles matching is a non-trivial problem. In fact, having missing readings requires to choose some axles which will not get classified, while having spurious ones means that some tag codes must be discarded. Moreover, the fact that trains have a dynamic composition implies that an agnostic approach with respect to the overall train structure must be adopted: no reliable prediction on the sequence of cars can be done. Therefore, in order to deal with the aforementioned issues different heuristic choices have to be made: they will be discussed in the following, along with the underlying intuitions.

3.1.2 System Configuration

It is also worth to have a clear understanding of the overall physical configuration of the measuring system, in particular about the reciprocal positions of the different involved sensors. In fact, an effective solving algorithm needs to consider this information to correctly integrate input data from tags and from axles.
Therefore, the schema of figure 3.1 visualizes the standards which have been adopted to make the system configuration measurable.

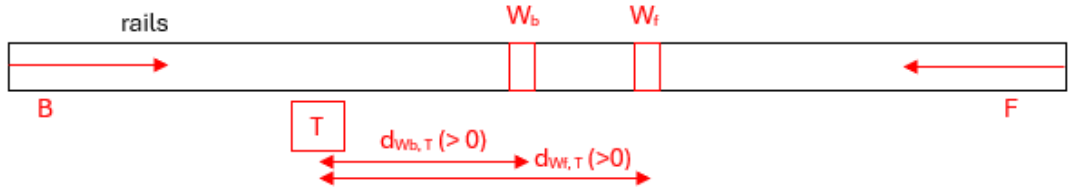

FIGURE 3.1: Physical configuration of a measuring system.

First, the running direction of the train is said to be *forward* (F) or *backward* (B), according to some arbitrary convention. One of two different wheel measurement system units (W_b, W_f) is activated on axle transits, depending on train direction.
Then, using the forward direction as a Cartesian axis, the tag reader position can be univocally described with:

- its side with respect to the rails (left, like in the example, or right);
- the distance between W_f and the tag reader T, which is positive if the tag reader is placed after the wheel system (like in the above schema) and negative otherwise;
- the distance between W_b and T, with the same convention.

In different deployments of the system the values of these parameters can be substantially different, because of the variability in terms of functional requirements.

3.1.3 Rolling Stock Registry

This section briefly presents the most useful information about train car models which can be extracted from the rolling stock registry database using the tag code as a key. In fact, some further details are essential for any matching strategy: it is enough to think about the number of axles of the car associated to a certain tag.

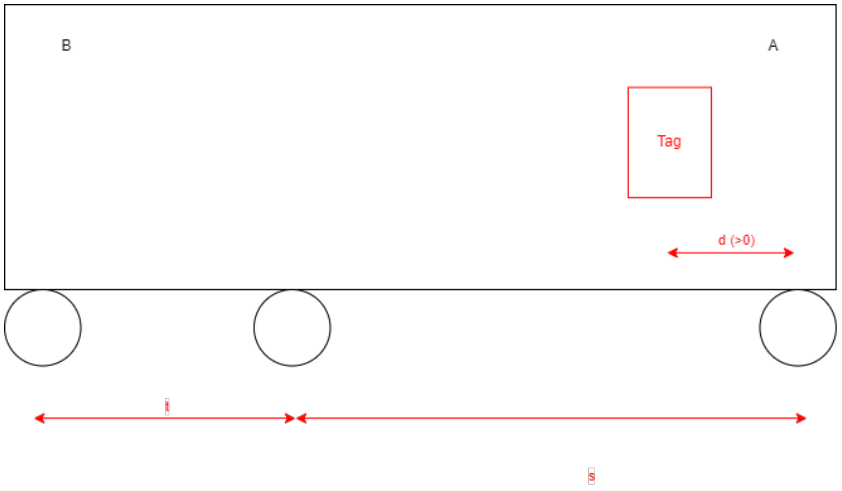

FIGURE 3.2: Main car details in the database.

The model-specific information which have been used in the proposed solutions are:

- tag side: given the fact that each car is assigned a *head* (A) and a *tail* (B), a tag can be on its left or right side, if watching the car from above it and keeping its head on top. For example, in the above schema, the tag is on the right. Cars should have (different) tags on both their sides;
- number of axles l_c and *footprint*. The footprint represents the sequence of the (l_c - 1) inter-axle distances in a car, starting from its head. In the example, $l_c = 3$ and the footprint is *s*-*t*;
- index of the axle which is closest to the tag of this car, with a numbering starting from car head and from 1: in the example, this index is equal to 1;
- horizontal distance between car tag and closest axle (*d* in the schema): its value is positive if the tag is closer to the tail B than to the axle, and negative otherwise.

These data are supposed to be reliable, in the sense that every car of a certain model in the fleet can be assumed to be compliant to the description registered in the database.

3.2 Solver Output

It is also essential to clarify what a solver must output, and so what a *feasible solution* to this matching problem consists of. A solution can be expressed in different ways and with different formats, but it is essentially a labeling of all detected axles, assigning to each of them the corresponding car (identified by the code associated with its RFID tag). Furthermore, it is required to output if each car is *reversed*: a car is said to be reversed if its orientation (head-tail) is the opposite of the running one of the train.
A feasible solution must comply with some requirements:

- each detected axle must be assigned to one single car or left unlabeled;
- no isolated axles can be left unlabeled: in fact, no undetected car can have just one single axle;
- only consecutively detected axles might have the same label, and labels have to follow the chronological order of RFID tag detections.

One compact way to describe a solution is then by indicating the *starting index* of each car, which is the index of the first axle assigned to that car, if numbering the detected axles by chronological order. The solution can be then represented by an array-like structure, in which the index represents the car (if cars are also numbered by chronological order of corresponding tag detections) and the value is a pair consisting of the starting axle index of that car and of a flag telling if it is reversed:

$$sol[c].start \in \{0...N-1\},\ sol[c].reversed \in \{\text{True, False}\}\ \forall c \mid 0 \leq c < C$$

In this representation, it must then hold that:

- $sol[c+1].start \geq sol[c].start + l_c\ \forall c \mid 0 \leq c < C$ (a)
- $sol[0].start \neq 1,\ sol[C-1].start \neq N - l_{C-1} - 1,\ sol[c+1].start \neq sol[c].start + l_c + 1\ \forall c \mid 0 \leq c < C-1$ (b)

In fact, inequality (a) formalizes the feasibility requirement about car ordering and no overlapping labels, while inequality (b) forbids the presence of isolated unlabeled axles.

Finally, picture 3.3 is a visual interpretation of a feasible solution, automatically plotted using Python's *matplotlib* module functions. On the x-axis there are the indexes of detected axles, in chronological order; each rectangle identifies a set of axles with the same label, which are supposed to belong to the same car. In other words, the picture shows what the structure of the transiting train is supposed to be according to the solver, which "places" the cars upon the detected axles.
Each color represents a car model: in this example, yellow cars are locomotives. The

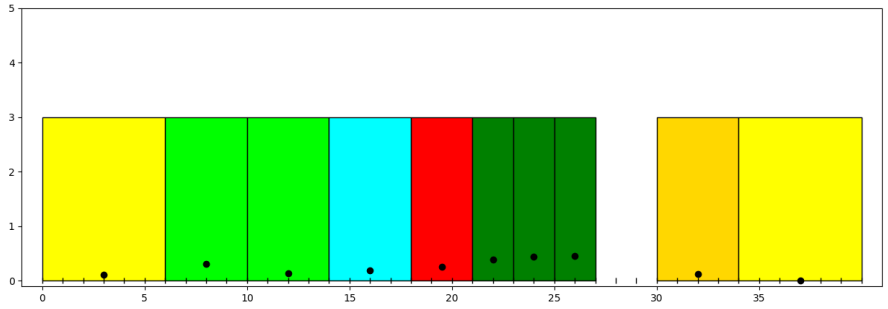

FIGURE 3.3: Visual representation of algorithm output.

y-coordinate of black dots represents the cost of each car; this concept will be introduced in the next section. Finally, in these kind of plots, a hatched rectangle means that the corresponding car is reversed.

3.2.1 Solution Space

One further aspect that is worth analyzing at this point is the size of the solution space, namely, the set of all feasible solutions.
The total number of feasible solutions is the total number of ways in which one can tag the whole sequence of detected axles following the rules described in previous chapter. It can be recursively computed as a function of the number of detected cars and the number of holes:

$$f(C,H) = \begin{cases} 0 & \text{if } H = 1 \\ 1 & \text{if } H = 0 \vee (C = 0 \wedge H \neq 1) \\ C'_{C+1,H} - \sum_{i=1}^{min(C,H)} f(C-i, H-i) & \text{oth.} \end{cases}$$

In fact, it must be noted that:

- no feasible labeling choice is possible if there is only one hole (it would be an isolated one);
- if no holes (or, in a corner case, no cars) are present, there is only one possible labeling choice;
- in all other situations (third row of the function definition):
  - $C'_{C+1,H}$ (combinations with repetitions) is the number of ways in which H holes can be placed in $C+1$ positions (before first car, between subsequent cars or after last one);
  - all cases in which one or more isolated holes are present must be subtracted from this total: the solutions with exactly i single holes can be obtained

by placing them in $C_{C+1,i}$ different ways, and then feasibly arranging the remaining holes in the remaining positions (without further isolated ones), so in $f(C-i, H-i)$ possible ways.

A loose upper bound for this function is then given by $C'_{C+1,H} = \binom{C+H}{H}$: practically speaking, it means considering also solutions with isolated holes. This binomial coefficient can be proven to be $O((\frac{e(C+H)}{H})^H)$. Excluding the trivial case with $H = 0$, it is possible to assume $C \in O(H)$: we have then that the solution space has size $f(C, H) \in O(e^H)$.

3.3 Cost Function

The goal of this section is to present the cost function which has been used to evaluate the quality of a certain feasible matching solution. Although deciding *what* must be solved to optimality and *how* to do that are separate tasks, picking a cost function in a context in which uncertainty plays an important role is inevitably a trial-and-error procedure, and therefore some adjustments which will be discussed in this section have been implemented after testing the cost function with different solving strategies.
The intuitive question one can ask himself to evaluate a solution is "given the input data, how likely this tagging/labeling choice is?". The key is then to formalize this likelihood to obtain a cost function associating to each car and then to the whole solution a non-negative real number:

$$z_{car} : \ (c, \ start_c) \to \mathbb{R}_0^+ \quad z : \ sol \to \mathbb{R}_0^+$$

In this work, the car cost is based on two different components:

- one considers inter-axle times;
- one is about the elapsed time between a tag reading and the corresponding closest axle detection.

3.3.1 Inter-axle Cost Component

The underlying idea here is to compare the real time elapsed between the detections of subsequent axles with an estimation based on the available data and the tagging choices in current solution. If car c has been placed on axles with indexes $\{start_c, \dots start_c + l_c - 1\}$ in transit data, it means that their real reciprocal distances should correspond to the terms $d_{i,i+1}$ in the footprint of the same car for each $i \in \{0 \dots l_c - 2\}$. Then, one can get an estimated inter-axle time out of the real transit speeds and the terms of the

footprint of the car which is projected on the considered axles:

$$\Delta t^c_{j,j+1} := \frac{d_{i,i+1}}{v_{j,j+1}}|_{i=j-start_c}$$

where $v_{j,j+1} := \frac{v_j+v_{j+1}}{2}$ is the average inter-axle speed, for each $j \in \{0...N-2\}$. Note that j is an absolute index spanning the whole set of detected axles, while i is a relative one, spanning the axles in currently analyzed car: it then holds that $i = j - start_c$.
If $\Delta t_{j,j+1} := t_{j+1} - t_j$ is the real inter-axle time corresponding to the same axles, a good solution is then supposed to have a small distance between it and the estimated one.
For a whole single car, the inter-axle contribution to cost can be therefore computed as:

$$z_{car,ia}(c,\ start_c) := \sum_{j=start_c}^{start_c+l_c-2} |\Delta t_{j,j+1} - \Delta t^c_{j,j+1}|$$

where $start_c$ is the first axle assigned to car c and l_c is the number of axles of the car itself.

3.3.1.1 Issues and Uncertainty

One situation in which time estimation becomes unreliable is when trains stops while transiting on the wheel measurement system. The problem is that, if the train stops between the detections of axles j and $j+1$, then $v_{j,j+1}$ computed as the average between v_j and v_{j+1} is a too large estimate of the real average speed in that time interval. A possible workaround, which is implemented here, is to take the largest term D in all footprints for this train and to use a corrected formula for the inter-axle speed:

$$v_{j,j+1} := \begin{cases} \frac{v_j+v_{j+1}}{2} & \text{if } \frac{v_j+v_{j+1}}{2} < TOL\ \frac{D}{\Delta t_{j,j+1}} \\ \frac{D}{\Delta t_{j,j+1}} & \text{otherwise} \end{cases}$$

D is chosen with the goal of computing an upper bound to the inter-axle speed, in a way which is independent on the car which is placed on these axles in the considered solution; $TOL = 1.1$ coefficient is introduced to provide some tolerance.
Another more general issue is that the cost, computed as described, always comes with an uncertainty: it is due to the sensitivities of the sensors, but most of all to the estimation of average inter-axle speeds.
Considering that this is an underestimation in case the train both speeds up and slows down between the detection of axles j and $j+1$, the absolute and relative errors associated to speed $v_{j,j+1}$ can be computed as:

$$e(v_{j,j+1}) = \frac{|v_{j+1} - v_j|}{2},\ e_r(v_{j,j+1}) = \frac{|v_{j+1} - v_j|}{2v_{j,j+1}} = \frac{|v_{j+1} - v_j|}{v_{j+1} + v_j}$$

Given that the distances in the footprint can be considered as exact values, $e_r(v_{j,j+1})$ is also the relative error on $\Delta t^c_{j,j+1}$.
If not considering sensitivities, the absolute error on $z_{car,ia}(c,\ start_c)$ is equal to the one on $\Delta t^c_{j,j+1}$ and so it holds that:

$$e(z_{car,ia}(c,\ start_c)) = \sum_{j=start_c}^{start_c+l_c-2} e_r(v_{j,j+1})\Delta t^c_{j,j+1} = \sum_{j=start_c}^{start_c+l_c-2} \frac{d_{i,i+1}|v_{j+1} - v_j|}{2v^2_{j,j+1}}|_{i=j-start_c}$$

It can be observed that this uncertainty is larger for low axle speeds: not only for the fact that the denominator increases, but also because when the speed is near to zero it is very likely that the train is breaking or accelerating, resulting in a larger gap between the speeds of subsequent axles. As a consequence, in these cases the average speed might not be a reliable estimation of the real inter-axle speed at a specific index.

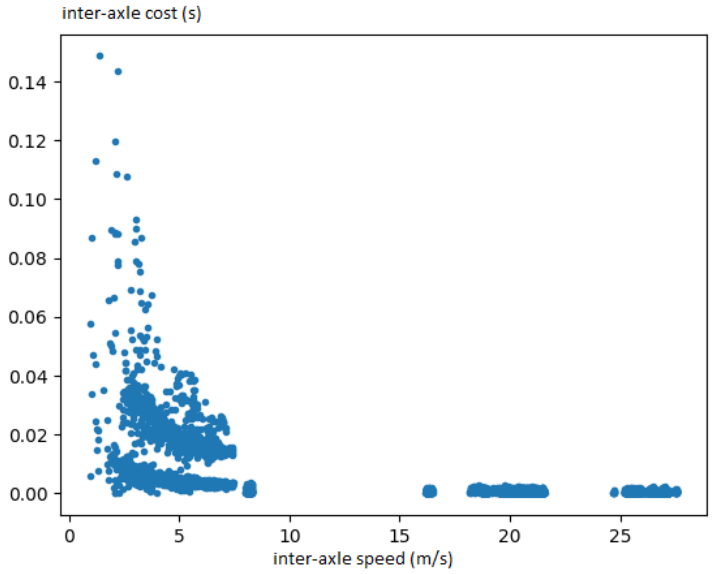

FIGURE 3.4: Inter-axle cost against inter-axle speed.

Picture 3.4 plots the inter-axle cost against the corresponding inter-axle speed considering all the (non-stopping) transits with a reliable output; previous observations about uncertainties at low speeds are confirmed by the sparser distribution of costs in the left side. Furthermore, two roughly separated clusters of points can be observed in the same area: this can be explained considering the other values used in the formula for the uncertainty, which are the inter-axle distances. In fact, their values in the rolling stock database are not uniformly distributed, with two peaks around 1-2 m and around 10 m that are the cause of the behaviour observed in the plot.

3.3.2 Tag-Closest Axle Cost Component

Different tags might correspond to identical cars, and then using only the inter-axle time estimation as a cost measure wouldn't be a reliable choice. In fact, it is worth to also consider the reading time of each tag, and in particular the time elapsed between it and the detection of the corresponding closest axle in current feasible solution. As for the axles, there is a real value for this time, and it can be compared to an estimated one coming from the real speeds and the shape of projected cars (along with system configuration parameters). Let then:

$$z_{car,ta}(c,\ start_c) := |\frac{\Delta s_{cl,c}}{v_{cl,c}} - (t_c - t_{cl})|$$

be the tag-closest axle cost component, where:

- t_c is the reading time of the tag associated to this car, and t_{cl} is the time in which the closest axle according to this car placing is detected;
- $\Delta s_{cl,c}$ is the distance the train has travelled in the same time interval;
- $v_{cl,c}$ is the weighted average of inter-axle average speeds (after applying the patch described above for stopping transits) corresponding to all axles detected between t_{cl} and t_c (or the opposite, if the tag is detected before closest axle), using inter-axle times as weights:

$$v_{cl,c} := \frac{\sum_{t_a \leq t_j < t_{j+1} \leq t_b} v_{j,j+1} \Delta t_{j,j+1}}{\sum_{t_a \leq t_j < t_{j+1} \leq t_b} \Delta t_{j,j+1}} |_{t_a = min(t_{cl}, t_c),\ t_b = max(t_{cl}, t_c)}$$

It is worth to note that for $\Delta s_{cl,c}$ four different cases are possible, depending on the combination of train direction and car face exposed. For example, the schema of figure 3.5 represents the situation in which the train is running forward, and the tag (seen by a tag reader placed on the right of the rails) is the right one of the car: this means that car direction is the same of the train, and so that the car is not reversed. Then, in this case, it holds that $\Delta s_{cl,c} = d_{cl,c} + d_{W,T}$. The schema represents the instant t_{cl} in which axle cl is detected (in this example, the first one of the car). The train will then run a distance of $\Delta s_{cl,c}$ with an average speed estimated by $v_{cl,c}$ until the tag will reach the reader, which will happen at instant t_c.
In the other three possible cases, the formula for $\Delta s_{cl,c}$ must be corrected taking the opposite of $d_{cl,c}$ if the car is reversed, and taking the opposite of $d_{W,T}$ if the train is running backwards.
Similar reasonings must be done to compute the absolute index of closest axle in $\{0, \dots N-1\}$, and so to obtain t_{cl} and $v_{cl,c}$.

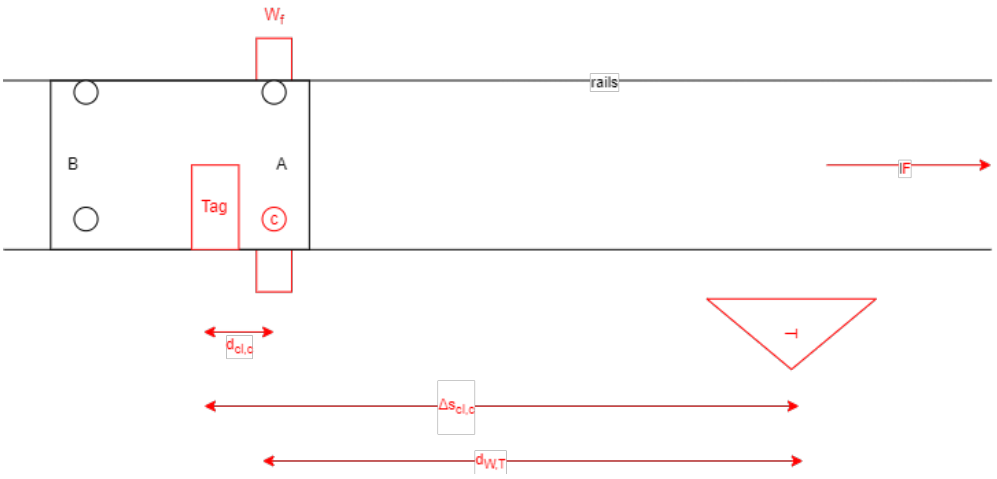

FIGURE 3.5: Distance travelled by the train between tag and closest axle detections.

3.3.2.1 Issues and Uncertainty

The most impactful issue is again the one of stopping trains: here, the problem arises if train stops between the detection of a tag and closest axle in current solution. In these cases, the estimate on the average speed becomes unreliable. The same workaround used for the inter-axles is then applied.
The uncertainty on this cost component can be computed in a similar way, too. Here, tag reader sensitivity cannot be considered negligible, and it is estimated in 300 ms, as previously mentioned:

$$e(z_{car,ta}(c,\ start_c)) = e(v_{cl,c})\frac{\Delta s_{cl,c}}{v_{cl,c}^2} + 0.3s$$

where $e(v_{cl,c})$ is the weighted average of all inter-axle speed uncertainties in the same range in which $v_{cl,c}$ is computed:

$$e(v_{cl,c}) = \frac{\sum_{t_a \le t_j < t_{j+1} \le t_b} \frac{|v_{j,j+1} - v_j|}{2} \Delta t_{j,j+1}}{\sum_{t_a \le t_j < t_{j+1} \le t_b} \Delta t_{j,j+1}} |_{t_a = min(t_{cl}, t_c),\ t_b = max(t_{cl}, t_c)}$$

Figure 3.6 shows the plot of tag-closest axle cost component against $v_{cl,c}$, from the same reliable matching outputs used in previous section. The same problem discussed there for low speeds can be observed here; in this case the variance is larger because $\Delta s_{cl,c}$ is often one order of magnitude larger than the terms in footprints of car models.

3.3.3 Final Cost

The two cost contributions which have been described in previous section must be added to obtain the actual cost of a car; the total cost of a solution is the average of all car

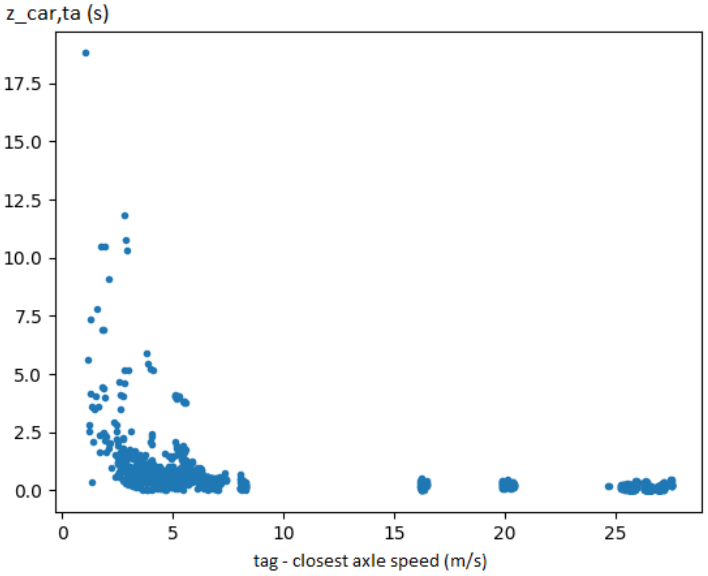

FIGURE 3.6: Tag-closest axle cost against average inter-axle speed.

costs.

$$z(sol) := \frac{1}{C}\sum_{c=0}^{C-1} z_{car}(c,\ start_c) = \frac{1}{C}\sum_{c=0}^{C-1} \frac{z_{car,ia}(c,\ start_c) + z_{car,ta}(c,\ start_c)}{l_c}$$

Notice that the cost of each car contains a normalization with respect to the number of components contributing to it ($l_c - 1$ inter-axle components and the tag-closest axle one, so l_c in total).
Some variations might be the use of different weights for the two cost components in each car, or to add the (possibly weighted) standard deviation of car costs to solution cost. This latter option would penalize matching choices in which some cars have a too privileged placing to disadvantage of other ones. In this case, these fine-tunings have proven to suffer of some "overfitting", being beneficial in some problematic transits but not in other ones: therefore, this simpler one is the cost function which has actually been used.
A similar formula holds also for the total uncertainty:

$$e(z(sol)) = \frac{1}{C}\sum_{c=0}^{C-1} e(z_{car}(c,\ start_c)) = \frac{1}{C}\sum_{c=0}^{C-1} \frac{e(z_{car,ia}(c,\ start_c)) + e(z_{car,ta}(c,\ start_c))}{l_c}$$

The plots in figure 3.7 represent the costs of best matching solutions for two different transits, in ascending order and associated with corresponding uncertainties (the error bars). The plot in the right side corresponds to a train transiting at a very low speed on the wheel system, and thus inducing very large uncertainties in the estimation of average speeds.
This could be a starting point for a probabilistic interpretation of solution cost in which,

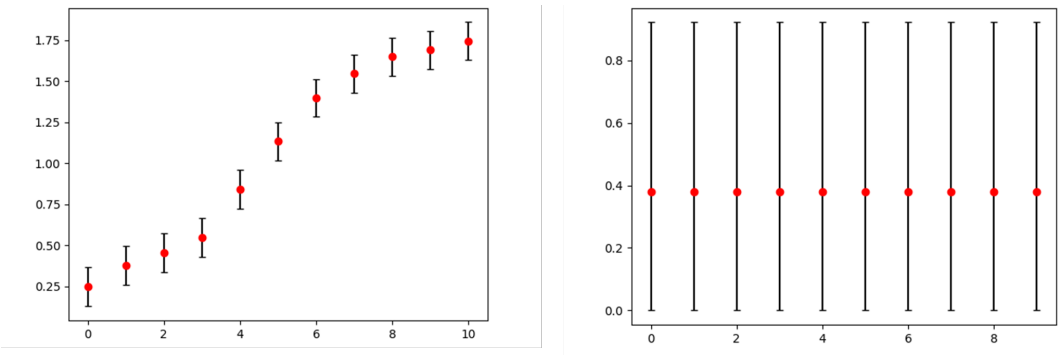

FIGURE 3.7: Uncertainties in the 10 best solutions for two different transits.

instead of minimizing an objective function, the goal would be to find the feasible matching having the maximum probability to be the best one in solution space. In other words, letting Z_s be a random variable representing the cost of solution s and S the space of feasible solutions, then the optimal output would become:

$$argmax_{s \in S} \; P[Z_s \leq Z_{s'} \; \forall s' \in S]$$

However, some preliminary experiments carried out on the most problematic transits have suggested that in those cases a distribution function inferred from uncertainties would not succeed in clearly differentiating between candidate solutions, because of too large variances. As a consequence, the following chapters only focus on the implementation of different deterministic optimization strategies.

Chapter 4

Solving Strategies

This chapter is intended to present the solving algorithms which have been implemented and tested, highlighting their strengths and flaws. These solvers are inspired by or directly implement the known optimization approaches described in the preliminaries, and they can be divided into:

- exact algorithms, requiring more computing time but ensuring the optimality of the solution;
- heuristic algorithms, running in fast time but providing no guarantees about cost function minimization.

4.1 Preliminary Operation: Tag Filtering

As previously mentioned, one possible problem is that the tag reader might detect some tags belonging to a different train nearby. This is particularly evident in some corner cases in which the total number of axles of the cars corresponding to the detected tags overflows the actual number N of axles transiting on the wheel system.
A possible workaround is to order tags by decreasing *best cost*, and to leave out the ones on top of this list, in a number allowing to avoid the overflowing condition. The *best cost* of a car in this context is the minimum cost associated with it considering all possible placings on the whole set of axles, forgetting the presence of the other cars:

$$z^*_{car}(c) := min_{\ 0 \leq start_c \leq N - l_c}\ z_{car}(c,\ start_c)$$

The underlying idea of this approach is to avoid placing cars which, even in their absolute best possible positioning choice, would be the least reliable ones in the input data. This filtering is performed before applying any solving strategies, and it can be summarized as in algorithm 4.

Algorithm 4: Spurious tag filtering

Input: Matching alg. input.
Output: Matching alg. input, with filtered tag data.
$carAxles \leftarrow \sum_{i=0}^{C-1} l_i$
if $carAxles > N$ **then**
  $bestCosts \leftarrow [\,]$
  for $c \in \{0...C\text{-}1\}$ **do**
    $z_c* \leftarrow min\ _{0 \leq start_c \leq N - l_c}\ z_{car}(c,\ start_c)$
    $bestCosts \leftarrow bestCosts + [(c,\ z_c^*)]$
  end
  /* sort $bestCosts$ by descending z_c^* */
  $i \leftarrow 0$
  while $carAxles > N$ **do**
    $j \leftarrow bestCosts[i][0]$
    $carAxles \leftarrow carAxles - l_j$
    /* remove car j from detected tags */
    $i \leftarrow i + 1$
  end
end
else
  /* no overflow to be corrected */
end

It is worth to observe that avoiding the overflow in the number of axles is necessary but not sufficient to ensure that all spurious tags are removed; however, experiments on test data have showed that more effective filtering strategies should probably consider at least some minimal knowledge about the expected structure of trains. Since the solutions presented here adopt an agnostic approach with respect to it, this algorithm based on a minimal feasibility requirement is the one which is applied.

4.2 Exact Strategy: ILP Model

A naïve approach to find the optimal solution might be to iteratively explore the whole space of feasible solutions, returning the one with smaller cost. This approach (which is an example of *exhaustive search*) becomes easily very inefficient: in fact, as discussed above, the number of feasible solutions can be considerably large. Furthermore, computing the cost of a solution does not come in constant time $O(1)$ but in $O(CN)$, because of the use of $O(N)$ inter-axle speeds and times in computing the average tag-axle speed for each of the C cars.
A more meaningful minimization strategy providing optimality proof can be obtained formalizing all the requirements which have been so far described in a linear programming model, and particularly an integer one (ILP), which in this case is also binary. The choices made in terms of variables and constraints are discussed in this section.
First of all, one binary integer variable represents each possible placing of a certain car;

it is worth to note that each car has a limited range of values for its starting axle index ($start_c$), because of the constraint that every previous and following car must be placed, too. To be more precise, the most left and right placings of car c occur respectively when 0 or H holes in total are placed before c, which can then "slide" in a range of length $H + 1$, causing $H + 1$ variables to be introduced in the model for each car. A minimum starting index can then be defined for each car c, and it is equal to the total number of axles of cars detected before it, since all of them must fit in axles range:

$$minstart_c := \sum_{k=0}^{c-1} l_k$$

It can be observed that a transit with $H = 0$ is trivially solved assigning $start_c = minstart_c$ for each car c.

Otherwise, let $x_{ij} := \begin{cases} 1 & \text{if car } i \text{ is placed at offset } j \\ 0 & \text{otherwise} \end{cases} \quad \forall i \mid 0 \leq i < C,\ \forall j \mid 0 \leq j \leq H$

where:

- i is the index of the car, from the chronological order of detected tags;
- j is the offset of its starting index with respect to $minstart_i$; it can be interpreted as the total number of holes which have been placed in current solution before car i.

In other words, $x_{ij} = 1$ holds if and only if $start_i = j + minstart_i$ in the selected feasible solution.

At this point, the cost function which must be minimized can be formulated as

$$\sum_{i=0}^{C-1} \sum_{j=0}^{H} c_{ij} x_{ij}$$

where c_{ij} is the cost associated to car i if it is placed with offset j:

$$c_{ij} := z_{car}(i,\ start_i + j)$$

A unique placing for each car must be ensured: then, one can add a constraint stating that $\sum_{j=0}^{H} x_{ij} = 1,\ \forall i \mid 0 \leq i < C$. In fact, it guarantees that a single offset is selected for each detected car.

Three aspects of "physical" feasibility still need to be ensured:

- car placings must comply with the chronological order of tag detections;
- for each axle there might be at most one tagging choice: in other words, adjacent cars cannot overlap;

- no isolated holes can be present in the output because no train cars have only one axle.

The following constraints ensure that the solution does not violate the above requirements.

$x_{ij} + \sum_{k=0}^{j-1} x_{i+1,k} + x_{i+1,j+1} \leq 1,\ \forall i \mid 0 \leq i < C-1,\ \forall j \mid 0 \leq j < H$ (1)
$x_{iH} + \sum_{k=0}^{H-1} x_{i+1,k} \leq 1,\ \forall i \mid 0 \leq i < C-1$ (2)
$\sum_{i=0}^{C-1} x_{i,1} + x_{i,H-1} = 0$ (3)

In particular, constraints (1) and (2) ensure that if a car is placed with offset j (so $x_{ij} = 1$), then the following one must have an offset greater or equal than j to prevent overlapping, and not with value $j+1$, which would induce an isolated hole: the ≤ 1 inequality guarantees that none of these conflicting placings are chosen at the same time. Row (2) specifically handles the case in which $j = H$, which does not allow the offset of the following car to take value $j+1$. (3) states that it is impossible that an isolated hole is placed before or after any car; it fixes to zero $2C$ variables, which can potentially be removed for the model. Here, they are not deleted for an easier interpretability of the indexes; they were actually removed in the final implementation of the exact solver.
The above considerations can be summarized in the following ILP model:

$$\begin{cases} min \sum_{i=0}^{C-1} \sum_{j=0}^{H} c_{ij} x_{ij}\ s.t. \\ \sum_{j=0}^{H} x_{ij} = 1,\ \forall i \mid 0 \leq i < C \\ x_{ij} + \sum_{k=0}^{j-1} x_{i+1,k} + x_{i+1,j+1} \leq 1,\ \forall i \mid 0 \leq i < C-1,\ \forall j \mid 0 \leq j < H \\ x_{iH} + \sum_{k=0}^{H-1} x_{i+1,k} \leq 1,\ \forall i \mid 0 \leq i < C-1 \\ \sum_{i=0}^{C-1} x_{i,1} + x_{i,H-1} = 0 \\ x_{ij} \in \{0,1\},\ \forall i \mid 0 \leq i < C,\ \forall j \mid 0 \leq j \leq H \end{cases}$$

This model involves then $(H+1)C$ binary variables and $C + 1 + (H+1)(C-1)$ constraints. Although these inequalities don't come in exponential number, most of them are often not useful: in fact, when the acquisition of transit data works well, best car placings according to this cost model are indeed the correct, real ones, which trivially satisfy all the inequalities. Then, in order to improve solving efficiency, it is possible to leave the $(H+1)(C-1)$ *no overlappings, no isolated holes* sets of constraints (1) and (2) out of the original model, using them as lazy constraints and so adding them just in case they turn out to be violated. The problem might be then solved iteratively: if the output is unfeasible in a certain step, the model is updated and solved again; the execution stops whenever a feasible matching is met.
The resulting solving strategy is summarized in algorithm 5, where `solve()` is an implementation-dependent black-box solver for (binary) integer linear problems.

Algorithm 5: Exact solver for tag/axles matching

Input: Matching alg. input.
Output: Matching solution with minimal cost.
$model \leftarrow$ /* basic model without (1) and (2)*/
$continue \leftarrow$ True
while *continue* **do**
  $continue \leftarrow$ False
  $x^* \leftarrow$ solve($model$)
  for $i \in \{0...C\text{-}1\}$ **do**
    for $j \in \{0...H\text{-}1\}$ **do**
      if $x_{ij} + \sum_{k=0}^{j-1} x_{i+1,k} + x_{i+1,j+1} > 1$ **then**
        $model \leftarrow model \cup \{x_{ij} + \sum_{k=0}^{j-1} x_{i+1,k} + x_{i+1,j+1} \leq 1\}$
        $continue \leftarrow$ True
      end
    end
    if $x_{iH} + \sum_{k=0}^{H-1} x_{i+1,k} > 1$ **then**
      $model \leftarrow model \cup \{x_{iH} + \sum_{k=0}^{H-1} x_{i+1,k} \leq 1\}$
      $continue \leftarrow$ True
    end
  end
end
/* here, x^* is feasible and optimal */

4.3 Heuristic Strategies

A different option is to sacrifice the optimality proof to obtain a solution of reasonably good quality in a very short computing time. In this case, it is necessary to implement a general heuristic optimization algorithm to make it suitable for this problem, or to develop a more specific one following some problem-dependent intuitions.
This section is therefore intended to present the strategies which have been developed for this matching problem: one consists of a greedy policy based on locally optimal choices, coming along with a randomized version, and the other one is a genetic algorithm.

4.3.1 Greedy Algorithm

With respect to the general description of greedy algorithms provided in the preliminaries chapter, E is in this case the set of all feasible single car placings; each of them has a cost. Then, a subset T of E is an *independent set* if it corresponds to a set of placings which is feasible, namely, which has no car repetitions, no overlappings and no isolated holes, and which preserves car ordering based on detection times (this is what `isFeasible()` checks in pseudocode). Finally, S is the set of all feasible sets of car placings. Greedy algorithm for car/axles matching can then be summarized as in 6.

As previously mentioned, car cost can be computed in $O(N)$. Actually it is also $\Omega(N)$, and then $\Theta(N)$: to prove this it is enough to think of a possible transit in which each

Algorithm 6: Greedy solver for tag/axles matching

Input: Matching alg. input.
Output: Feasible matching solution (no optimality proof).

$E \leftarrow \varnothing$
for $c \in \{0...C\text{-}1\}$ **do**
  $minstart_c \leftarrow \sum_{i=0}^{c-1} l_i$
  for $j \in \{0...H\}$ **do**
    $E \leftarrow E \cup \{(c,\ minstart_c + i)\}$
  end
end
$E_s \leftarrow$ /*E sorted by ascending $z_{car}(e)$*/
$sol \leftarrow \varnothing$
for $i \in \{0...|E_s| - 1\}$ **do**
  if $|sol| == C$ **then**
    break
  end
  if *isFeasible*$(sol \cup \{E_s[i]\})$ **then**
    $sol \leftarrow sol \cup \{E_s[i]\}$
  end
end

car cost is computed by scanning at least $N/2$ axles. Furthermore, there are exactly $C(H+1)$ possible car placings; then, the computational complexity associated with the process of building and sorting E by car placing cost is $\Theta(CN(H+1))$, or $\Theta(CN^2)$ as $H \leq N$.
The last for loop runs for $\leq C(H+1)$ iterations, and the feasibility check requires to compare current placing with $< C$ previous placing which are already in the solution, so it runs in $O(C^2(H+1))$, being also $O(CN(H+1))$ as $C < N$.
The overall time complexity associated with the algorithm is therefore $\Theta(CN(H+1))$.

4.3.2 GRASP Algorithm

As previously explained, GRASP is an iterative variation of greedy algorithms, involving some randomization to perform some sub-optimal local choices. In this specific application, randomizing element choice means not always picking the car placing with minimal cost: one might choose to pick the second best one with a certain probability p_{sec}. GRASP strategy for this matching problem has then the scheme sketched in alg. 7.

The tests which have been performed to compare the solvers suggest that, in this context, GRASP randomization does not lead to significant performance improvements with respect to plain greedy strategy. Therefore, the deterministic version of greedy algorithm will be the one subject to further performance analysis.

Algorithm 7: GRASP solver for tag/axles matching

Input: Matching alg. input.
Output: Feasible matching solution (no optimality proof).

```
sol* ← ∅
while /* time lim. not expired*/ do
    E_s ← /* as in greedy */
    sol ← ∅
    for i ∈ {0...|E_s| − 1} do
        // allow randomization if more than one element is available
        if i ≠ |E_s| − 1 ∧ U([0, 1]) < p_sec then
            swap(E_s[i], E_s[i + 1])
        end
        if isFeasible(sol ∪ {E_s[i]}) then
            sol ← sol ∪ {E_s[i]}
        end
    end
    if z(sol) < z(sol*) then
        sol* ← sol
    end
end
```

4.3.3 Genetic Algorithm

In every specific implementation of the genetic algorithm, its key features must be defined in a more precise way. In this case, an *individual* is a feasible matching choice, and then the *population* is a subset of the solution space. *Crossovers* are done by placing the first half of the cars in the positions given by a parent, and the other half in the ones given by the other parent. There can obviously be some conflicts (i.e. overlappings), which can be solved by translating central cars towards the two ends, with as less moves as possible. Finally, *mutations* can be implemented by random car translations in an individual.
GA implementation is summarized in algorithm 8.

In this GA:

- $ubFeasSeq$ is the total number of solutions, possibly with isolated holes ($C'_{H+1,C}$);
- $popSize$ is equal to $ubFeasSeq/20$, bounded if needed in $[10,\ 1000]$;
- $nChildren$ is half of $popSize$;
- $maxMutations$, $nRefinements$ are equal to $popSize/10$;
- `shake()` is a function performing a random translation of cars; in other words, some holes are moved to a different position, namely, between different cars;

- `refine()` explores the neighborhood of a given feasible solution by implementing subsequent translations, returning the solution with minimal cost which has been met in this process.

The tests which have been performed to understand the performances of the implemented algorithms suggest that, in this context, the flaws of the GA overcome its advantages.
In this case, for example, it is difficult to set a short and strict time limit, because its expiration is checked at the end of a generation, and handling it requires a relatively large amount of time.
Consequently, even with a more efficient implementation, the GA would probably be competitive neither with the greedy algorithm nor with the exact solver in terms of running times. In fact, in those cases they depend only on some features of the instance, and they are in practice usually limited to fractions of a second.
Another note is about the effectiveness: while in few transits the GA succeeds in reaching optimality even with a short time limit, in other cases the quality of the output turns out to be very unsatisfactory.
For these reasons, although a lot of improvements and parameter tuning could be carried out on the algorithm described above, no further efforts have been put on these activities. In the following chapters, the greedy algorithm will be preferred as a heuristic solver because it better catches the specific features of the problem, resulting in better effectiveness and efficiency.

Algorithm 8: Genetic algorithm for tag/axles matching

Input: Matching alg. input.
Output: Feasible matching solution (no optimality proof).

$population \leftarrow$ /* $popSize$ random feasible solutions */
while /* *time lim. not expired* */ **do**
  // mutations and refinements
  for $i \in \{0 \ldots maxMutations - 1\}$ **do**
    $r \leftarrow$ /* random integer in [0, $popSize - 1$] */
    if $z(population[r]) > min_j\ z(population[j])$ **then**
      $population[r] \leftarrow$ shake($population[r]$)
    end
  end
  for $i \in \{0 \ldots nRefinements - 1\}$ **do**
    $r \leftarrow$ /* random integer in [0, $popSize - 1$] */
    $population[r] \leftarrow$ refine($population[r]$)
  end
  // children generation
  $children \leftarrow \emptyset$
  for $i \in \{0 \ldots nChildren - 1\}$ **do**
    // parents chosen with a prob. increasing with cost decrease
    while *True* **do**
      $p_1 \leftarrow$ /* random integer in [0, $popSize - 1$] */
      $ratio \leftarrow \frac{max_j z(population[j]) - z(population[p_1]) + \epsilon}{max_j z(population[j]) - min_j z(population[j] + \epsilon}$
      if *U([0,1])* $< ratio$ **then**
        break
      end
    end
    /* same for p_2 */
    $children \leftarrow children \cup \{\text{generateChild}(population[p_1],\ population[p_2])\}$
    // replace old sols. with children, with larger prob. for sols. with larger cost
    $rep \leftarrow \emptyset$
    while *rep.length* $< nChildren$ **do**
      $r \leftarrow$ /* random integer in [0, $popSize - 1$] */
      if $r \in rep$ **then**
        continue
      end
      $repProb \leftarrow \frac{z(population[r]) - min_j z(population[j]) + \epsilon}{max_j z(population[j]) - min_j z(population[j]) + \epsilon}$
      if *U([0,1])* $< repProb$ **then**
        $rep \leftarrow rep \cup \{r\}$
      end
    end
    $i \leftarrow 0$
    for $ind \in rep$ **do**
      $population[ind] \leftarrow children[i]$
      $i \leftarrow i + 1$
    end
  end
end
$sol \leftarrow population[argmin_j\ z(population[j])]$ // final sol.: best individual alive

Chapter 5

Performance Analysis and Comparison

The goal of this chapter is to analyze the performances of the different solvers whose implementation has been previously presented. This analysis considers the greedy algorithm as a heuristic strategy, comparing it to the solution based on the ILP model. Two separated aspects are considered:

- the effectiveness, evaluating the quality of the output of the solvers;
- the efficiency, comparing the running times and verifying their compliance with the requirements.

The following analysis is based on the results of tests carried out on a test bed of 292 train transits, which have been monitored in 3 different deployments of the measuring system.
The table below summarizes some simple statistical insights on these instances. It is worth to remember that, in the context of a transit, N is the number of detected axles, C the number of detected cars (after applying the filtering procedure) and H the number of holes (i.e. the unlabeled axles in the solution).

	N	C	H
max	458	110	206
avg	92	21	14

In order to provide a more detailed description of these features, figure 5.1 shows the distributions of N, C and H in the test bed. It can be observed that they are very far from being uniform, with some clear peaks: this is due to the presence of multiple transits corresponding to the same train, which is a common practice when testing a system, and to the fact that a lot of trains follow few typical fixed structures.

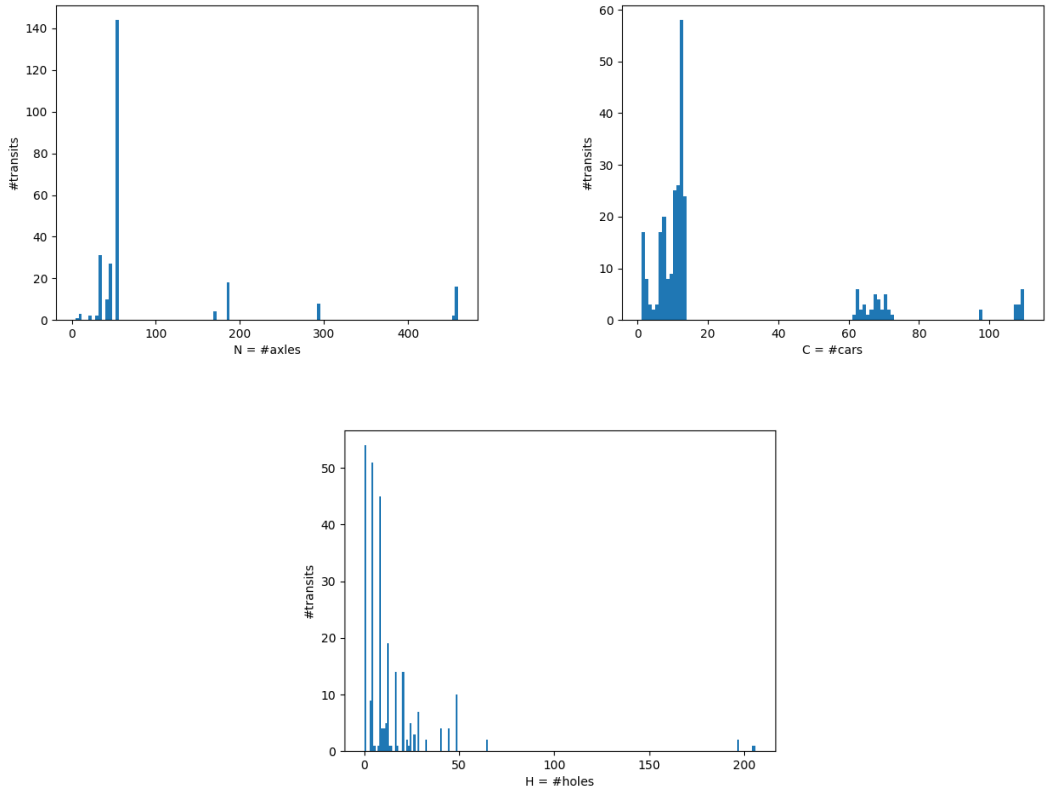

FIGURE 5.1: Distributions of N, C, H in test bed.

Furthermore, it is worth to know that 268 out of 292 transits have produced a non-empty output (in the other ones, all tags were filtered out, or they were not associated to any entry in the database). 214 out of these 268 transits have $H > 0$, which means that some tags or their readings were missing in about 80% of the inputs.

5.1 Effectiveness

It is possible to make different considerations about the quality of a solution. This section is intended to better analyze this topic.

5.1.1 Optimality of Greedy Solutions

The first consideration that can be done in terms of solving effectiveness is about the greedy solver. Remembering that in principle it does not provide any optimality guarantees, the question is, how often does it succeed in reaching the minimal-cost solution? The tests that have been carried out answer that it fails only in 4 out of the 214 non-trivial transits in test bed. In fact, in those cases it does not output the same labeling

choice of the exact solver, whose matching output comes instead with proof of optimality. In other words, the greedy solver provides the solution which minimizes the cost function in about 98,1% of the cases.
These results suggest that the developed greedy strategy is able to properly catch the nature of the problem. In fact, it fails to provide the optimal solution only in few slow-speed or stopping transits, in which the cost function, which does not depend on the solving algorithm, becomes a less reliable evaluation of solution quality; this is actually the next topic to be discussed.

5.1.2 Cost and Solution Quality

A more difficult task is to evaluate whether a certain matching solution (optimal or not) succeeds in representing the real configuration of the train corresponding to the analyzed transit. In fact, given the fact that trains might have a dynamic composition and that they often contain identical cars, it can be hard to infer the correct sequence of cars without manually registering it, if there are some *holes* in the transit instance (namely, $H > 0$).

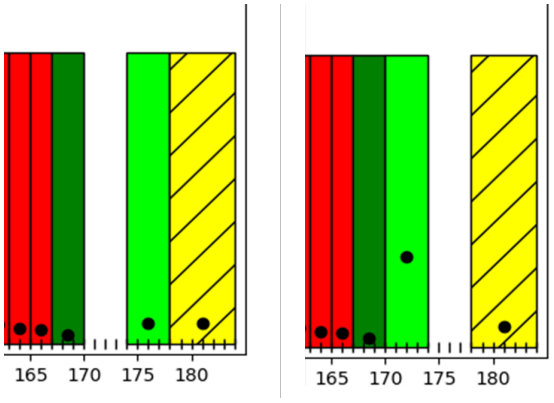

FIGURE 5.2: Same portion of different feasible outputs for the same transit.

For example, figure 5.2 represents the same portion of two different feasible matching solutions for the same transit. The optimal one in terms of cost is the left one: in fact, it can be observed that the light green car has a lower cost in that case. However, this picture proves that there is no guarantee that the minimal-cost solution actually corresponds to the real composition of the train: in fact, in principle also the right solution can be assumed to be realistic.
Therefore, what can be done having only these input data available is to manually analyze the output of the solvers, trying to understand in a reliable way if they are at least visually acceptable. In fact, while it is quite difficult to ensure that a certain feasible solution is correct, it is definitely easier to exploit some typical car patterns to infer that some matching choices are definitely unrealistic. The examples of figure 5.3 should clarify what this means.

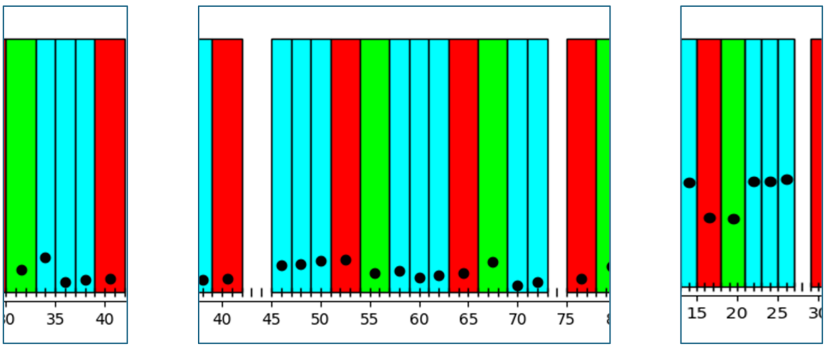

FIGURE 5.3: Car pattern, with complying and non-complying transits.

The first picture shows a reliable pattern that can be found in many transits. In fact, as it is also specified in the rolling stock registry, the five cars of it are tightly associated because they are just a fictitious separation of a certain large articulated well wagon. As a consequence, the second picture shows a realistic train configuration, and thus it can be considered an acceptable output: in fact, its holes can be seen as a placeholder for a green and a light-blue car whose detection was missed for some reason. On the other side, the third picture is a portion of another transit which results in an output that is not acceptable: it does not comply with the pattern described before, having two unexpected unlabeled axles which break it.
These observations lead to a further effectiveness question: are optimal solutions always at least realistic? By visually inspecting the outputs of runs on test bed, in 9 transits the minimal-cost solution provided by the exact solver has proven not to be *visually acceptable*. In other words, the optimal solution can be assumed to possibly represent the real train transit in 205 out of 214 outputs containing some holes, which is about 95,8%. It is therefore worth to better visualize this result to have a clearer understanding of which features make an output unrealistic.

5.1.2.1 Possible Feature Choices for Quality Analysis

A first possible feature choice is used in figure 5.4, which plots the costs of optimal solutions against the average train speeds (weighted average of inter-axle speeds, where inter-axle times are the weights) for all transits, with the ones inducing an unrealistic output marked with a red cross. This plot highlights a behaviour that was anticipated when discussing the sources of uncertainty in the computation of solution cost: very slow or, even worse, stopping trains make $v_{j,j+1}$ as computed above a less reliable measure of the real average inter-axle speed, inducing larger costs also in optimal solutions.

For example, the output of the exact solver in figure 5.5 refers to a transit in which the train stopped while the reversed (hatched) and dark green car was over the wheel system. Although the cost of that car is limited by the workaround which provides an

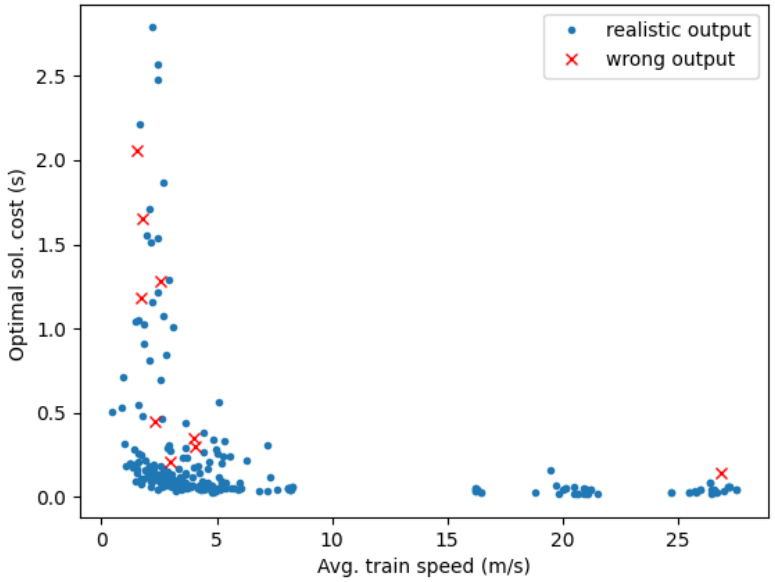

FIGURE 5.4: Costs against average train speeds for all optimal solutions.

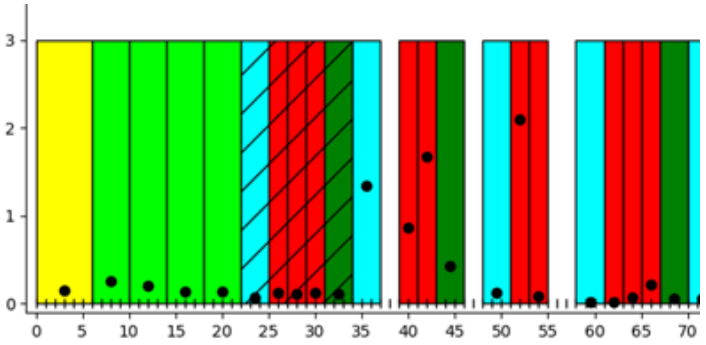

FIGURE 5.5: Portion of optimal solution for a stopping transit.

upper bound in inter-axle speed computation, the large costs of nearby cars offer a visual representation of the uncertainties arising in these problematic situations.

Getting back to figure 5.4 it is clear that, under this speed/cost feature representation of transits, it is impossible to linearly separate wrong and realistic outputs; it can only be observed that unrealistic ones are associated to a cost which tends to be larger than the correct ones at similar speeds.

Figure 5.6 provides a different point of view on the same data, having on the x-axis the ratio between average train speed computed from inter-axle speeds and the same measure, but computed as:

$$\frac{(N-1)\sum_{0\leq c<C}\sum_{0\leq i<l_c-1} d_i}{\sum_{0\leq c<C}(l_c-2)}\frac{1}{t_{N-1}-t_0}$$

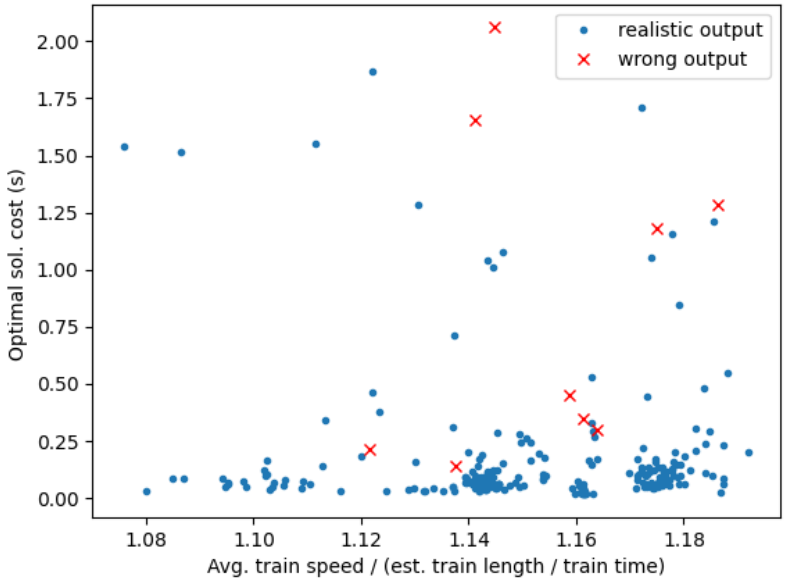

FIGURE 5.6: Costs against speed estimation ratio.

which is the fraction of train length estimated from the available inter-axle distances over the time elapsed between the first and the last axle detections. It is worth to note that train length is exactly the distance the train has run between these two moments; only outputs with at least three cars were considered for this plot to make length estimation more reliable.
The values of this ratio are right-shifted with respect to 1.0: in fact, average inter-axle speed becomes an overestimate of real train speed in case of slow transits, and it is also possible that the unknown inter-car distances induce an underestimation of train length. The underlying idea behind this feature choice is that a ratio which is farther from 1.0 should correspond to a more problematic transit, having too large estimates of inter-axle speeds, but no clear correlation is highlighted between a high ratio and wrong outputs; again, there is no linear separability.

This desirable property was not found in any of the different feature representations which have been explored; however, one possible (and quite predictable) suggestion that could be grasped is that an high cost could be a warning for the user about solution quality. It is then worth to better analyze this measure.

5.1.2.2 Cost Distribution and Quality Warnings

The histogram of figure 5.7 shows the distribution of transits having an output with cost $< 0,4\ s$ (87% of the test bed), since this is the region in which they are more concentrated. The red lines represent the costs of the 4 wrong outputs falling inside this range: it can be observed that they are all at the right of the peak in terms of transit

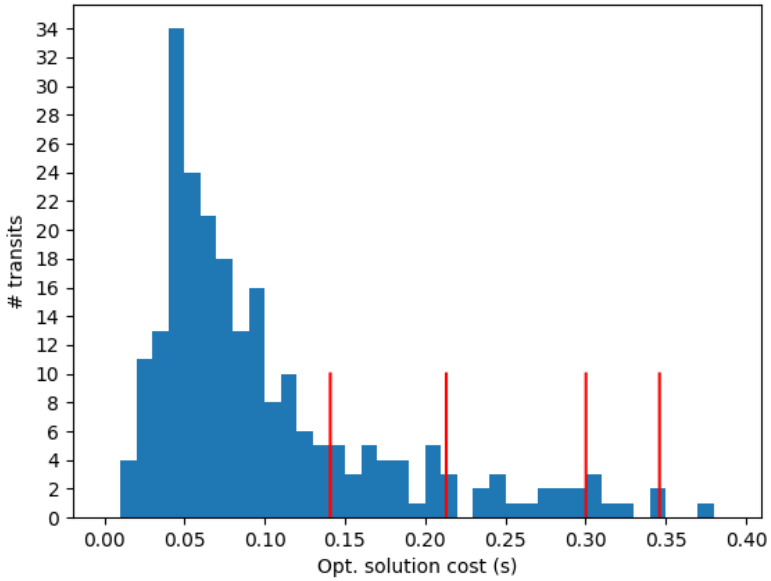

FIGURE 5.7: Cost distribution.

concentration.

As a consequence, a possible idea is to generate a warning for the user indicating if the cost of output solution enters a (possibly customized) *unsafe* cost region, intuitively leaving the *safe* region around the peak.

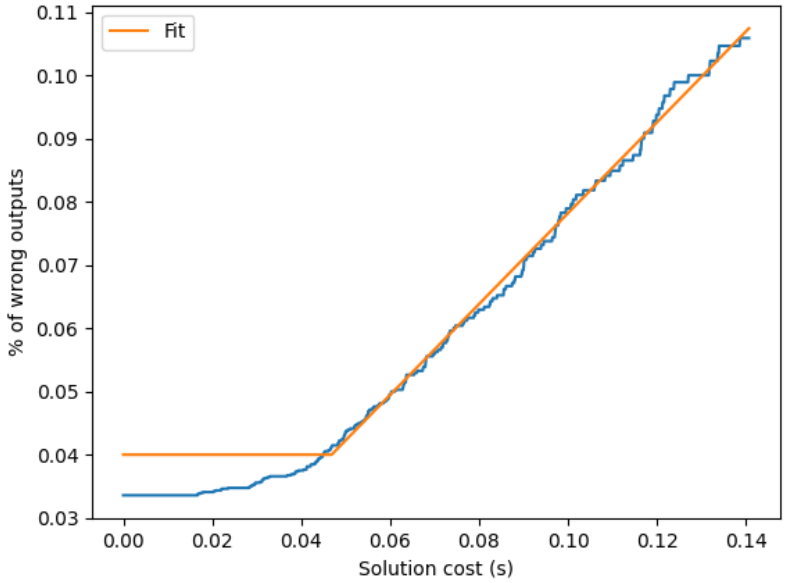

FIGURE 5.8: Perc. of wrong outputs between the ones with $z \geq x$, plus linear fit.

Figure 5.8 shows a further investigation on this aspect: x represents solution cost, in a range going from 0 to the minimum cost between wrong outputs, while the measure

mapped to y is the percentage of wrong outputs between the ones having a cost $\geq x$. It might be observed that this value is inversely proportional to the total number of transits with cost $\geq x$, since all wrong outputs are at the right of the considered domain for x, and their cardinality is then constant inside it:

$$y = \frac{\#wrong\ solutions}{\#solutions \mid z_{sol} > x}$$

This measure can be interpreted as an indicator of the *closeness* of a solution having cost x to the region containing wrong outputs. The orange plot is a linear fit of the data, with a slightly conservative lower bound imposed at 4% to avoid the saturation on the left side, which is caused by the aforementioned peak in transits concentration.
A possible warning mechanism can be then based on the bounded linear fit function:

$$f(z) = max(0.04, mx + q) \text{ with } m = 0.71828826,\ q = 0.00635425$$

In fact, given a warning level $L \in [0.04, 0.1]$, the user can be informed if the provided solution has a cost z with $f(z) > L$, entering then a region in which a percentage of $f(z)$ outputs are wrong.
A more user-friendly policy could be to have $L_{user} \in \{0, 1, 2, 3, 4, 5\}$, internally mapping these integer values to decreasing percentage thresholds. For example, $L_{user} = 5 \Rightarrow L = 0.04$, because in this case the user requires the maximum caution in terms of output quality, and therefore a warning should be generated after the lowest possible threshold on z_{sol}.
Solutions with a cost out of the range of the plot (so after the first wrong output) could be warned by default.

5.2 Efficiency

Another point of view on the performances of the proposed solvers is the one of efficiency. Considering the final C# implementations of the algorithms, it is then worth to analyze and compare their running times on the instances in test bed.
An efficiency comparison between an heuristic and an exact strategy might be in general not very meaningful because of their different nature, but in this specific case the similar performances in terms of cost minimization suggest that it could be worth to analyze this aspect.

In order to graphically represent running times, a choice for the measure to plot them against has to be done.
A possibility could be to use two between N, C and H values as parameters, fixing their values, and to study the behaviour of times with respect to the remaining feature. As a matter of fact, this wouldn't turn out to be an informative analysis given the available dataset, because of the lack of uniformity in the (combined) distribution of the three

measures. An example might clarify this issue. For the pair (N, C), the most frequent combination of values is $(52, 12)$, with 32 transit instances, but there is no variability in H for them. In fact, they all have $H = 4$, because they represent the same high-speed train, or identical ones.

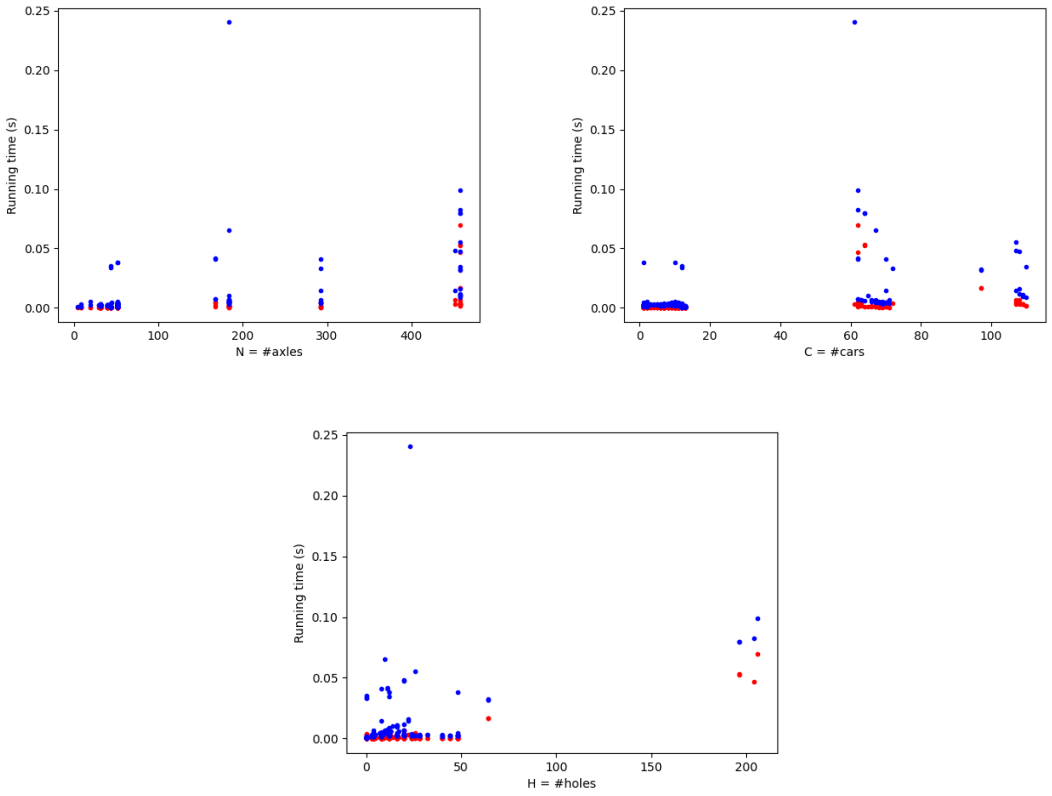

FIGURE 5.9: Running times against N, C, H.

A different possibility is to simply put on the x-axis one between N, C and H, without considering their combined distribution with the other two parameters. The three plots obtained in this way are showed in figure 5.9. A very first conclusion is that both the solvers comply with the minimal efficiency requirements: in fact, except for one outlier, all running times are bounded under around 0,1 s. One can also grasp the suggestion that times increase with the increase of every of the three features but, again, their distributions make the reading of these plots an hard and not very informative operation.

Finally, another idea might be to represent transits in terms of $C(H+1)N$. In fact, it was previously observed that the greedy solver has a time complexity of $\Theta(C(H+1)N)$, and this measure can also be thought as an estimation of instance complexity which better diversifies the transits in test bed. Figure 5.10 plots then the running times of the two solvers on the whole dataset against it: a clearer increasing behaviour can be observed now.

Another possible (and also predictable) observation is that the greedy algorithm has the shortest computing time in every transit. Figure 5.11 plots the ratios between the

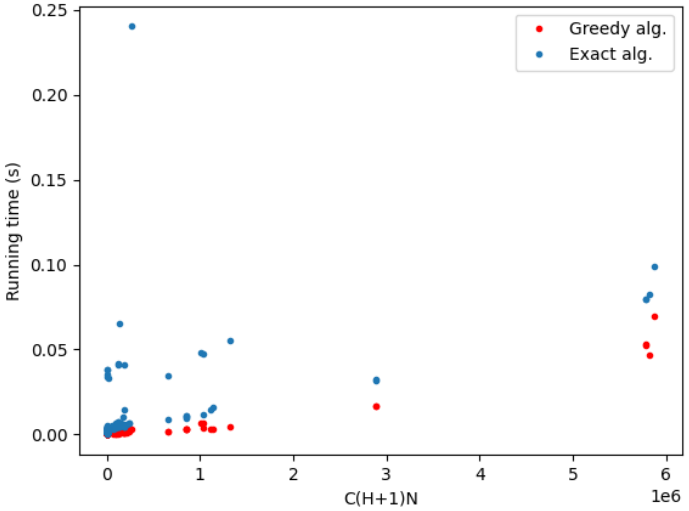

FIGURE 5.10: Running times against C(H+1)N.

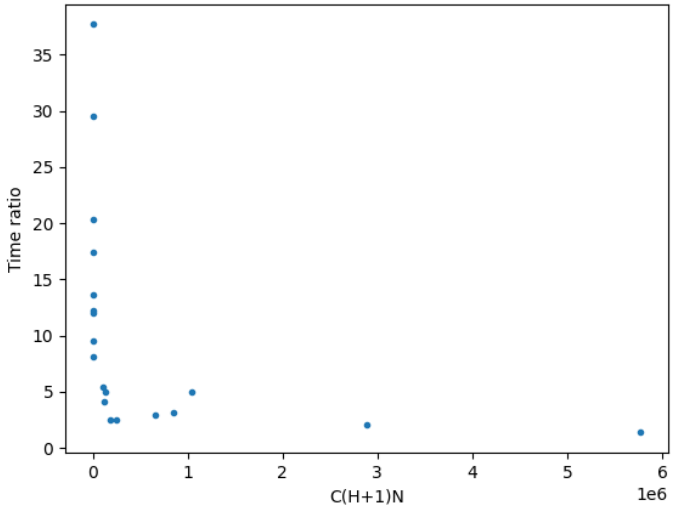

FIGURE 5.11: Running times ratios between exact and greedy solvers.

running times of the two algorithms, averaging on ranges of 1000 units in terms of $C(H+1)N$ in order to have a more readable result; with the same goal, the times of the exact solver have been normalized for this plot with respect to the number of iterations it needed to ensure feasibility with the *lazy constraints* approach. It can be observed that the ratio decreases when the complexity of the instance increases: this can be explained with the fact that the overload of calling an external solver and setting up an ILP model has a larger relative contribution to running time if the problem itself is simpler.

Figure 5.12 offers a different take on the same comparison, this time without any normalization in running times of the exact solver. It is a *performance profile* plot, which is a useful tool to visually compare the performances of different algorithms. In this

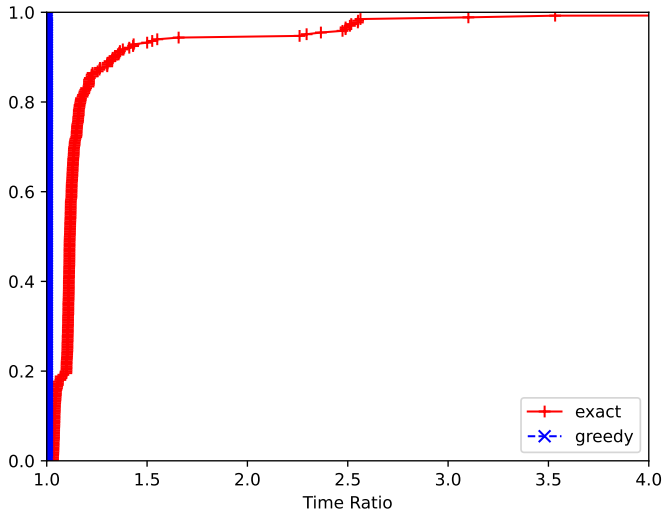

FIGURE 5.12: Performance profile for running times.

case, for each algorithm alg (greedy or exact solver) which runs on test bed instances, it plots:

$$p_{alg}(x) = \frac{\#\text{instances s.t. } \frac{t_{alg,inst}}{t_{best,inst}} \leq x}{\#\text{instances}}$$

where $t_{alg,inst}$ is the running time of alg on instance $inst \in test\ bed$, while $t_{best,inst}$ is the one of the quickest algorithm on that instance. It is worth to note that p_{alg} is therefore a non-decreasing function of x.

For example, $p_{alg}(1.1) = 0.8$ means that algorithm alg runs on 80% of the considered instances in a time which is at most 10% greater than the one of the quickest algorithm on the same instances.

In this case, a shifting term of $0.02s$ was added to both times in the ratio to provide a clearer plot. The blue plot, describing the behaviour of the greedy solver, immediately gets to $p_{greedy} = 1$ since it is the best performer on each instance.

A more interesting observation that might arise here is about the exact solver: with p_{exact} around 0.2, its plot gets quickly farther from the one of the greedy algorithm. The explanation of this behaviour lies in the fact that those 20% transits are the ones with $H = 0$, which are internally handled and solved without actually building the ILP model.

TABLE 5.1: Iterations of exact solver.

#iters.	0	1	2	3-6	>7
#transits	54	190	13	7	4

Finally, table 5.1 summarizes the number of iterations required to solve the ILP models. The 54 transits with 0 iterations are the already mentioned ones with no holes. Leaving them out, 190 out of 214 instances (88,8%) have not required the activation of any lazy

constraints, and so they were solved in one single iteration. This suggests that these inequalities were handled in a meaningful way or, from a different point of view, it shows that for the majority of the instances it is enough to place each car in the best position of its potential range of axles to obtain the best feasible solution. This latter one is also the reason why the greedy algorithm has turned out to be very effective as an optimization strategy.

Chapter 6

Conclusions

The overall goal of this work was to develop a both efficient and effective matching strategy to assign axles to the correct cars. It is now worth to summarize the results that has been presented in this paper.

Different problems were known in advance, and they have also been discussed here. It is worth to mention at least the issues of slow and stopping trains, the detection of spurious tags and the time uncertainty associated with tag readings.
For what concerns the spurious tags, the proposed heuristic filtering was triggered just in few transits in test bed. In some of them it was effective, while in some other ones it was observed that plugging in some further minimal rules about car patterns would allow to develop a more effective strategy.
Talking about the baseline uncertainty associated with tag detection, the cost function which has been presented seems to be robust with respect to it.
The major issue remains then the one of slow speeds. As discussed in performance analysis, even the quality of the minimal-cost solution decreases in those situations, and this should at least be reported as a warning to the user, possibly along with a measure of the reliability of the provided matching choice: this is the solution which has been adopted in this work.

As it was discussed, both the implemented solvers comply with the minimal efficiency requirements. Therefore, the exact solver might be preferred considering its stronger mathematical ground, unless any further considerations about security, disk space or more strict efficiency needs suggest to avoid using an external LP solving library. In this case, one might opt for the greedy heuristic solver, or for the use of a specific and internally developed ILP solving tool.

For what concerns possible future improvements of this work, a path that could be followed is to study the typical behaviour of train motion, in order to better estimate the inter-axle average speed.
Another way to improve this project might be to exploit some expected patterns in the

sequence of cars in a train. Although, as explained, trains could be in principle assumed to have a dynamic composition, in some customer-dependent contexts it might be possible to have a more specific prior knowledge on their typical structure.

Bibliography

[1] MERMEC. *Wheel Parameters*. URL: https://www.mermecgroup.com/measurement-br-trains-e-systems/train-monitoring/87/wheel-parameters.php.

[2] Dieter Jungnickel. "The Greedy Algorithm". In: *Graphs, Networks and Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 129–153.

[3] Mauricio G. C. Resende and Thomas A. Feo. "Greedy Randomized Adaptive Search Procedures". In: *Journal of Global Optimization* 6 (1995), pp. 109–133.

[4] José Antonio Parejo et al. "Metaheuristic optimization frameworks: a survey and benchmarking". In: *Soft Computing* 16 (2012), pp. 527–561.

[5] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[6] M. Fischetti. *Introduction to Mathematical Optimization*. M. Fischetti, 2019.

[7] Michael B. Cohen, Yin Tat Lee, and Zhao Song. "Solving linear programs in the current matrix multiplication time". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2019. Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 938–942.

[8] Robin Harris Pearce. "Towards a general formulation of lazy constraints". PhD thesis. University of Queensland, 2019.