UNIVERSITÀ DEGLI STUDI
DI GENOVA

PhD Thesis

Artificial Intelligence for Automated Design of Elevator Systems

MARCO MENAPACE
Universitá degli Studi di Genova
Phd in Computer Science and Systems Engineering - XXXIII cycle

Abstract

Configuration and design of complex products represents a challenge in many application fields. The designer must take into account many different aspects and make decisions typically driven by experience while taking into account performance constraints and costs. Methods and tools for design automation represents a viable solution to such complex decision problems, giving also the possibility to optimize the performance of the final product on particular context-driven aspects. Artificial intelligence (AI) algorithms can help in dealing with complexity and enhance the current tools by supplying solutions in feasible time.

My research is concerned with the development and testing of different artificial intelligence (AI) techniques to automate the design of elevators. Elevator design is a problem with many interesting aspects like the need to deal with a hybrid search state space (continuous and discrete variables) constrained by design requirements and safety regulations. The study, design and integration of AI techniques in this particular application field can provide the end user with design automation tools that output feasible solutions within acceptable computation times.

My research considered AI techniques such as special-purpose heuristic search, genetic algorithms and constraint satisfaction to solve elevator configuration problems. I tested them considering different setups and parts of the whole design process. I have also implemented a tool LIFTCREATE, available as a web application. LIFTCREATE leverages the findings of my research to automate the design of elevators and, to the best of my knowledge, there is currently no similar tool publicly available from either academia or industry that provides the same level of design automation.
Keywords: artificial intelligence, automated design, product configuration, elevator design

Acknowledgments

This research could not be finished without the help and support of so many people. I express my gratitude to each one of you, who ever gave me support, advice.
My warmest thanks go to my supervisor, Prof Armando Tacchella, for his inspiring guidance, patience, valuable advice and encouragement throughout my research.
Thanks are also due to all the staff from AIMS Lab, for the warm support over the years. The long days spent developing and testing solutions became light working with them side by side.
Many thanks to Leopoldo Annunziata that shared all his knowledge in the field of mechanical engineering; his sincere passion for his work driven the development of the LiftCreate tool, and built the basis for a new research.
Finally, I would like to extend my greatest thanks to my family, for your support, blessing and encouragement.

Table of abbreviations

Abbreviation	Description
LC	LIFTCREATE
AI	Artificial Intelligence
GA	Genetic Algorithms
CSP	Constraint Satisfaction Problem
BIM	Building Information Modelling
SMT	Satisfiability Modulo Theories
OMT	Optimization Modulo Theories
CAD	Computer-Aided Design
FEM	Finite Element Analysis
RHE	Roped Hydraulic Elevator
MRL	Machine Room-Less

Contents

Chapter 1

introduction

1.1 Context

Implementing automated design techniques involves at least two major related steps: *product configuration* and *product optimization*. The former step is about computing feasible solutions given a catalog of components, relationships among them and structural constraints. The latter one is about finding among feasible solutions those that optimize given performance indicators. Both product configuration and product optimization enjoy a substantial body of scientific literature which can fuel research on the specific topic of CautoD for elevator systems.

In particular, concerning product configuration, the survey by Linda L. Zhang [1] provides a recent and comprehensive account of past and ongoing research in the field. Using the categories proposed in the survey, the ones that are more pertinent to my project are: *configuration ontology*, *configuration recommendation*, *configuration solving* and *configuration design and system development*. Configuration ontology is about conceptualizing the domain of components and relationships among them in order to store, update and retrieve the correct elements of a design when solving a specific product configuration. In [2] a general purpose ontology is provided which can be customized to elevator systems. A preliminary attempt in this direction is also provided in [3]. Configuration recommendation is about suggesting to the users feasible solutions among several and possibly overwhelming alternatives. For instance, the paper by Tiihonen,Felfernig and Mandl [4] suggest that recommendation technologies like the ones used in e-commerce sites can be integrated in product configuration. I did not consider specific recommendation algorithms for my investigation.

Configuration solving is the most important part of automated design applications and it involves the development of algorithms and methods that can compute feasible solutions based on product ontologies and user choices. As mentioned in [1] there are several techniques proposed in the literature including those based on heuristic search methods, constraint programming and also case based reasoning — see, e.g. [5]. All these techniques have been reviewed and I considered the most promising ones for integration in my prototype. A preliminary study comparing heuristic techniques versus computer intensive (brute force) approaches in solving product configuration is presented in [6]. Another study considering SMT/OMT techniques is presented in [7]. In this thesis I present also some results obtained with GAs, currently under review as a journal submission.

Considering system development, only two publicly available products are endowed with some CautoD functionality targeted to elevator design: LIFTDESIGN[1] from DigiPara® and LIFTMATIC[2] from ApplicativiCAD. Both applications offer libraries of commercial off-the-shelf components wherewith 2D elevator drawings (plan and vertical views) are generated trying to accomodate physical constraints, designers' choices, and customers' requirements. While LIFTDESIGN can also generate 3D models, it consists of "predefined elevator parameters, component structure and elevator logic" which makes the creation of customized solutions rather difficult. Furthermore, LIFTDESIGN does not provide guidance to the designer amidst alternative implementations, but it just provides warning and error messages. LIFTMATIC provides more support for customization and more design automation than LIFTDESIGN, in that it guides the user through various steps of the design by trying to ban alternatives that will almost surely lead to unfeasible designs. The main issue with LIFTMATIC is that it relies on a rather contrived and acronym-laden graphical interface which, together with some maturity issues, severely affects usage by all but the most experienced designers.

1.2 Motivations and Goals

Adautomated design of systems is the process whereby a design of some implement is carried out by computer programs which supports the work of engineers and technicians. At the highest level of automation, the process requires a designer to enter few configuration parameters, guidelines and

[1] https://www.digipara.com/products/liftdesigner/.

[2] http://www.applicativicad.it/ascensori.php.

physical constraints only, and the work of generating feasible designs will rest on computer programs. Automated design differs from "classical" computer-aided design (CAD) in that it is oriented to replace some of the designer's capabilities and not just to support a traditional workflow with computer graphics and storage capabilities. While automated design programs may integrate CAD functionalities, their purpose goes far beyond the replacement of traditional drawing instruments and most often involves the use of advanced techniques from AI. As mentioned in [8], the first scientific report of automated design techniques is the paper by Kamentsky and Liu [9], who created a computer program for designing character-recognition logic circuits satisfying given hardware constraints. In mechanical design — see, e.g., [10] — the term usually refers to tools and techniques that mitigate the effort in exploring alternative solutions for structural implements, and this is the flavour of CautoD that will be considered hereafter. — see, e.g., [11] —

The main question of my research was driven by the search of techniques and solutions to create products able to emulate the decision process made by engineers. The research in this field was historically addressed with the development of expert systems. Previous works highlighted how such systems were able to solve such problems by inferring new solution from a predefined knowledge base. Expert systems had good performances but the real challenge was the definition of the problem encoding: to specify a very complex design process an highly skilled expert of the field considered study must be directly and constantly involved. Considering an engineering problem, many variables and rules must be traversed, with no warranty on the optimality of the solution.

In elevator design context questions like "which components better fit the requirements?" or "what is the best position of that particular part?" pose a challenge for the design teams in many different fields. The main goal of the thesis is to study, design and develop tools able to solve such problems relying on AI techniques for the encoding and elaboration of solution. The candidate algorithms that I have considered are genetic algorithms and constraint-based technologies such as SMT and OMT solvers.

Another research question in my work is about the possibility to support the design process of a particular product as a service via a web application. Digital solutions are currently pervasive in the manufacturing industry. The industry 4.0 field deals with this new process of integration of digital solutions with production systems. A branch of industry 4.0 is strictly correlated with product design process proposed: *Engineer to order* (ETO).

This, deals with all the procedures required to define a project, accordingly to installation constraints and customers needs. New technologies led the software industry to reconsider how the software capabilities are supplied to customers: canonical monolithic solutions deployed on several firm's servers migrated to cloud technologies bringing enhancements under may different aspects. The first impacted aspect is the development process, now dealing with services, that are components implementing the business logic and providing the functionalities to the users. This approach enable the possibility to test single parts in isolation and provide the possibility to update part of the solution without downtimes of the complete application. Also, architectural solutions oriented to service permit the so called horizontal scalability of the computational resources: firms no longer requires to estimate the traffic to size their data center. The established approach is to define rules to launch new instances in case of demand peaks happens. During my research I easily migrated to different deploying solution with the main objective to keep the user experience inside at an high level of interactivity. This objective is motivated by considering the audience of our solutions: our users expected solution in the order of tenth of seconds. The evolution of encoding and solutions proposed took in account also such time constraints, posing a real challenge.

1.3 Structure of the document

The document is structured as follows:

- In chapter 2 we outline the main elements of the elevator design and the various verification steps required to validate a design; we also provide a brief survey of the software technologies on which LIFTCREATE is based
- In chapter 3 we introduce special-purpose heuristic search techniques and compare them with brute force.
- In chapter 4 we introduce genetic algoritms and compare them with heuristic search.
- In chapter 5 we introduce constraint-based techniques and compare them with genetic algorithms and heuristics.
- We provide some final remarks and hints for future research directions in chapter 6.

Chapter 2

Background

2.1 Elevator Design

An elevator is a complex product and the definition of a project of an elevator must pass through different procedures in order to obtain feasible and viable solutions.
An elevator design can be considered as a an hybrid problem where some some parameters are dictated by the commercial available components and some values must be simulated and validated, taking in account the constraints from safety norms. The parameters coming from a selection are then evaluated considering specific calculations to check the final safety of the configuration. The decision variables deals with the positioning and the sizing of the various components.
This daunting process usually is done by expert technicians just before the release of a new family of products, leading the firm to the definition of pre-defined solutions considering the installation parameters.
It's also important to note that elevators are established and mature products: standard solutions are available for many cases but, for each installation all the technical verifications must be carried out in order to certificate the elevator itself.
As in many other standardization process, this brings an optimized flow for the commercial and technical aspects: the solutions are pretty well pre-defined and the correct configuration could be founded directly via simplifications of the complete problem, but at the same time cuts out many best performing solutions in terms of profitability margins.
As in other engineering problems, designers tends to oversize some parts, bringing more expensive solutions, even if cheaper viable solutions are

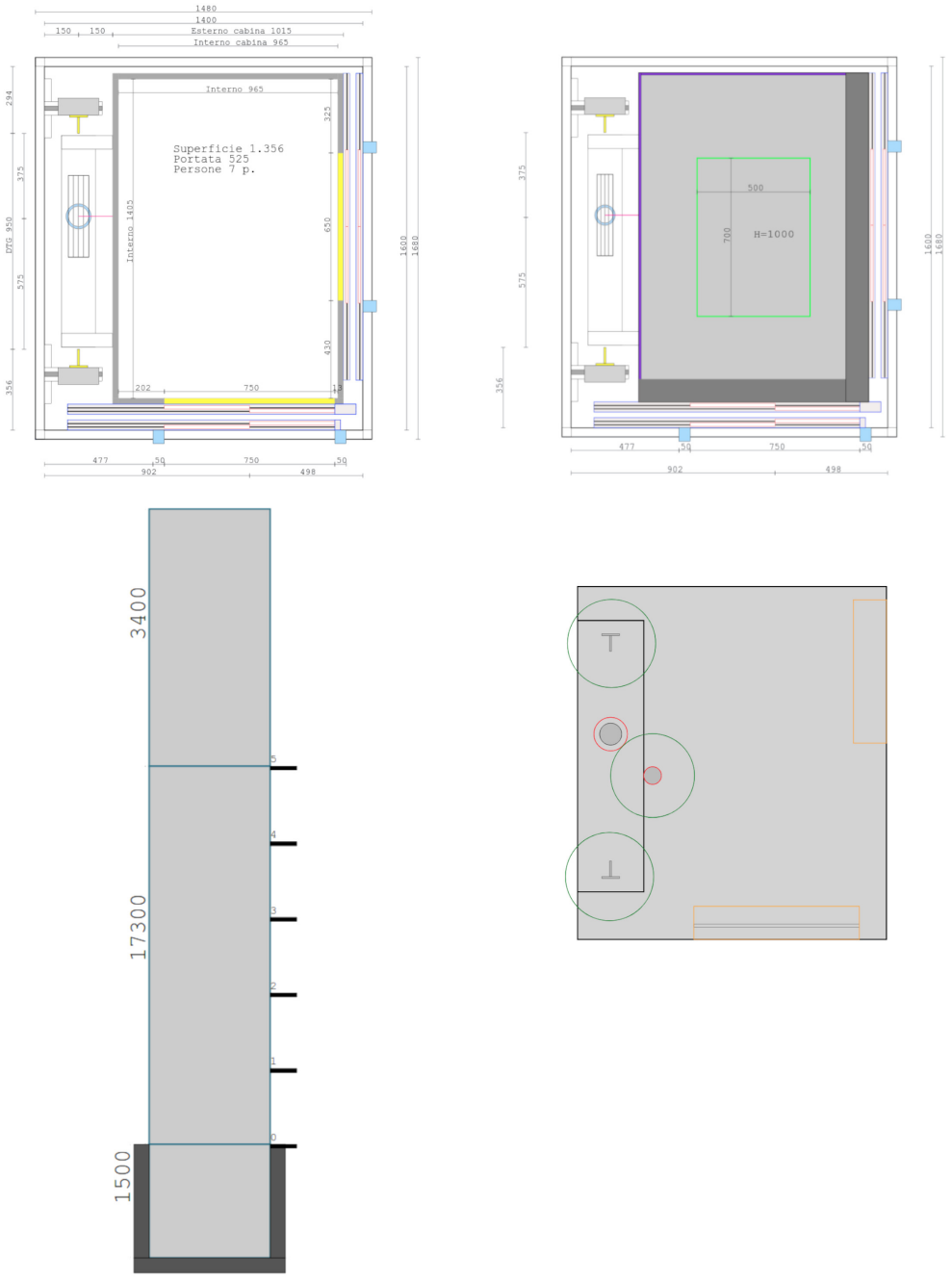

Figure 2.1: Technical drawing reporting the 2D cad drawing of a RHE - Roped Hydraulic Elevator

available, because the cost of verification and certification of the solution is higher. Also, some safety margins dictated by norms, in fact, sets the minimum acceptable values for an already overestimated safety margins. The safety interval imposed by the European Normative impose a wide gap for acceptance; parameters like the maximum load for the single ropes are accepted with a very high level of safety. This means that, starting from the material composition of the ropes, its geometry, alloy and the process used to build it the supplier certificate that product with, among other parameters, the maximum peak load sustainable. Such certifications are obtained by calculation results and standard testing procedures. This value, in order to use that particular ropes in the project, must be at leat 6 times bigger than the maximum load obtained by simulating the worst case scenario, where the cabin, at full load, stops the falls stretching out the ropes.
The input of the entire verification process is the complete design, composed by all details of the final project, and should deal with all the related quantities. The model considered evolved during the development of the solutions, dealing with an increasing number of parameters.

2.1.1 General shaft configuration

The common parts of any implements of elevators is the **cabin**, where the passengers move along the different floors and the **shaft**, the space in which the elevator is installed.
Another important aspect in the process of design regard the door selection. Elevators can be configured with up to 3 different accesses, accordingly to the the opening at the various floor levels. For each access a *car door* must be considered and, for each floor a corresponding *landing door* must be installed.
Doors can be categorized in 3 different types:

- Central opening door, where 2, 4 or 6 panels automatically opens leaving the opening on the center.
- Telescopic opening door, where 1, 2 or 3 panel collapse on one side.
- Folding doors, a particular central opening door with panel folding in half instead of sliding away.

During the design phase the selection of doors must consider the available space in the shaft because doors impose a different depths. Also, designers

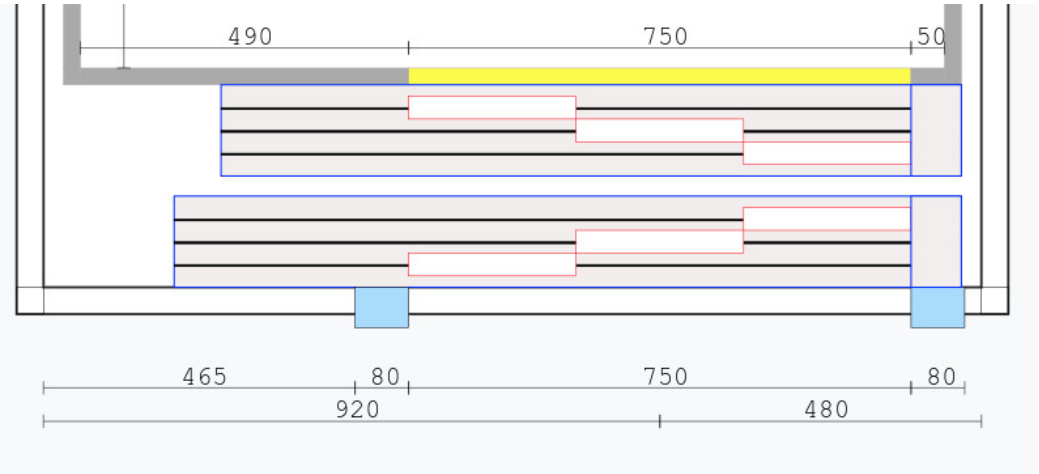

Figure 2.2: Blueprint of a telescopic door pairs. On the upper side the car door is reported

must check that a *safety gap* is available between the doors and other fixed parts present in the shaft.

2.1.2 Hydraulic elevators

The research started considering a particular class of elevators: hydraulic elevators. The study started from this particular type because implements of this type are pretty common. Around 20% of new elevators installations use specific hydraulics circuits in order to move the cabin.
This category can be furthermore split between direct hydraulic elevators and roped hydraulic elevators.
In the first instance the cabin is operated directly by a piston which support the weight of the cabin along the shaft. This particular implements aren't common in the European market because the piston moving the cabin along the shaft, at his maximum extension, must be two time taller than the total travel of the cabin. Such hydraulic components are composed by a fixed part, the cylinder, and a moving part, the rod.
To install direct hydraulic elevators, an hole in the pit of the shaft must drilled down and stabilized witch concrete walls.
In this case The cabin is mounted upon specific steel structure which support the sliding car rails shoe. The car rails mounted along the shaft

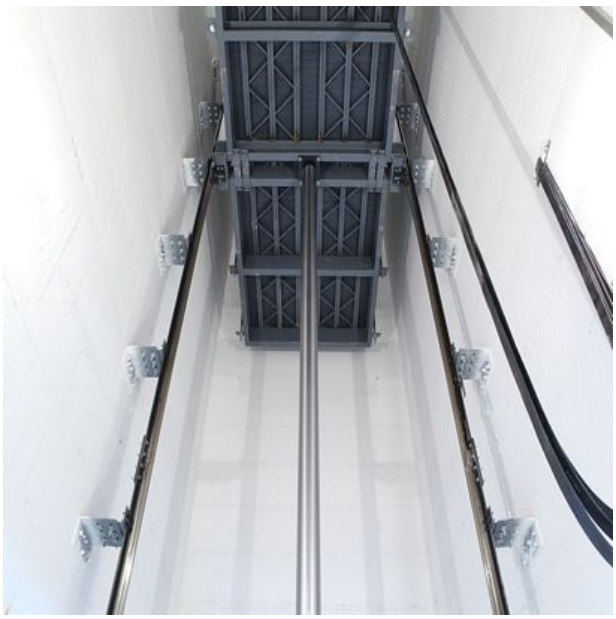

Figure 2.3: A direct hydraulic elevator - note that the weight of the cabin is imposed directly on the piston

Figure 2.4: An example of roped hydraulic elevator - the cabin is moved along the shaft by pushing the ropes by the piston. The main pulley in this picture is hidden by a protection safety carter

walls interacting with the car rails shoe binds the cabin to only vertical movements .
The cost of this installations are considerable and they also pose the concrete risks of drilling through water tables: geographical regions like Italy, with particular orographic conformations, brings a considerable number of artesian spring and drilling through one can lead to serious damages to the structures.
Considering all these peculiarities and issues, direct hydraulic elevators represents viable solutions for low-rise buildings in some selected geographical regions.
A more common solutions that overcome such limitations resides in hydraulic roped elevators: with such implements the cabin is moved along the shaft driven by a piston but in a undirected way. The cabin is hanged upon a set of ropes which pass through a system of pulleys.
On top of the piston resides a pulley which pushes the ropes: this leads to a movement with a ratio of 2:1 of the cabin with respect to the cylinder rod elongation.

The piston is installed on the basement of the shaft, raised by a pillar of proper length, removing the need of drilling through the pavements of the pit.

2.1.3 Piston verification

The process of piston selection is driven by several parameters gathering and consequent checks, namely:

- *Maximum diameter*
- *Critical force*
- *Static pressure*
- *Empty car pressure*
- *Maximum pressure*
- *Cylinder stability*

The first check is the least computing intensive: it deals with the the geometrical maximum diameter of the piston and it compares it to the maximum allowed by the car frame structure.

The **critical force** is calculated considering a parameter λ computed as follows:

$$\frac{Lf}{i}$$

Considering the value of λ, two different formulae are used to calculate the critical forces. If λ has a value greater or equals to 100:

$$Fcr = \frac{((\pi^2) \cdot E) \cdot mip)}{(2 \cdot Lf^2)}$$

in other case:

$$Fcr = ((\frac{An}{2}) \cdot (210)) \cdot (\frac{\lambda}{100})^2$$

where:

- E is a constant equals to 206000, which is the elasticity modulus of the particular alloy used for pistons.
- mip is the moment of inertia of the piston.
- i is the radius of gyration, the distribution of cross sectional area in a column around its centroidal axis with the mass of the body.
- Lf is the final length of the piston, obtained considering also geometry of the car frame.

The **Static pressure** is the pressure that the piston system must support during holding the cabin in fixed position along the shaft.
The first parameter to consider is the *load on piston* (loadP) obtained as follow:

$$loadP = gn \cdot (cm \cdot (P + Q)) + Pr + Prh)$$

where:

- gn is the gravity force constant equals to 9.81 m/s^2
- cm is a security coefficient

- P is the total weight of the cabin
- Q is the total payload of the elevator, computed accordingly to the available surface of
- Pr is the weight of the piston rod
- Prh is the weight of the suspended part of the car frame

The static pressure is then obtained using the following formula:

$$StaticP = (loadP/areaP) + (OilDensity \cdot (Lf/100))$$

where:

- $areaP$ is the sectional area of the piston
- $OilDensity$ is the density of the oil used in the hydraulics(with commercially-available oils this value is considered constant and equals to 0.88)

The **Empty car pressure** should be checked using the same formula as $loadP$ but excluding the Q factor from the formula, obtaining $netP$. Then the same formula used for $StaticP$ is applied:

$$StaticP = (netP/areaP) + (OilDensity \cdot (Lf/100))$$

This value should be grater than a conventional value of 1.2 to check that, without payload, the cabin weight is sufficient to overcome the oil resistance plus the dynamic seals friction on the piston. If such constraint is not satisfied, the cabin will not move along the shaft without the added weight of the occupants.
The **Maximum pressure** i obtained by $8 \cdot StaticP$ and it should be checked against the maximum pressure sustainable by flexible pipes used for the installation.
To verify the **Cylinder stability** 2 parameters, namely efm and ulm must be computed and checked that they are less than the thickness of the cylinder and of the rod.

$$efm = ((0.4 \cdot diamP) \cdot \sqrt{(2.3 \cdot 1.7 \cdot (\frac{StaticP}{355})) + 1}$$

and

$$ulm = ((1.3 \cdot \frac{diamP}{2}) - inRayP \cdot 2.3.1.7 \cdot (\frac{StaticP}{355})) + 1$$

Such consideration must be made considering all the possible pistons in order to resolve this part of the design process. Some a-priori knowledge can be used to filter out some solution: for instance for elevator projects with a very high rise the smaller pistons can be filtered out of simulations, because their stability cannot satisfies the requirements. The same filter can be applied to bigger pistons on low rise and low payload, even if the sizing limit on car frame allows for a fast and direct mathematical comparison.
On top of the piston, a specific circular pulley is installed. This mechanical components, interacting with ropes, converts the linear displacement of the piston to a variation of the ropes position. Such ropes are fixed on one side to the shaft wall or to steel beams and, on the other end, to the moving part of the carframe. The variation of the position of the ropes along the shaft impose the movement to the cabin in a ratio of 2:1 so a single meter of movement of the piston let the cabin travel for 2 meters, hence the commercial classification of such installations.
The main pulley has several grooves where the ropes resides. On hydraulic elevator, the dimension of this singular pulley is directly derived from the available space imposed by the car frame. In this context no evaluation must be done on static and dynamic friction of the ropes on the grooves, and, due to the fact that a single pulled is used, no consideration about the pulleys chain must be simulated. Also the weight of the cabin is always imposed to the pulley and there's no active rotation imposed to the pulley, like case where a motor drives it. Problems rises when dealing with other classes of elevators, like the roped elevator or machine room less elevators, and this considerations are introduced in the following.
Hydraulic elevators are then built upon different other several parts:

- *Car rails*
- *Car frame brackets*
- *Car rails shoes*
- *Car frame*

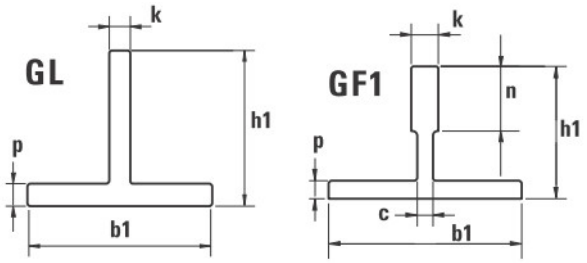

Figure 2.6: Car rails technical drawing showing the differences among the two available sections

2.1.4 Car rails verification

Figure 2.5: An section of a car rails - Note the holes used to bolt each section to the brackets

The *car rails* are "T" shaped steel beams that are installed for the totality of the elevator travel along the elevator hoistway. Such mechanical components are characterized by some important measurements that defines their geometry. Another important parameter is the material used to assemble car rail, which brings different capabilities under load. The selection of these components take in account different constraints regarding the results of different tests, imposed by European normative:

- *Flex strain test*
- *Combined strain test*
- *Peak load strain test*
- *Compression/traction strain test*
- *Combination of compression and flex strain test*
- *Torsion test*

Every single calculations must take in account several parameter derived from the project, specific data from the configuration and rules imposed by normative. First, this process is carried out by procedures that make an initialization different parameters.
Before any further consideration is done, the geometrical parameters of the car rails are considered: car rails comes with 2 different sections, one with a regular "T" shape and one with slimmed out part portion interacting with car rails shoe. The slimmed out section is typically used for bigger car rails, leading to a lower weight of each section, but at the same time keeping the structural strength required.

The simulations take in account three **critical scenarios**:

- **The intervention of the safety gear** where an emergency brake stops the fall of the full loaded cabin in case that all the device sustaining the cabin breaks apart.
- **The cabin movement** where the elevator starts the travel along the different floors.
- **The loading of the cabin** where the weight forces of the full payload are considered applied to the entrance of the elevator.

Each of the above scenario take in account the potential force applied to the rails in the worst case: for each the forces are over-estimated in order to guarantee a proper safety gap.
The most critical scenario, in most cases, is represented by the energy dissipated by the emergency brakes that insist on the car rails.
It's interesting to note that maximum forces obtained during these verifications represents a crucial parameter not only for the elevator installation, but for the entire building stability. Even if the installation is done using an external hoistway, composed of steel beams, the interaction with the construction must be taken in account to avoid critical situations and future structural issues.
Car rails verification is done considering some geometrical parameters obtained from the project: first, an axis system is derived starting from the geometrical center of the car frame along the car rails center. From the origin the x-axis its realized in the direction of the cabin. The y-axis is obtained with a consequent 90 degree rotation.

Using this coordinates system four coordinates (x, y) are then obtained:

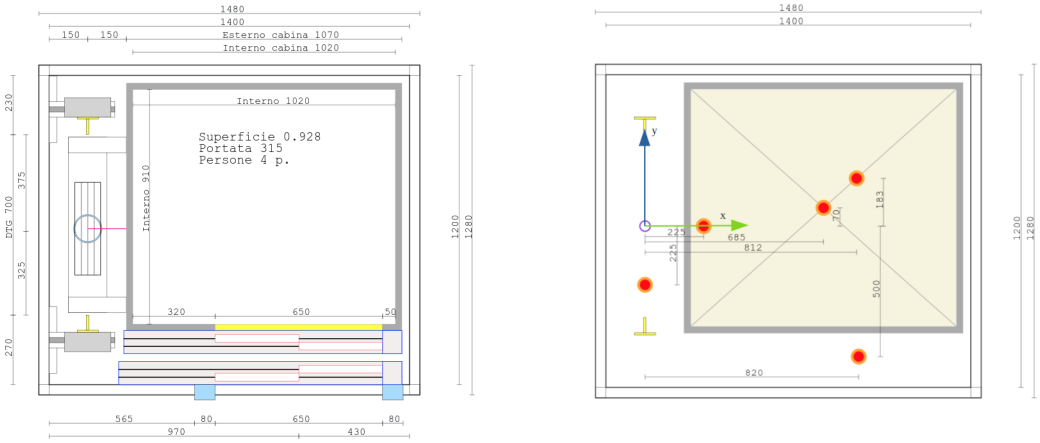

Figure 2.7: CAD 2D drawing, obtained with LC that highlights the car rail verification axys system - The dots on the drawing on the right represent the several COG considered

- P the position of the center of gravity of the cabin.
- S the position of suspension point, which is the point of the pulley where the suspension ropes applies.
- D the position of the the car door center of gravity.
- Q the position of the payload.

All these values are used to calculate the torsional momentum that car rails must sustain, in the previous introduced scenarios.
The **impact coefficient**, $k1$ is derived from a table taking in account the emergency brake used.
n is The number of car rails installed. This values is usually equals to 2, but there are peculiar cases where the car rails used can be up to 4.
Then the **weight of a single rails elements** is obtained with

$$Mg = T \cdot crw$$

where:

- T is the total travel, in meters, of the elevator
- crw is the weight, per meters, of a single car rail

Another crucial parameter imposed by the normative is λ: this value is calculated as:

$$\lambda = dstFix/mRIcr$$

where :

- $dstFix$ is the distance between the car frame brackets, the components that fix the car rails along the hoistway.
- $mRIcr$ is the minimum radius of inertia of the car rails. This parameter is supplied by the car rails builder.

Obtained λ, the process leads to the calculation of Ω. This value is imposed by the normative using two different tables, each one consider different alloy used to build the car rails. This table lookup is done to obtain 3 different coefficients:

- Ω_c
- λ_c
- Add_c

then the value for Ω is obtained as

$$\Omega = \Omega_c \cdot (\lambda^{\lambda_c + Add_c})$$

To conclude the initialization phase, before the actual verification, the following parameters are obtained:

- l is the distance between the brackets
- h is the distance between car rails shoes
- Q is the payload in Kg
- Q_{calc} is the value of Q + 75 Kg, imposed by normative for safety critical computations
- P is the cabin total weight, considering also the car doors weight

The check first case introduced, the intervention of the safety gear, the values of flexion forces have to be computed, dividing the singular contribution among the pre-defined axis system. **FFlexX** and **FFlexY** are obtained by:

$$FFlexX = \left| \frac{k_1 \cdot gn \cdot (Q \cdot Q'_x) + P \cdot P_x + DP_x \cdot P_{weight} + DS_x \cdot DS_{weight} + CF_x \cdot CF_{weight)}}{(n \cdot h)} \right|$$

and

$$FFlexY = |\frac{k_1 \cdot gn \cdot (Q \cdot Q'_y) + P \cdot P_y + DP_y \cdot P_{weight} + DS_y \cdot DS_{weight}}{(\frac{n}{2} \cdot h)}|$$

Where :

- P is the total weight of the cabin plus the weight of the car frame sustaining it
- P'_x and P'_y are the coordinates of the in which the cabin weight is considered, in general an approximation considering the COG of the cabin itself is acceptable
- Q is the total payload
- Q'_x and Q'_y are the coordinates of the in which the payload is considered
- DP_x and DP_y are the coordinates of the COG of the car door
- DS_x and DS_y are the coordinates of the COG of the optional secondary car door
- n is the number of car rails
- h represents the vertical distance between the car rails shoes
- k_1 is the impact coefficient obtained from a dedicated table

It's important to note that the coordinates of the payload, namely Q'_x Q'_y, are obtained by starting from the COG of the cabin and then, move them in both possible direction of $1/8$ of the cabin depth/width. This operation must be done for all the four possible combinations, keeping the maximum value for each $FFlex$ pairs. This is imposed by normative in order to consider the worst case scenario of each case.
Obtained $FFlexX$ and $FFlexY$ a designer is able to actually evaluate the capability of the car rails to resist the mechanical solicitations considered. For the sake of clarity, the complete calculation for other two case studies are not reported here, but they follow the same procedure, with slight changes over static coefficients.

The **Car frame brackets** are dedicated steel structures with the duty to hold the car rails in vertical position. In case of concrete shafts these are attached to walls by using expansion bolts or, under some particular circumstances like very narrow installations, by walling a portion of bracket itself. The last possibility is very sporadic and used for low rise elevator, because the alignment of brackets is very challenging when a part of the

bracket is embedded in concrete: no movement for regulation are possible so the perpendicularity with regards the shaft must be checked for each installation. This bring higher cost an higher risk related to final quality of the the installation. In some critical cases a minimal angle deviation is acceptable but brings an early consume of car rails and car shoes, imposing shorter interval of maintenance.

The canonical installation, using expansion bolts, anyway, gives to the installer the possibility to compensate deformation of the shaft maintaining the car rails perpendicular with respect to the pit floor. The number of brackets that must be installed is a parameter usually imposed by the builder of the car frame, taking in account the height of the installation and imposing a maximum distance between the brackets.
Further consideration can be dictated by the seismic characteristics of the region where the elevator will be installed.

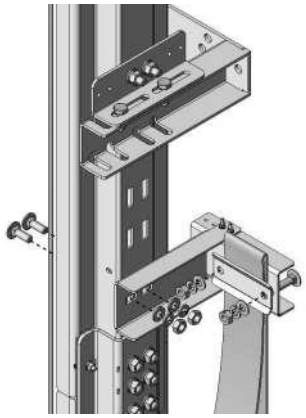

Figure 2.8: Technical drawing of car frame brackets. The image shows also the fitting of the car rails

Car rails shoes are the components that interacts directly with the car rails, keeping the the cabin in position along the shaft.

These components consist of an "U" shaped steel beans with a polymeric serviceable tabs that actually slides on car rails.
On a standard elevator with reasonable rise, the car rails shoes installed are composed of 2 pairs, but in some particular cases, like very fast installations or installations with very high payload capability, the number of such devices can be increased. The distance between each pair of car rails is also an important parameters used to calculate the stability of the system during various verification phase: this value is strictly correlated with the car frame structure and with the selected car rails.

Figure 2.9: An example of polymeric car rail shoe.

2.1.5 Machine room less elevators

A different class of elevator, namely ropes elevator, differs from the hydraulic elevator by imposing the movement to the cabin with an electric motor that drives a ropes and a pulleys system.
There are several classes of roped elevators, but in general, the higher level classification regards the presence or not of a dedicated room for the motor. To supply popular and industrial related solutions, the research focused on **Machine room-less**(MRL) elevator which, in terms of numbers, represent the bigger percentage of new installations. This type of elevator have interesting characteristics in term of simplicity, because an additional room to accommodate the motor is not required: all the components can be installed directly in the hoistway and, thanks to the relatively recent advances in brushless electrical engines, these implements are very compact and energy efficient.

2.1.6 Pulleys and ropes verification

Dealing with ropes, from a design point of view, it's a rather difficult process in this case. In contrast with the process described then dealing with hydraulic elevators, where the verification checks only the maximum load on ropes versus the certificated values from the supplier, with electrical motors moving the ropes and, consequently the cabin, the dynamic and static friction must be simulated on all the pulleys chain components.
Pulleys have several characteristics:

- *Number of ropes supported*
- *Pulley diameter*

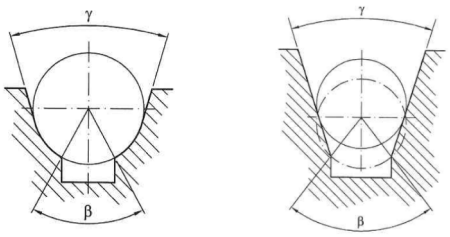

Figure 2.10: Drawing describing the pulley's grooves characteristics angles - the left one is a semi-circular type - on right a "V" shaped grooves is reported

- *Shape of the grooves*
- *Characteristics angles*
- *Wrap angle*

With roped elevators in general, the **dynamic and static friction** are critical and need to be evaluated considering the characteristic of each pulley, which can be different along the elevator.
Dealing with MRL elevator, featuring a simplified pulleys chain, verification is easier in this case, but non trivial.
To check if the current configuration complies with the minimum accepted value imposed by normative, four different cases must be evaluated and, for each case, a parameter,μ, is defined:

- Loading operation $\mu = 0.1$
- Emergency braking on descent $\mu = \frac{0.1}{(1+(\frac{rs}{10})}$
- Emergency braking on ascent $\mu = 0.1$
- Blocked counterweight $\mu = 0.2$

where rs represents the ropes speed in m/s.
In case of pulleys with semi-circular grooves the value of f can be obtained by:

$$f = \mu \cdot \frac{(\cos\frac{\gamma}{2} - \sin\frac{\beta}{2})}{\pi - \beta - \gamma - sin\beta + sin\gamma}$$

where:

- β is the carving angle.

- γ is the characteristic angle.

If the pulleys used are V-shaped f is calculated as:

$$f = \mu \cdot \frac{4(1 - \sin\frac{\beta}{2})}{\pi - \beta - sin\beta}$$

where:

- β is the carving angle.
- γ is the characteristic angle.

The next to obtain an ropes system compliant with European safety norms, requires the evaluation of the ratio between two values: **T1** and **T2**.
These parameters are obtained with the following formulae:

$$T_1 = \frac{(P + Q + M_{CRcar} + M_{Trav})}{r} \cdot (g_n \pm a) + \frac{M_{comp}}{2 \cdot r} \cdot g_n + M_{SRcar} \cdot (g_n \pm a \cdot \frac{r^2 + 2}{3})$$

$$\pm(\frac{i_{PTD} \cdot m_{PTD}}{2 \cdot r} \cdot a) \pm \frac{(m_{DP} \cdot a)}{r} \pm \frac{\sum_{i=1}^{r-1}(m_{Pcar} \cdot i_{Pcar} \cdot a)}{r} \mp \frac{FR_{car}}{r}$$

and

$$T_2 = \frac{M_{cwt} + M_{CRcwt}}{r} \cdot (g_n \mp a) + \frac{M_{comp}}{2 \cdot r} \cdot g_n + M_{SRcwt} \cdot (g_n \mp a \cdot \frac{r^2 + 2}{3})$$

$$\mp(\frac{i_{PTD} \cdot m_{PTD}}{2 \cdot r} \cdot a) \mp \frac{(m_{DP} \cdot a)}{r} \mp \frac{\sum_{i=1}^{r-1}(m_{Pcwt} \cdot i_{Pcwt} \cdot a)}{r} \pm \frac{FR_{cwt}}{r}$$

Where :

- a is the deceleration (positive value) of the cab in meter per second squared
- FR_{car} is the frictional force of the shaft (efficiency of the cab side supports and friction on the guides) in Newton
- FR_{cwt} is the frictional force of the shaft (efficiency of the counterweight side supports and friction on the guides) in Newton

- g_n is the gravity acceleration in meter per second squared
- H is the stroke length in meter
- i_{Pcar} the number of pulleys on the cab side with the same rotation speed v_{pulley} (except deflection pulleys)
- i_{Pcwt} the number of pulleys of the counterweight side with the same rotation speed v_{pulley} (except deflection pulleys)
- i_{PTD} is the number of pulleys of the tensioning device
- m_{DP} is the reduced mass (referred to the car or counterweight) of the car side and / or counterweight side deviation pulleys in kilograms
- m_{Pcar} is the reduced mass (referred to the cab) of the cab side pulleys in kilograms
- m_{Pcwt} is the reduced mass (referred to the counterweight) of the counterweight side pulleys in kilograms
- m_{PTD} is the reduced mass (referred to the cab / counterweight) of a pulley of the tensioning device in kilograms
- M_{comp} is the mass of the tensioning device including the mass of the pulleys in kilograms
- M_{CR} is the effective mass of the ropes / compensation chains in kilograms
- M_{CRcar} is the mass M_{CR} cab side item M_{CRcwt} is the mass M_{CR} counterweight side
- M_{cwt} is the mass of the counterweight including the mass of the pulleys in klograms
- M_{SR} is the effective mass of the suspension ropes in kilograms
- M_{SRcar} is the mass M_{SR} cab side
- M_{SRcwt} is the mass M_{SR} counterweight side
- M_{Trav} is the effective mass of the flexible cables in kilograms
- n_c is the number of the ropes / compensation chains
- n_s is the number of the suspension ropes
- n_t is the number of the flexible cables
- P is the mass of the empty car in kilograms
- Q is the scope in kilograms

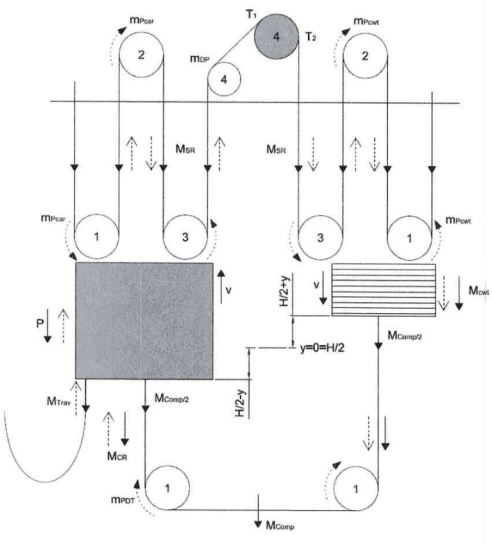

Figure 2.11: The image report a generic schematics used for the generic pulley slip calculation. The rectangle on the left represents the cabin, with its payload. On the right the counterweight.

- T_1 T_2 is the strength of the rope in Newton
- r is the size coefficient
- v_{pulley} is the speed of rotation of the pulley in meter per second

A complete examination of the problem of elevator design represents indeed a challenging and complex task.
During the years of research in this field, thanks to a constant involvement of professional from the industry, participation to technical seminar and fairs, the capabilities to translate this normative constraints to a former software solution were acquired. In the following a set of solution are introduced dealing with the problems introduced above.
For each techniques experimented and validated a different encoding is realized with an incremental approach. The foundation of all these experimental campaign is a set of special purpose heuristics developed and tested for the LiftCreate web application[1].

[1]Application available, after a brief registration at `http://liftcreate.ailift.it`

2.2 Software Technologies

During the research an industry-oriented tool was developed. To achieve this objective a complete stack of technologies, both for back-end and front-end, was used.
This section present a brief introduction to technologies and methodologies used to develop LC.
The tool was developed using mainly Java code on Spring framework, a popular open source framework for the development of web application.
Spring is widely adopted by the Java community for its capabilities of building web application on top of the Java Enterprise Edition platform model. This lead Spring to be recognised as a gold standard by commercial leading firms as strategic importance framework. The first release of the framework was done in 2001 and, since then, it evolved integrating over the years new technologies and development technologies.
Spring historically was one of the first framework to invest a lot of effort in peculiar technologies like Inversion of Control (IOC) Reflection and Aspect Oriented Programming(AOP). A brief introduction to each techniques is reported; IOC is a architectural pattern developer conceived to support developer in the process of simplification and reuse of the code. With the application of IOC the various libraries that compose a software solution are independent.
With a techniques called *injection* the dependencies between different components of the software are handled directly by the framework; to achieve this objective, in contrast with a canonical imperative languages where dependency must be encoded explicitly, the developer must specify a sort of contract between the different parts.
A viable way to specify such dependencies is the *Dependency Injection* which represent a pattern to implement the IOC
Spring relies heavily on this, leading to an optimization in terms of payload of the code, delegating to this solution the burden of initialization logic: with a standard solution based on different services the responsibility of the initialization of the different objects handling their logics is a task that must be done by the programmer. In contrast with IOC and in particular with Dependency Injection the framework handles directly the availability of the service itself, letting the developer focus more on business logics than architecture of the solution.
The main drawback of the application of dependency injection is the risk that for some modules, too many dependencies are injected, leading to

became an anti-pattern when overused or used incorrectly.

For the generation of technical drawings the availability CAD library represented an important requirement and a basic foundation for the develop of automated design techniques.
Many mechanical engineering procedure of design relies heavily on technical drawing, using CAD software.
Typical CAD solution support the designer with the possibility to create 2D drawings and 3D shapes compositions. On top of this firms like AutoCad and PTC offers advanced tools for specific simulations, supporting engineers in the product design phase with a deep knowledge about complex characteristics parameters.
Advanced analysis like Finite Element Method are able to guide the designer across different phase of the product development supplying a fundamental insight about, for instance, structural analysis, heat transfer, aerodynamics and fluid flow.
In the context of elevator design such advanced analysis tools are generally not needed, except for particular case where and engineer must evaluate stability of the entire building after the installation. This study is usually done to check the structure capability under earthquake stresses conditions.
In general all the information required to evaluate the interaction between the elevators and buildings are derived from computation done for the European norm verification.
During our research the eventuality of a further advanced analysis was not considered.
In the elevator design field a relevant novelty was the introducing of BIM technologies, which represents the standard of integration of different part of the building.
Building information modeling (BIM) is a process supported by various tools, technologies and contracts involving the generation and management of digital representations of physical and functional characteristics of places.
Building information models (BIMs) are representations of the building which can be extracted, exchanged or networked to support decision-making. BIM software is used by individuals, businesses and government agencies who plan, design, construct, operate and maintain buildings and diverse physical infrastructures, such as water, refuse, electricity, gas, communication utilities, roads, railways, bridges, ports and tunnels. In particular the survey[12] pose the problem to propose a methodology to apply such concepts to existing building how such problems can be

addressed starting from an existing building and

In our case the decision process and the evaluation of the requirements leads us to develop a 2D cad library. A preliminary experimental campaign in order to evaluate the different technologies were done bringing our consideration to two major candidates solutions, both using client-side scripting: OpenJSCAD and Snap.svg.
OpenSCAD is a free software application for creating solid 3D CAD (computer-aided design) objects. It is a script-only based modeller that uses its own description language; parts can be previewed, but cannot be interactively selected or modified by mouse in the 3D view. An OpenSCAD script specifies geometric primitives (such as spheres, boxes, cylinders, etc.) and defines how they are modified and combined (for instance by intersection, difference, envelope combination and Minkowski sums) to render a 3D model. Snap is a javascript library that permits the generation of SVG, a web standard for vectorial drawings. Considering that a CAD2D library was enough for the production of technical drawings the decision was to use the Snap library.
Also, a precise and complete 3D rendering would require a bigger set parameters that in the designing of an elevator are not considered. Expanding the model with such parameters would also make the computation of solution more difficult, with the risk of losing the interactivity of the application.

The preliminary work focused on acquiring of the current state of the art about topics such cloud technologies, product configuration and optimization.
Regarding cloud technologies a deep study about concepts like software as a service(Saas), containerization of applications and most popular frameworks for web-based developments were done.
The preliminary studies focused mainly on:

- Amazon Web Service(AWS), a cloud computing family of services that gives the possibility, by using proprietary technologies, to obtain industry proven and scalable computing resources. In particular the EC2 computing product were exploited and used to deploy of the early prototypes. A beta version of the tool LIFTCREATEis now deployed on public domain with such technologies.
- Docker, a open source project related to deployment automation. The tools gives the possibility to create specialized virtual machines,

starting from a well defined image, that implements a very fast way to obtain an host machine.

- Spring framework, an open source cloud java framework developed by Pivotal dedicated to web applications.

Chapter 3

Special-purpose heuristic search

3.1 Preliminaries

Elevators can be differentiated in two broad categories, namely traction — also called rope in the following — and hydraulic elevators. In traction elevators, the car is suspended by ropes that are moved via an electrically driven sheave. The opposite end of the ropes is connected to a counterweight. Depending on whether the sheave is driven directly by the electric motor or whether a gearbox is used, these elevators are further differentiated into geared and gearless traction systems. According to [13], geared traction elevators are the most common "legacy" elevator type in Europe, constituting more than two thirds of the European elevator stock. Gearless traction elevators are a comparatively young technology and only constitute about 8% of the total elevator stock. The remaining elevators operate on hydraulics, i.e., they rely on one or more pistons to move the car. Energy is usually provided to the hydraulic fluid by an electrically driven pump, and typically no counterweight is needed to compensate for the weight of the car. Hydraulic elevators (HEs) are often used in low-rise applications and are widely used in new installations in some European countries, including Italy: Their low initial costs, compact footprint and ease of installation makes them the most viable choice for retrofitting old residential buildings, and a cost-effective solution for new ones alike. The choice of HEs as a case study is thus motivated by their popularity, and by the fact that, in spite of their relative low part count, their structure presents already most of the challenges that are to be found in other elevator types.

The components of HEs considered by LIFTCREATE CautoD procedures are shown in Figure 3.1 using an UML class diagram to outline the corresponding

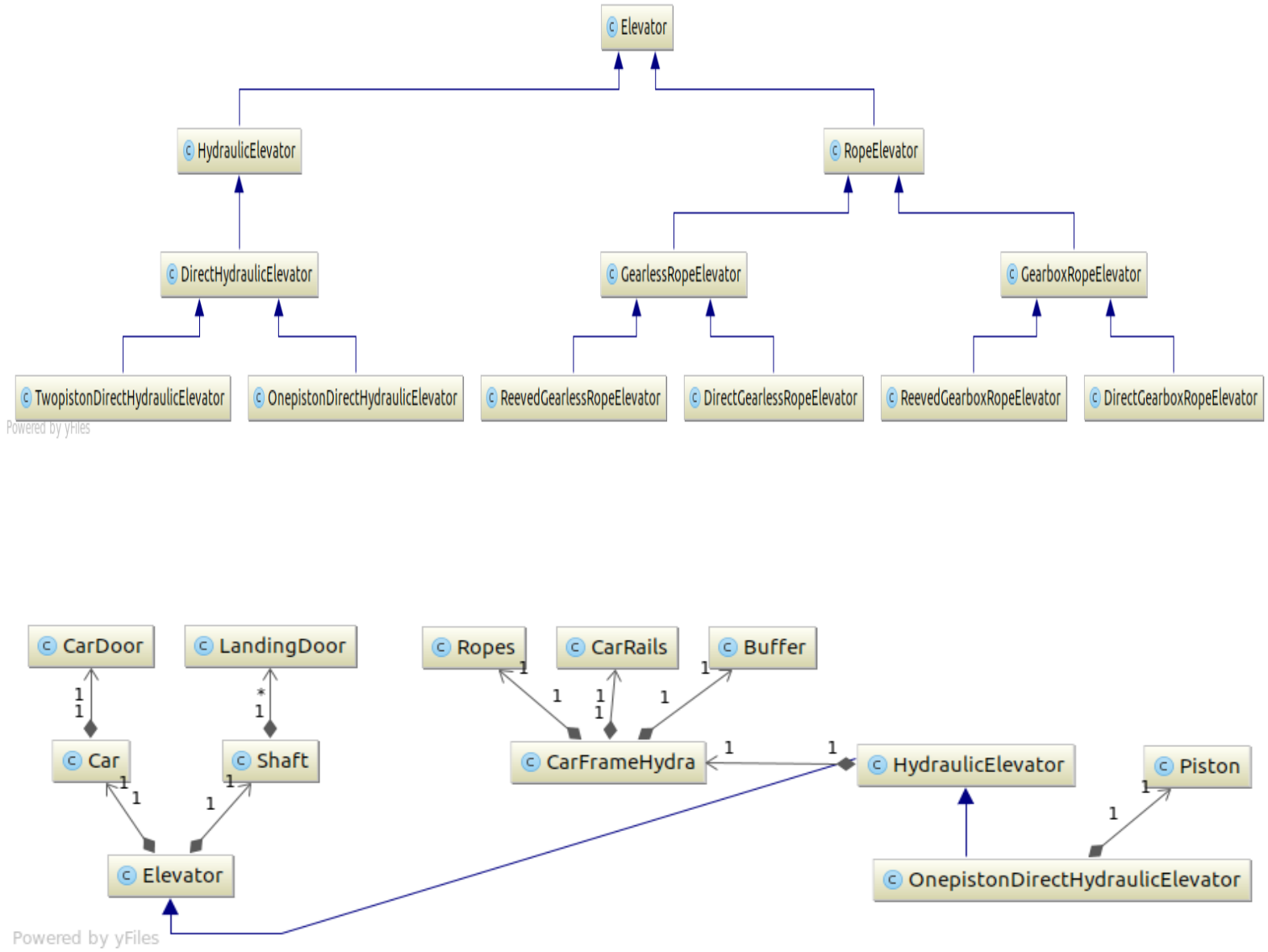

Figure 3.1: Taxonomy of elevator types handled by LIFTCREATE (top)and components of *OnePistonHydraulicElevator* (bottom).
Rectangles represent entities, IS-A relations are denoted by solid arrows, and HAS-A relations are denoted by diamond-based arrows.

part-whole hierarchy. Notice that, in order to manage the space of potential designs components cannot be solely available as drawing elements, like in classical CAD solutions, but they must be handled as first class data inside LIFTCREATE logic. In particular, *OnePistonDirectHydraulicElevator* is both a leaf entity in the taxonomy shown in Figure 3.1 (top), and also the root node of corresponding part-whole hierarchy in Figure 3.1 (bottom). Looking at the hierarchy, the structure of HEs with one piston direct drive can be easily learned, the only peculiar aspect being that these implements feature only one piston (*Piston*). The remaining components are common to *HydraulicElevator* or *Elevator*. In particular, the car frame (*CarFrameHydra*), i.e., the mechanical assembly connecting the car with the piston, is specific of hydraulic elevators. Albeit not physically part of the car frame, the entities *CarRails*, i.e., the rails along which the car is constrained to move, *Buffer*, i.e., the dumping device placed at the bottom of the elevator shaft, and *Ropes*, are logically part of it since their type and size must be inferred from or melded with the type and size of the car frame. Common to all elevator types, the entities *Shaft* and *Car* are both logically part of the *Elevator* entity, but only *Car* is also a physical component, together with its sub-component *CarDoor*. In the case of *Shaft*, while landing doors (*LandingDoor*) are not physically part of the shaft, they are attached to it and their size and type must be inferred from or melded with car doors. The relationships encoded in such part-whole hierarchy are instrumental to LIFTCREATE when it comes to handle drawing, storage and retrieval of designs, but also to reason about the various trade-offs of a design when searching in the space of potential solutions, as described in the next section.

3.2 Heuristic search

For the sake of clarity, in the ensuing discussion about LIFTCREATE CautoD procedures for hydraulic elevators it is assumed that only one supplier and build are available for car frames — including all logically-attached components, i.e., car rails, buffers and ropes — and for doors. This is not a severe limitation, as often designers and elevator installers will have their preferred pool of suppliers and builds for car frames and doors, opting for different ones only when the setup requires solutions which are manufactured only by specific suppliers. The CautoD procedure operates according to some predefined parameters:

- Reductions, i.e., distances from car to shaft on those sides of the car which are free from doors and car frame.
- Car wall thicknesses (different values for each car wall).
- Maximum car frame overhang (distance from the central axis of the car and piston).
- Choice of reduced or standard landing door frames.
- Door size tolerances with respect to other components, e.g., car frame.

Finally, it is assumed that the car will have only one door on the front, and that the car frame is to be placed either on the left side or at the back of the car. The case in which the car frame is placed to the right is simmetrical to the one considered.

Independently from whether LIFTCREATE uses heuristics or computer-intensive methods to guide the designer amidst alternative choices, the CautoD procedure scheme is the following:

1. Shaft size (width and depth) is input by the designer; no other configuration parameters are necessary since it is assumed that there is only one door on the front side of the car.
2. All available car frames are considered in ascending payload order; each selected car frame is placed either on the left or at the back of the car, aligned to its center.
3. Taking into account the selected car frame size, car wall thicknesses and reductions, the current internal width of the car is computed.
4. Door selection depends on whether heuristics or computer-intensive techniques are used (see below)
5. For each selected car frame and door, the weight of the car — doors included — and its payload are computed; given also the maximum overhang, it is possible to validate the selected car frame: if adequate, the current solution is saved into a list of feasible designs; otherwise, the solution is discarded and the procedure goes back to step (3).

To complete point (4) in the procedure scheme above, one could resort to either heuristics or computer intensive methods. In the former case, the following steps are taken:

a. The "internal cabin door" parameter — *ICD* in the following — is computed starting from the value computed in step (3) above, considering the size of landing door frames.

b. In order to select car and landing doors of feasible size, the ICD parameter, the shaft width, and the door size tolerances are considered to perform checks depending on the door types, i.e., sliding and folding; for the sake of brevity, details are omitted, but it is important to notice that such checks involve some non-trivial reasoning about door placement.

c. The doors selected at step (b), together with the selected car frame are part of the evaluation carried out in step (5) of the CautoD procedure scheme.

If computer-intensive methods are opted for, the following steps are taken:

a. Door sizes larger than the shaft are filtered away.

b. For each combination of door size and selected car frame, the parameter "residual car space" — RCD in the following — is computed; the parameter amounts to the difference between the shaft width and the total space allocated for the door; RCD is computed for each door type, since the space available for door placement is clearly a function of the current combination of door, car-frame, reductions and car wall thicknesses.

c. RCD is divided up into intervals of equal length — 5mm in the current implementation — so that a set of projects is generated, each with a different door placement; some of these projects will not be feasible, because door placement will not be coherent with the overall constraints, but these will be filtered at the end of the CautoD procedure.

While heuristics generate projects that are guaranteed to be feasible, the computer-intensive approach requires post processing to filter out remaining unfeasible projects. There are five checks that serve this purpose, namely:

- the door should not be placed outside the shaft;
- the door opening should be contained in the car;
- the car frame should be contained in the shaft;
- there should be no interferences between car doors and car frame;
- the landing door frame, once aligned to the car door, should not be outside the shaft;

If at least one of the checks above fails, the corresponding design is discarded.

3.3 Experimental results

The experimental evaluation is carried out considering eleven hydraulic elevator case studies — CS in the following. These include both configurations for which there exists feasible solutions, and configuration for which there are none. Among configurations for which feasible designs exist, both typical and "borderline" cases are considered. In more details:

- CS#1 features a shaft size which is too small to have feasible solutions;
- CS#2 is the minimum shaft size to have exactly one feasible solution: clearly the solution found has to be the same across heuristics and computer-intensive methods;
- CS#3-7 represent "typical" shaft sizes found in residential buildings;
- CS#8-10 feature unconventionally large shaft sizes;
- CS#11 features a shaft size which is too large to have feasible solutions.

In all the cases above, feasibility is constrained by the working hypothesis outlined before and by the available set of components. For instance, in CS#11 there are no feasible designs because in the component library there are no car frames available that can handle the resulting maximum payload. The experimental results herewith presented are obtained using LIFTCREATE prototype implemented in Java 8 and based on the SPRING object-persistence framework [1] using Vaadin[2] to generate and display GUIs. To handle components data and generated projects, a local instance of Mysql server 5.7 is adopted. All the simulations are executed on a machine with an Intel i7 5th generation 8 core CPU, featuring 8GB of RAM and running Ubuntu Linux 16.10.

Table 3.1 presents experimental data about the comparison between heuristics and computer-intensive methods on the selected case studies. In the Table, **CS** is the unique case study id, **W** and **D** are the corresponding shaft width and depth, respectively; both for heuristics and computer-intensive methods, **CPU** is the amount of time (in milliseconds) required to generate solutions, and **OK** is the number of feasible solutions found; **GEN** is the number of generated projects which, in the case of computer-intensive methods, does not readily correspond to feasible solutions, i.e., those that pass the checks mentioned at the end of the previous section.

[1] https://spring.io/.
[2] https://vaadin.com/home.

Table 3.1: Heuristics vs. computer-intensive methods.

CS	W	D	Heuristics		Computer-Intensive		
			CPU	OK	CPU	GEN	OK
1	900	830	235	0	1504	50	0
2	910	830	147	1	1488	120	1
3	1200	1000	670	122	3260	18972	1777
4	1200	1200	596	174	3736	18078	4181
5	1400	1000	1314	265	4122	40930	5939
6	1400	1200	943	291	3367	36440	11355
7	1600	1000	1606	406	3913	66717	9035
8	1600	1200	1349	470	2801	63854	24935
9	1800	1200	1614	516	2576	73046	28920
10	2000	1200	3628	360	1838	58888	22131
11	2000	1400	4236	0	1266	0	0

From Table 3.1 it can be observed that heuristics and computer-intensive methods do not present substantial differences when it comes to over-constrained configurations. In particular, for CS#1 and CS#2, the only noticeable element is that heuristics are faster than computer-intensive methods, as they can prune many unfeasible designs in the early stages of search — in CS#2 there is a $10\times$ factor between the two. As far as "typical" configurations are considered, the picture changes. The gap in performances is never greater than a $6\times$ factor, and computer-intensive methods are generating a strict superset of the projects generated by heuristics. The difference set is populated by solutions that, albeit feasible, do not correspond to straightforward designer choices, whose spirit is embedded into heuristics methods. Nevertheless, many such designs do have practical value. For instance, since computer-intensive methods explore many alternative door placements, they find solutions which often end up being preferred by implementors because they allow an easier fitting of cables or other implements, whereas customers may prefer them, e.g., because of aesthetic reasons. Finally, as for configurations which admit many alternative solutions, it can be observed that both heuristics and computer-intensive methods struggle with an ever-increasing search space. In some cases, e.g., CS#10, *a-posteriori* pruning techniques implemented in the computer-intensive approach end up being more efficient that heuristics.

In figure 3.2 an alternative view of the results shown in Table 3.1 is

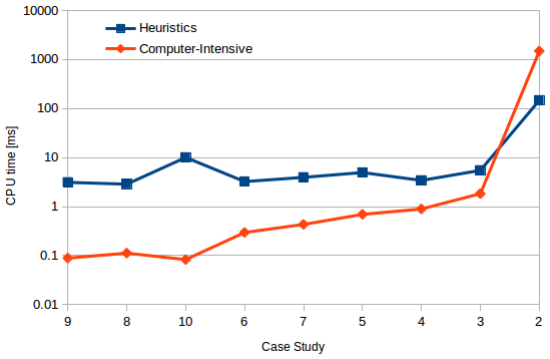

Figure 3.2: Average time to compute a single solution; case studies appear along the x-axis, ordered descendingly according to their estimated complexity; y-axis report CPU times in milliseconds, on a logarithmic scale; the plot report the performances of heuristic (squares) vs. computer-intensive (diamonds) methods.

shown. In the Figure, the average time to compute a single solution for heuristics and computer-intensive methods is plotted. For each case study, the average time per solution is just the ratio between the total time and the number of feasible projects — columns CPU and OK in Table 3.1. However, case studies are sorted along the abscissa of the plot considering an ascending order of configuration complexity, whose approximate indication can be obtained considering the number of solutions generated by the computer-intensive approach — column GEN in Table 3.1. The principle behind this choice is that, the more solutions are evaluated by computer-intensive techniques, the less constrained the original configuration is, and the less complex the overall problem is. Notice that CS#1, CS#2 and CS#11 are excluded from the plot in Figure 3.2 since the corresponding data would not make any sense. What can be observed from the plot is that, when the complexity of the problem is low, computer-intensive methods may have an edge over heuristics: it takes more than $10\times$ time to compute a single solution using heuristics, on average. The gap becomes smaller and smaller while the complexity increases. This is to be expected, because computer-intensive methods will have to reject more and more solutions, increasing the overhead to reach feasible solutions. For the most complex design, namely CS#2, the cost of computing 120 solutions in order to reach just one, is overwhelming with respect to what can be

achieved using heuristics. This reveals that, all other things being equal, computer intensive methods have an advantage over heuristics when the problem complexity is relatively low, whereas contrived configurations might benefit the most from the pruning power of heuristics.

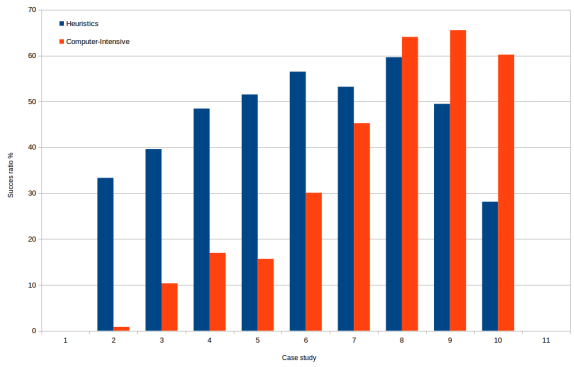

Figure 3.3: Success rate (percentage); each bar represents the success rate of either heuristic (blue) or computer intensive (orange) methods.

One last perspective about the results is shown in Figure 3.3. Here the success ratio of heuristics and computer-intensive methods is computed as a percentage value, considering the number of feasible designs and the number of attempts made to generate them. While in the case of computer-intensive methods this is simply the ratio between columns OK and GEN in Table 3.1, in the case of heuristics the value is computed considering the number of solutions that are "early-pruned", i.e., those which heuristics do not attempt to extend into a full-fledged solution. For all the case studies, except CS#1 and CS#11 which do not admit feasible solutions, Figure 3.3 tells that the success ratio of heuristics falls below 30% only in one case, i.e., CS#10, whereas computer-intensive methods are more configuration-dependent. In particular, the success ratios of "easy" configurations — namely CS#8-10 — are much higher than relativealy "hard" configurations, e.g., CS#2, but also CS#3-5. This confirms the evidence exctracted from the plot in Figure 3.2, i.e., that heuristics have an edge over computer-intensive methods on "hard" configurations, whereas "easy" configurations can be within the reach of computer-intensive methods.

Chapter 4

Genetic Algorithms

4.1 Preliminaries

Genetic Algorithms (GAs) are optimization procedures based on ideas borrowed from natural selection and evolution. Detailed descriptions of GAs are to be found, e.g., in [14]. For the purpose of this paper, it is sufficient to recall that GAs consider a *population* as a finite set P of potential solutions to the target optimization problem. Each individual $p \in P$ is characterized by a *genotype* comprised of *chromosomes*. As in nature, chromosomes define the individual and are the basis for the obtaining different individuals by "mating" procedures. The *fitness function* is a mapping $f : P \to \mathbb{R}$ which ranks the individuals according to a *fitness score*: the higher the chance of being a good solution, the higher the fitness score. Notice that GAs provide *unconstrained optimization* over the space of potential solutions. In order to take into account constraints, as our elevator design problem requires, the fitness function should contain one ore more *loss factors* — see, e.g., [15] — which penalize the individual design when it violates specific constraints: in this way, hard constraints are turned into preferences about solutions. By shaping the loss factors adequately we are able to control how much getting closer to violating a constraint can be discouraged. GAs are initialized with a randomly chosen population P and then they seek to improve the initial choice by repeating the following steps:

1. the fitness $f(p)$ of each individual $p \in P$ is computed;
2. a set $M \subset P$ is extracted from P such that individuals in M have the highest fitness among those in P;

3. the individuals in M are subject to "mating" procedures such as *crossover*, or other evolutionary phenomena such as *mutation*: informally, crossover occurs when the genotypes of two individuals are split and recombined to form new ones bearing some chromosomes, i.e., common traits, from both their parents.
4. The result of the previous step is a population P' which might contain individuals fitter than those of the previous population P; in particular, the crossover operation attempts to combine the genes of fit individuals to produce fitter children, and mutation attempts to maintain diversity in a population of designs.
5. Population P' becomes the new population P and the search restarts from step (1) unless some *termination condition* occurs, e.g., the fitness of the fittest individuals did not change in the last k steps, or a fixed number of h generations has been produced, where k and h are user-controlled hyper-parameters.

In our case each element of the population is a RHE design, and the genotype is meant to describe its main elements.

4.2 Model

4.2.1 Decision variables and parameters

We now formalize our model introducing the decision variables, the parameters and the constraints that involve them. We note that the reference system in LIFTCREATE origins from the top left corner of the internal shaft wall and the y axis is inverted with respect to a canonical Cartesian system. The origin $\mathcal{O}_{(x,y)}$ of this reference system coincides with the *shaft base point* (x_{shaft}, y_{shaft}) which is always set to $(0, 0)$ — see Figure 7.1. In Figure 7.2 we present a fragment of the plan view focusing on the car frame structure, which is comprised of the *brackets* — wall-mounted T-shaped components — to support the car rails on which the car frame *core gear* slides. The *car frame base point*, i.e., the insertion point of the car frame structure in the configuration, lies on the outer corner of the topmost bracket and it is marked with a cross. The coordinates of the car frame base point (x_{cf}, y_{cf}) — denoted by $(x, y)_{cf}$ in Figure 7.2 — determine a specific placement of the structure. The *overhang* of the car with respect to the car frame is the distance from the car walls to the car frame core gear edges — the top

edge is y_{gear}. As shown in the drawing, the overhang correspond to two parameters oh_1 and oh_2, required to handle the cases in which the car is not centered with respect to the car frame. The parameter dcr denotes the distance between the car rails, i.e., the size of the core gear. Starting from the base point of the car frame, w_{cf} and d_{cf} are the width and the depth of the car frame, respectively, whereas d_{br} is the depth of the brackets; the total encumbrance of the car frame in the shaft is given by the sum $d_{cf} + 2d_{br}$.

In Figure 7.3 we consider a fragment of the plan view focusing on the door pair — notice that the car door and the landing door *opening* must be aligned. The drawing in Figure 7.3 represents a pair of telescopic doors with 3 panels. The *car door base point* (x_{cd}, y_{cd}) — denoted as $(x, y)_{cd}$ in the plan view — and the *landing door base point* (x_{ld}, y_{ld}) — denoted as $(x, y)_{ld}$ in the plan view — are always at the top left corner of the corresponding structure. The value of these coordinates represents a specific placement of the car/landing door pair. The landing door opening is surrounded by the *frame*, i.e., the structure that surrounds the entrance to the car, with width w_{frame}. The total door width is the sum of two parameters, the *left axis* — la_{cd} and la_{ld} for car and landing door, respectively — and the *right axis* — ra_{cd} and ra_{ld} for car and landing door, respectively. Both axes originate from the opening midpoint and, as shown in the drawing of Figure 7.3, in general they may not coincide. Finally, $step_{cd}$ and $step_{ld}$ denote the depth of the step in the car and landing door, respectively; d_{step} is the distance between car and landing doors.

In Tables 7.1 and 7.2 we summarize all the quantities involved in the configuration, separating decision variables (Table 7.1) from parameters (Table 7.2) either related to the initial specification or extracted from the components database — all quantities are in millimeters. We introduced all the decision variables beforehand with the exception of (x_{car}, y_{car}), i.e., the car base point coordinates corresponding to the top-left internal edge of the car in Figure 7.1, and w_{car} and d_{car}, i.e., the width and depth of the car. Concerning parameters, we consider four groups of them. The first group is related to shaft measurements and includes w_{shaft} and d_{shaft} — width and depth of the shaft, respectively; also in this group we have *reductions* (red_N, red_S, etc.), i.e., the distance between the car walls and the shaft, and *car wall thicknesses* (cwt_N, cwt_S, etc.). For both such groups of parameters we have four values (N, S, W and E) to account for different sizes on all sides (top, bottom, left and right, respectively). The second group is related to car frame dimensioning and includes max_{oh}, i.e., the maximum overhang, and other parameters detailed in the table. The third group is related to

doors, with no further introduced parameters.

4.2.2 Constraints

Having defined our decision variables and parameters, we proceed to describe the (hard) constraints required to find feasible solutions, divided into two groups related to car frame and doors respectively. The constraints to place the car frame must take into account two main issues. First, given the shape of the brackets, it is not possible to model the car frame as a simple rectangle in order to fit it with the other components. Therefore the placement of the car frame is computed by subtracting residuals from the total shaft measures. Second, the placement of the car frame must take into account its maximum overhang, i.e., the car cannot "lean" too much outside the car frame core gear. The considerations above lead to the following set of constraints:

$$\begin{cases} y_{cf} - d_{br} \geq y_{shaft} \\ y_{cf} + d_{cf} + d_{br} \leq y_{shaft} + d_{shaft} \\ 0 \leq y_{cf} + y_{gear} - y_{car} < max_{oh} \\ 0 \leq y_{car} + d_{car} - y_{cf} - y_{gear} - dcr < max_{oh} \end{cases} \tag{4.1}$$

The first two constraints are required to fit the shape of the car frame, while the last two are required to satisfy the requirement about the maximum overhang.

The constraints to place the car/landing door pair should guarantee that both structures fit the shaft, that the actual opening fits the car and that the landing door frame does not exceed the shaft size. These requirements can be translated into the following set of constraints:

$$\begin{cases} x_{cd} \geq x_{shaft} \\ x_{ld} \geq x_{shaft} \\ x_{cd} + la_{cd} + ra_{cd} \leq x_{shaft} + w_{shaft} \\ x_{ld} + la_{ld} + ra_{ld} \leq x_{shaft} + w_{shaft} \\ x_{cd} + la_{cd} - \frac{opening}{2} \geq x_{car} \\ x_{cd} + la_{cd} + \frac{opening}{2} \leq x_{car} + w_{car} \\ x_{ld} + la_{ld} + \frac{opening}{2} + w_{frame} \leq x_{shaft} + w_{shaft} \end{cases} \tag{4.2}$$

The first four inequalities are required to guarantee that the car and the landing door structures fit the shaft; then we list two inequalities related

to the car opening, and the last inequality guarantees that the landing door frame size is adequate for the shaft. In addition, the alignment of the landing door with respect to the car door must be enforced with the following equality constraint:

$$x_{ld} = x_{cd} + la_{cd} - la_{ld} \tag{4.3}$$

The placement of the car and landing door on the y axis is also enforced with equality constraints:

$$\begin{cases} y_{ld} = y_{shaft} + d_{shaft} - step_{ld} \\ y_{cd} = y_{ld} - d_{step} - step_{cd} \end{cases} \tag{4.4}$$

Further equality constraints are required to take into account that the door placement over the y axis, together with the car frame and door selection, influences the car size as follows:

$$\begin{cases} x_{car} = w_{cf} + cwt_W \\ y_{car} = red_N + cwt_N \\ w_{car} = w_{shaft} - w_{cf} - cwt_W - cwt_E - red_E \\ d_{car} = d_{shaft} - red_N - cwt_N - cwt_S - H_{doors} \end{cases} \tag{4.5}$$

where H_{doors} stands for the total door occupancy over the y axis computed as:

$$H_{doors} = step_{ld} + step_{cd} + d_{step} \tag{4.6}$$

Notice that when the car frame is positioned on the left hand side of the elevator, its x base coordinate x_{cf} is always set to 0.

Since the car door body may protrude over the car walls, in order to minimize the risk of collision with other components, designers must consider a safety margin. To guarantee this requirement, specific non-overlapping constraints are implemented. For example, if we let r be the security margin, the non-overlapping constraint relative to car frame and car door can be written as follows:

$$x_{cd} - r \geq x_{cf} + w_{cf} \vee y_{cd} - r \geq y_{cf} + d_{cf} \tag{4.7}$$

4.3 Implementation and experimental results

In this Section we introduce and present experimental data about three different solution strategies, namely custom heuristics (LIFTCREATE-HR),

genetic algorithms (LIFTCREATE-GA) and SMT solvers (LIFTCREATE-SMT). We will show some results obtained with brute-force (LIFTCREATE-BF) and random sampling (LIFTCREATE-RS) in the space of possible solutions, to witness that the configuration task we consider is complex enough to make exhaustive search unfeasible and random search ineffective. Our analysis is empirical, and the data to support our conclusions are obtained by running different versions of LIFTCREATE using the same database of components — 25 car frames and 236 doors — to configure RHEs in thirty different setups. These include both cases in which, given the available components, feasible solutions exist, and others for which there are none. The setups we consider represent typical shaft sizes found in residential buildings: two families of 15 setups, the former featuring 1300 mm shaft width and the latter featuring 1500 mm shaft width; shaft depth varies in both families from 800 mm to 1500 mm. Overall, these setups enable a thorough evaluation of LIFTCREATE versions considering realistic settings.

Due to the specific features of LIFTCREATE versions, there are some differences in how configurations are generated and results are presented, differences that we describe in the following and that we tried to harmonize as much as possible in order to make our experimental comparison meaningful. In particular, since the methodology implemented by LIFTCREATE-HR is under patent scrutiny, we do not disclose its details and consider it as a black-box. LIFTCREATE-HR produces at most one solution — supposedly the "best" one according to the heuristics — for each *prototype*, i.e., a pair comprised of a door and a car frame which together fit the shaft. Therefore, for each given setup, LIFTCREATE-HR produces as many solutions as there are prototypes, where a solution features a specific placement of car frame and doors. The three versions of LIFTCREATE based on search in the space of configurations, namely LIFTCREATE-BF, LIFTCREATE-RS and LIFTCREATE-GA, feature a common data flow implementing the following phases:

- *Prototype generation*: amounts to list all prototypes, pruning up-front those which cannot fit the given shaft.
- *Expansion*: given a prototype, potential configurations are explored by attempting to place the car frame and the doors inside the shaft within the allowable ranges: the results of this phase are *early designs*.
- *Design validation*: given an early design, a number of checks is performed in order to validate the corresponding configuration and declare it

feasible or not.

Specifically, the validation phase must guarantee that every constraint of Section 4.2.2 is satisfied. The difference among LIFTCREATE-BF, LIFTCREATE-RS and LIFTCREATE-GA lies in the expansion phase: exhaustive search for LIFTCREATE-BF, random sampling for LIFTCREATE-RS and genetic algorithms for LIFTCREATE-GA. Since these versions may produce many feasible configurations for each prototype, whereas LIFTCREATE-HR outputs only one, we *cluster* configurations after the validation phase. In more details, the set of valid placements for each given prototype is clustered around a representative, in order to make the comparison with LIFTCREATE-HR possible. Finally, in LIFTCREATE-SMT the prototype expansion phase is replaced by the SMT encoding of the constraints introduced in Section 4.2.2, where the choice of components is restricted to those that can fit the shaft. The subsequent phases of expansion and validation are merged in the search for a solution by the SMT solver. If a cost function to drive the search towards "optimal" designs is supplied, then LIFTCREATE-SMT outputs exactly one configuration for each given setup.

4.3.1 Custom heuristics: LIFTCREATE-HR

The results of LIFTCREATE-HR are reported in Table 7.3. As the results show, all the setups can be solved in less than 2 CPU seconds [1]. The number of configurations found ranges from 0 for the two setups having shaft depth 800 mm, to more than one thousand for deeper shafts. Notice that the number of configurations found by LIFTCREATE-HR may not coincide with the total number of feasible configurations. This is because heuristics in LIFTCREATE are geared towards providing arrangements that a human designer finds satisfactory and not just feasible ones.

To better appreciate the complexity of the configuration task and the results obtained with LIFTCREATE-HR, in Table 7.4 we present also the results obtained with LIFTCREATE-BF. In these experiments, a timeout of 30 minutes has been set in order to avoid excessively long runtimes. As expected, the runtime of LIFTCREATE-BF grows with the shaft size and it is up to three orders of magnitude greater than LIFTCREATE-HR. Consider now the difference among the time to generate prototypes (TTP), the time to expand them (TTE) and the time to validate (TTV) early designs. As it

[1] The heuristic search herewith considered focuses only on the car frame and doors coupling, so that the results of the full design may appear inconsistent with [16]

can be observed, the largest slice of runtime is consumed by the validation phase. This is reasonable because even if the expansion phase generates a large number of alternatives, no processing is performed on them. Notice also that the number of early designs (column "E" in Table 7.4) is about four to five times greater than the number of valid designs for all but the setups for which no feasible configuration exists. This means that LIFTCREATE-BF wastes a lot of processing time just to discard unfeasible configurations.

Finally, we clustered all the valid designs generated by BF search according to the procedure described before. Looking at column "C" in Table 7.4, we can see that the number of clusters is much smaller than the number of valid designs, indicating that LIFTCREATE-BF finds many feasible configurations sharing the same car door and frame with slightly different placements. On the other hand, LIFTCREATE-HR nails down one design for each pair of car door and frame *at most*. As we mentioned previously, the output of LIFTCREATE-HR is supposedly the "best" alternative according to a human project engineer, but LIFTCREATE-HR does not explore exhaustively all possible feasible placements, which means that some pairs of car door and frame are discarded beforehand. This difference surfaces in column "C ∩ H" of Table 7.4 where we count the number of clusters shared between LIFTCREATE-HR and LIFTCREATE-BF. As we can see, the values in column "C" are always greater than or equal to the values of column "C ∩ H", i.e., LIFTCREATE-BF finds all the feasible configurations found by LIFTCREATE-HR, but its runtime is not adequate for practical applications. On the other hand, LIFTCREATE-HR is able to find most of the feasible design clusters in a fraction of the time taken by LIFTCREATE-BF.

4.3.2 Genetic algorithms: LIFTCREATE-GA

In LIFTCREATE-GA the genotype is composed by 4 chromosomes, each one consisting of a single gene. As in [17], genes are represented by integer numbers as follows:

- *Gene 1*: Value of the car door x base point — x_{cd} in Table 7.1.
- *Gene 2*: Value of the car frame y base point — y_{cf} in Table 7.1.
- *Gene 3*: Choice of the car frame; to encode the choice among available car frames we assigned to each one a unique integer identifier.
- *Gene 4*: Choice of the car door; also in this case we encoded each door with a unique integer code.

Since the available car frame and door identifiers, i.e., the domains of the genes 3 and 4, are restricted to those that could fit a given shaft, each individual represents a single early design. We do not encode the variables x_{cf} and y_{cd} because, as pointed out in Section 4.2.2, the coordinates are fixed in the design.

The fitness function to score individuals is computed by associating costs corresponding to violations of the feasibility constraints presented in Section 4.2, i.e., we turn hard constraints into soft ones to discourage designs that violate the constraints. However, we cannot completely exclude unfeasible designs and this is why LIFTCREATE-GA still retains a validation step at the end to make sure that all generated designs are valid. The cost function built in LIFTCREATE-GA includes also two terms that provide some bias towards designs that are considered more desirable than others. In particular, these terms encode preferences for choice and placement of doors, and placements of the car frame. Regarding the doors, we penalize projects whose door placement is misaligned with respect to the car. To this end, we define the car axis along the x coordinate as the car base point x coordinate plus half of the car width:

$$axisX_{car} = x_{car} + \frac{w_{car}}{2}$$

and the door axis along the x coordinate as the car door base point x coordinate plus the length of its left axis:

$$axisX_{door} = x_{cd} + la_{cd}$$

In the case of telescopic of folding symmetric doors — i.e., when the opening midpoint coincides with the door frame midpoint — good design practices suggest that $axisX_{door}$ and $axisX_{car}$ should be aligned. In this case, the cost corresponds to the absolute difference of the two. In the case of asymmetric telescopic doors, considering that we are configuring an elevator with door opening on the "bottom" and the car frame on the "left" of the plan view in Figure 7.1, it is preferable to have the opening as close as possible to the "right" side of the car wall defined as:

$$x_{wall} = x_{car} + w_{car}$$

When considering this case, the cost is the difference between x_{wall} and the right edge of the opening defined as:

$$x_{edge} = x_{cd} + la_{cd} + \frac{opening}{2}$$

Finally, doors whose opening is less than 550mm and doors with opening greater than 800mm are discouraged: the former are used only in very special situations and the latter are usually very expensive. For this reason we try to prefer designs with the doors opening far enough from these limit values. The corresponding cost is then generated by summing up three values:

1. The absolute value of the difference between the car x axis and the door axis if the door is symmetric, the difference between x_{wall} and x_{edge} otherwise.
2. A parabolic function that grows up quickly for values less than 550mm or greater than 800mm. The function we consider is $f(x) = 0.01 \cdot x^2 - 13.5 \cdot x + 4561.25$ computed with $x = opening$.
3. A fixed weight which is either 10^5, if the actual door width is less than 550mm, or 300 if the actual door width is greater than 800mm.

We use a parabolic function because, as we mentioned before, GAs provide unconstrained optimization over the solution space, and a continuous objective is better for the fitness computation.

Regarding the car frame, we penalize projects in which the car frame is misaligned with respect to the car axis — i.e., considering Figure 7.1, the car frame should appear vertically centered with respect to the car. We define the car frame axis as the car frame base point y coordinate plus half of the distance between the car rails:

$$axisY_{cf} = y_{cf} + \frac{dcr}{2}$$

and the car axis along the y coordinate as the car base point y coordinate plus half of the car depth:

$$axisY_{car} = y_{car} + \frac{d_{car}}{2}$$

The cost term is the absolute value of the difference between the two axes. If the choice of the car frame is such that overhangs are negative — i.e., if the car frame dcr is greater than the resulting car depth d_{car} — the resulting value is multiplied by 10^3.

In Table 7.5 we present a comparison among LIFTCREATE-HR, LIFTCREATE-RS and LIFTCREATE-GA. The mutation rate was set to 10% which turned out to be the best value according to some preliminary experiments that

we do not show here for the sake of brevity. LIFTCREATE-RS provides a baseline for LIFTCREATE-GA: we consider LIFTCREATE-GA effective as long as it can perform better than LIFTCREATE-RS, either in the quality of the final designs and/or in the time spent to find them. Notice that LIFTCREATE-RS samples only a few configurations — 10% of the total number of early designs in our implementation. More specifically, for each prototype, candidate designs are selected by sampling uniformly at random without replacement of the space of early designs. Candidate designs are then subject to the validation phase in order to discard unfeasible ones. Detailed results about LIFTCREATE-RS can be found in Table 7.6 and Table 7.7. As far as runtimes are concerned, we can observe that LIFTCREATE-GA is consistently faster than LIFTCREATE-RS: in most cases, the performance gap reaches one order of magnitude. On the other hand, the comparison with LIFTCREATE-HR is not favorable, particularly in the setups where the search space for potential designs is larger and LIFTCREATE-GA takes more time to converge. However, as mentioned before, we can see that the similarity between the designs found by LIFTCREATE-GA and those computed by LIFTCREATE-HR is very high, reaching 100% in the majority of cases, whereas LIFTCREATE-RS does not reach such figures, not even in the setups with a relatively small search space — see Table 7.6. From the figures of Table 7.5 we can conclude that LIFTCREATE-GA consistently outperforms LIFTCREATE-RS and thus the genetic model as well our choice of hyper-parameters make for a significantly better choice than basic random sampling. In terms of sheer performances LIFTCREATE-HR is still superior to LIFTCREATE-GA, but the runtimes of the latter are reasonable for online deployment with the added flexibility of a "declarative" encoding: adding a new constraint to LIFTCREATE-GA only amounts to add a term to the fitness function, whereas in the case of LIFTCREATE-HR any change involves modifying the code.

Chapter 5

Constraint Satisfaction

5.1 Preliminaries

Satisfiability Modulo Theories is the problem of deciding the satisfiability of a first-order formula with respect to some decidable theory $\mathcal{T}$. In particular, SMT generalizes the Boolean satisfiability problem (SAT) by adding background theories such as the theory of real numbers, the theory of integers, and the theories of data structures (*e.g.*, lists, arrays and bit vectors) — see, e.g., [18] for details. To decide the satisfiability of an input formula φ in conjunctive normal form, SMT solvers typically first build a *Boolean abstraction* $abs(\varphi)$ of φ by replacing each constraint by a fresh Boolean variable (proposition), e.g.,

$$\begin{array}{rl} \varphi & : \underbrace{x \geq y}_{A} \wedge (\underbrace{y > 0}_{B} \vee \underbrace{x > 0}_{C}) \wedge \underbrace{y \leq 0}_{\neg B} \\ abs(\varphi) : & \quad A \quad \wedge (\quad B \quad \vee \quad C \quad) \wedge \quad \neg B \end{array}$$

where x and y are real-valued variables, and A, B and C are propositions. A propositional logic solver searches for a satisfying assignment S for $abs(\varphi)$, e.g., $S(A) = 1$, $S(B) = 0$, $S(C) = 1$ for the above example. If no such assignment exists then the input formula φ is unsatisfiable. Otherwise, the consistency of the assignment in the underlying theory is checked by a *theory solver*. In our example, we check whether the set $\{x \geq y,\ y \leq 0,\ x > 0\}$ of linear inequalities is feasible, which is the case. If the constraints are consistent then a satisfying solution (*model*) is found for φ. Otherwise, the theory solver returns a theory lemma φ_E giving an *explanation* for the conflict, e.g., the negated conjunction some inconsistent input constraints. The explanation is used to refine the Boolean abstraction $abs(\varphi)$ to $abs(\varphi) \wedge abs(\varphi_E)$. These steps are iteratively executed until either a theory-consistent

Boolean assignment is found, or no more Boolean satisfying assignments exist.

Adding theories of cost to SMT yields Optimization Modulo Theories (OMT), an extension that finds models to optimize given objectives through a combination of SMT and optimization procedures [19]. For example,

$$\begin{cases} \varphi : x \geq y \wedge (y > 0 \vee x > 0) \wedge y \leq 0 \\ \min_{x,y}(x + y) \end{cases}$$

requires all the constraints in φ to be satisfied and the additional cost $x + y$ to be minimized. Notice that OMT extends classical formulations in mathematical programming, e.g., linear programming or mixed integer linear programming, since it allows Boolean structure to be taken into account together with the optimization target. OMT solvers have been developed for several first-order theories like, e.g., those of linear arithmetic over the rationals (LRA) or the integers (LIA) or their combination $(LIRA)$. In this paper we consider *quantifier free* theories in a mixed integer/rational domain — known as QF_LIRA in the literature [20]

5.2 Constraint satisfaction encoding

5.2.1 Component selection

The car frame, the cylinder and the doors are selected from a database of components. In order to automate the design of an elevator, we must consider that choosing different components yields different parameter values for each one. The relationship between the selection of a component and the assignment of the corresponding parameter values can be encoded via Boolean implications of the form

$$Id_x = i \Rightarrow x.p = v \tag{5.1}$$

where Id_x encodes the identifier of choice for component x (a decision variable), i is a specific identifier value, $x.p$ is some parameter of the component x and v is the value of $x.p$ given that the component x with identifier i was chosen. To encode constraints of the form (5.1) a combination of Boolean reasoning with Integer arithmetic is sufficient. However, considering the way data sets are usually encoded in MiniZinc with arrays [21], we consider an alternative encoding where we associate an array to each component parameter. For example, if a component x has two parameters

Surface (A)	Passengers (P_0)
$a1 < A \leq a2$	P_0
$a2 < A \leq a3$	$P_0 + 1$
$A > a3$	$P_0 + 2$

Table 5.1: Look-up table to encode the number of passengers as a function of the car surface.

p_1 and p_2, we build two arrays P_1 and P_2 that will store the values of p_1 and p_2 for each instance of the component. The index of the arrays becomes a decision variable and its choice by the solver enforces the correct values for the parameters.

5.2.2 Look-up tables

Some parameters, e.g., the maximum number of passengers that the car may accommodate, are a function of others, e.g., the car surface. However, instead of expressing such constraints directly — which might involve the use of non-linear or transcendental functions — the correspondence between free parameters and derived ones is encoded with *look-up tables*. Table 5.1 exemplifies such a table assuming that the car surface A is contained within three ranges. The car payload is computed in a similar way, but, since the surface ranges are different, we need another set of constraints structured in the same way. These requirements can be easily modeled with implications in the same way as component selection: the surface A is a decision variable that implies the number of passengers or the payload. However, both SMT-LIB [1] language [22] and MiniZinc allow users to define custom *functions*. In practice, functions are series of *if-then-else* statements about, e.g., the car surface, where each function returns, e.g., the corresponding number of passengers or the payload.

5.2.3 Integers vs. Reals

Most parameters involved in the design process are expressed in millimeters which suggests integer-based encodings. However, some parameters like the forces exerted on the car rails require arithmetic over reals. This makes the corresponding constraint satisfaction problems members of the mixed-integer arithmetic family. In such encodings, the main disadvantage is that

[1] http://smtlib.cs.uiowa.edu/

a large number of integer variables may increase considerably the solution time. We try to improve on this by relaxing some of the integer variables to reals. In particular, we consider component parameters since parameters are not *decided* but their value is only assigned based on the choice of a component. This means that the domain of the parameters is a finite set and we can relax the arithmetic encoding without producing invalid results. In this representation the only operation that could add decimal digits is division, but since in our encoding there are only a few such operations, boundary checking can be implemented easily. These considerations do not hold for some decision variables including, but not limited to, the index used to select components. Also, CP solvers like Chuffed are not affected by this choice due to the fact that they do not support floating-point arithmetic.

5.2.4 Single and Multi-objective optimization

Here we describe alternative constructions of the cost functions, mentioning the details of the parameters involved when necessary. In particular, we encode the design objectives associating a cost to each one of them. The cost associated to car frame misalignment on the y axis is expressed by the absolute value of the distance between the car frame and the car axes. We define the car frame vertical axis $axisY_{cf}$ as the car frame vertical base point y_{cf} plus half of the distance between the car rails dcr

$$axisY_{cf} = y_{cf} + \frac{dcr}{2}. \tag{5.2}$$

The vertical car axis $axisY_{car}$ is defined as the car vertical base point y_{car} plus half of the car depth d_{car}:

$$axisY_{car} = y_{car} + \frac{d_{car}}{2}. \tag{5.3}$$

The difference between the terms (5.2) and (5.3) gives us the first contribution to the cost function c_{cf}:

$$c_{cf} = |axisY_{cf} - axisY_{car}| \tag{5.4}$$

The second objective we consider is related to doors. In this case we define the horizontal car axis $axisX_{car}$ as the horizontal car base point x_{car} plus half of the car width w_{car}:

$$axisX_{car} = x_{car} + \frac{w_{car}}{2} \tag{5.5}$$

The horizontal door axis $axisX_{door}$ is defined as the horizontal door base coordinate x_{cd} plus the length of its left axis la_{cd}:

$$axisX_{door} = x_{cd} + la_{cd} \tag{5.6}$$

In the case of symmetric doors, good design practices suggest that $axisX_{door}$ and $axisX_{car}$ should be aligned. In the case of asymmetric doors, it is preferable to have the door *opening* as close as possible to the side of the car which is opposite to the car frame. In a configuration like the one in Figure 7.1 we can define the base coordinate of such side as:

$$x_{wall} = x_{car} + w_{car} \tag{5.7}$$

To take into account the different arrangement of doors, we introduce a binary variable, δ_t, which is assigned to 1 if the current door is an asymmetric door and to 0 otherwise. We can then summarize the contribution to the cost function as:

$$\begin{aligned} c_{door} = (\quad & (1-\delta_t)|axisX_{car} - axisX_{door}| + \\ & \delta_t(x_{wall} - (x_{cd} + la_{cd} + \tfrac{opening}{2})) \quad) \end{aligned} \tag{5.8}$$

The first contribution of (5.8) is zero when $\delta_t = 1$, i.e, for asymmetric doors we try to minimize the distance from the side of the elevator opposite to the car frame, whereas when $\delta_t = 0$ we try to align the door and the car axes.

The third objective is related to the components selection itself, and gives the guidelines for sizing the car frame, the doors and the cylinder. The maximization of the door *opening* leads to accessible elevators which are always considered a plus, whenever feasible; the minimization of the car frame depth dcr and the barrel diameter d_p suggests components which are not over-sized, thus helping to keep costs at bay. These criteria can be translated into one additional contribution to the overall cost function defined as:

$$c_{size} = (dcr + d_p - opening) \tag{5.9}$$

Finally, the last term to minimize is the sum of F_{cr}^x and F_{cr}^y, i.e., the x and y components of the force exerted on the car rails F_{cr}:

$$c_{cr} = F_{cr}^x + F_{cr}^y \tag{5.10}$$

The computation of F_{cr} is non-trivial and requires additional equations and parameters that we briefly describe. The components of F_{cr} are obtained as

$$\begin{aligned} F_{cr}^{x} &= k \cdot g \cdot \frac{Q_x(Q+75)+P_x \cdot car_W + cdP_x \cdot cd_W + cf_W \cdot CF_x}{2 \cdot h} \\ F_{cr}^{y} &= k \cdot g \cdot \frac{Q_y(Q+75)+P_y \cdot car_W + cdP_y \cdot cd_W}{2 \cdot h} \end{aligned}$$

where the parameters have the following meaning:

- k is a parameter depending on the kind of safety brakes installed;
- g is the standard acceleration due to gravity;
- Q is the car payload;
- P_x and P_y are the midpoint coordinates of the car;
- Q_x and Q_y are obtained through the equations

$$\begin{aligned} Q_x &= max\{P_x + \tfrac{w_{car}}{8}, P_x - \tfrac{w_{car}}{8}\} \\ Q_y &= max\{P_y + \tfrac{d_{car}}{8}, P_y - \tfrac{d_{car}}{8}\}; \end{aligned}$$

- car_W is the car weight;
- cdP_x, cdP_y are the coordinates of the center of gravity of the car door;
- cd_w is the car door weight and cf_W is the car frame weight;
- CF_x is a coefficient computed as

$$CF_x = 1.5 \cdot \frac{w_{cf}}{2}$$

  where w_{cf} is the distance from the car frame base point to to the left car wall;
- h is the distance between guide shoes, i.e., the supports which slide on the car rails.

In previous works of ours we consider the weighted sum of the costs c_{cf}, c_{door} and c_{size} to obtain the overall cost function, but the contribution c_{cr} may conflict with the previous ones because the farthest is the door from the car frame, the greater is the force exerted on the car rails. Nevertheless, since the car rails can be chosen once the other components are fitted, this objective can be considered with a lower priority. If we follow a single-objective approach, we can weight the cost c_{cr} significantly less than the other three. The overall cost function $\mathcal{C}$ becomes

$$\mathcal{C} = \alpha_1 c_{cf} + \alpha_2 c_{door} + \alpha_3 c_{size} + \alpha_4 c_{cr} \tag{5.11}$$

with $\alpha_4 \ll \alpha_i$ for $i \neq 4$. Alternatively, we can exploit priorities among different cost functions by resorting to multi-objective optimization using, e.g., the lexicographic method whereby preferences are imposed by ordering the objective functions according to their significance — see [23] for details. Here we consider two cost functions:

$$\begin{array}{rcl} \mathcal{C}_1 & = & \alpha_1 c_{cf} + \alpha_2 c_{door} + \alpha_3 c_{size} \\ \mathcal{C}_2 & = & c_{cr} \end{array} \tag{5.12}$$

where the objective function $\mathcal{C}_1$ is minimized first.

5.3 Experimental Results

To understand how different choices of encoding our problem impact on different solvers, we consider the OMT solvers z3 [24] and OptiMathSat [25] fed with SMT-LIB encodings, and five solvers fed with the corresponding encodings in MiniZinc language [21], namely Chuffed [26], OR-Tools [27], ECL^{i}PS^e CLP [28] CPLEX [29] and Gurobi [30].[2] All the results are obtained considering setups with shafts of varying sizes: we have a set of eight shafts with fixed width of 1300 millimeters and another set with 1500 millimeters width. In both cases, the depth goes from 800 to 1500 using a 100 mm step. Each experiment is subject to a timeout of 5 minutes of CPU time, and the times are expressed in milliseconds.[3] We consider two different sets of experiments: a baseline encoding dealing with the configuration of the car frame and the door pair only, and a full encoding dealing also with the selection and sizing of the hydraulic cylinder as well as the minimization of forces on the car rails. In particular, in the baseline encoding we consider only the cost components related to car frame and doors, whereas the full encoding takes into account all the cost components. Overall, the baseline encoding features 39 parameters of which 10 are decision variables, whereas the full encoding features 59 parameters and 17 decision variables. The number of constraints varies from a minimum of 30 of the baseline encoding considering arrays and functions to 401 of the full encoding considering implications for both parameters and look-up tables. Both baseline and full encodings are generated considering a database of 25 car frames, 236 doors and 47 hydraulic cylinders.

[2] In particular, we run Chuffed v0.10.3, OR-Tools v7.8, ECL^{i}PS^e v7.0, CPLEX v12.7, Gurobi v9.0.1, z3 v4.8.7 and OptiMathSat v1.7.0.1, all in their default configuration.

[3] All tests run on a PC equipped with an Intel® Core™ i7-6500U dual core CPU @ 2.50GHz, featuring 8GB of RAM and running Ubuntu Linux 16.04 LTS 64 bit.

Shaft	OR-Tools	Chuffed	ECL^{i}PS^e	CPLEX	Gurobi	z3	OptiMathSat
1300 × 800	662	200	924	856	886	**100**	254
1300 × 900	645	**198**	703	1020	1802	432	30680
1300 × 1000	674	**192**	1401	918	933	416	58066
1300 × 1100	659	**179**	1734	940	971	582	154739
1300 × 1200	655	**191**	1796	1056	1237	417	82698
1300 × 1300	661	**188**	1771	1090	1725	495	100822
1300 × 1400	637	**188**	1366	918	887*	435	79323
1300 × 1500	672	**206**	875	1118	925*	517	98355
1500 × 800	644	199	678	1023	824	**116**	247
1500 × 900	664	**179**	691	902	881	787	101458
1500 × 1000	673	**195**	1379	987	887*	619	70082
1500 × 1100	639	**206**	1942	971	903	682	105071
1500 × 1200	660	**264**	2024	1060	934	501	83719
1500 × 1300	636	**224**	2412	987	1018	417	121801
1500 × 1400	645	**192**	1509	871	919	470	97753
1500 × 1500	653	**216**	845	856	935	463	142557

Table 5.2: Comparison of solvers on the baseline encoding: the first column reports the setup and the other columns report the time (ms) taken to solve each setup by the solvers — best times appear in boldface.

In Table 5.2 we show the results obtained on the baseline encoding by all the solvers we consider. For each solver we report the best time obtained on two variations: one in which the selection of components is based on arrays and another featuring Boolean implications. Both variations are integer-based because not all the solvers support arithmetic over reals, so we do not consider relaxations; also, since the car surface computation involves a division, we omit the deduction of the car payload and passengers which are required for a complete design. All the solvers leveraging MiniZinc encodings fare the best runtime when the component parameters are encoded with arrays: CP solvers like Chuffed seem to make effective use of element constraints and MIP solvers appear to handle the translation of array constraints better than Boolean implications. On the other hand, OMT solvers run faster on the version based on Boolean implications, as the addition of arrays involves dealing with more theories at once and this inevitably hurts performances.

As we can observe in Table 5.2, Chuffed is the one yielding the best runtimes, except for two setups where z3 is the fastest solver. Noticeably, these setups do not admit a feasible configuration given the shaft size and the components available. z3 and OR-Tools are second best, their runtimes being always less than one second; MIP solvers CPLEX and Gurobi seem

Shaft	z3						OptiMathSat						Heuristic
	I		I + R		I + R + F		I		I + R		I + R + F		
	SO	MO	SO	MO	SO	MO	SO	MO	SO	MO	SO	MO	
1300 × 800	157	149	131	134	143	221	109	119	**100**	131	116	110	583
1300 × 900	8878	229522	1205	**844**	1978	2321	—	—	74960	53535	68215	52068	1784
1300 × 1000	36120	133325	2818	3362	4433	**2704**	156761	48328	136109	107356	132166	112379	921
1300 × 1100	36448	60589	3514	1967	2365	**1554**	192198	127753	160883	176491	199352	113889	2177
1300 × 1200	42328	5530	6876	3460	**2637**	4155	—	208852	276380	181817	193987	160401	6865
1300 × 1300	94325	8982	22279	**2521**	5294	5304	244973	129848	225067	165818	292777	197777	15278
1300 × 1400	30452	133087	7374	**1779**	11096	3707	259953	244078	—	256791	—	242488	11190
1300 × 1500	177355	25697	18810	4061	234235	**1998**	258119	213104	259693	172842	274986	222485	24380
1500 × 800	176	141	121	114	140	129	100	**85**	100	**85**	100	101	926
1500 × 900	25964	56674	1619	**1212**	3382	1876	141359	—	206751	95370	167671	100623	5215
1500 × 1000	91242	235192	2888	1803	5121	**1725**	—	118777	223241	93759	173623	187153	2952
1500 × 1100	—	18023	4977	7517	**3925**	4446	219596	187570	205041	179414	183862	156035	4875
1500 × 1200	139993	68562	7001	**1242**	7431	1571	251651	148664	231829	70431	254111	189705	6232
1500 × 1300	291712	—	26724	4895	20263	**4325**	—	225509	—	232735	—	255728	33785
1500 × 1400	—	6264	35073	3139	169215	**2675**	—	184555	271054	107886	—	180857	21910
1500 × 1500	—	17824	37722	2703	121472	**2528**	257242	222360	—	167762	—	251033	8699

Table 5.3: Comparison of z3 and OptiMathSat on the full encoding: the first column reports each setup; the other columns, grouped by solver, report runtimes (ms) of different versions: integer-based "**I**", relaxed "**I + R**" and relaxed with functions "**I + R + F**", respectively. Subcolumns "**SO**" and "**MO**" refer to single objective and multiobjective optimization, respectively — best times among z3 and OptiMathSat appear in boldface. The last column reports LIFTCREATE heuristic engine runtimes.

slightly less effective than the leading pack. In some cases, marked with an asterisk in Table 5.2, Gurobi returned *UNSAT or UNKNOWN* even if a solution exists. Since the encoding fed to Gurobi is exactly the same fed to the other solvers based on MiniZinc, we can exclude errors in the encoding while we investigate other possible causes. ECL^{i}PS^e results are mixed, i.e., some setups are solved faster than OR-Tools or z3, others take more than two seconds to solve. OptiMathSat is surprisingly slow on these encodings: if we exclude scenarios for which no feasible configuration exists, then OptiMathSat best result is 30 seconds to solve the 1300 × 900 setup.

When considering the full encoding, we limit our comparison to z3 and OptiMathSat, since they are the only ones that appear to handle encodings which contain a substantial part of arithmetic over reals, as it is required to take into account cylinder selection, sizing and computation of forces on the car rails. Among the MiniZinc-based tools, ECL^{i}PS^e is meant to support arithmetic over reals, but even the baseline encoding with relaxations resulted in a timeout for every setup other than the ones for which no feasible configuration exists. In Table 5.3 we collect the results of the comparison between z3 and OptiMathSat, adding the runtime of the heuristic search performed by LIFTCREATE for reference. We focus on the

implication-based encoding given the results with the baseline encoding. In the table, columns labeled "**I**" report runtimes on the integer-based versions, columns labeled "**I+R**" report runtimes on relaxed versions, and columns labeled "**I+R+F**" report runtimes on versions where look-up tables are represented as nested *if-then-else* functions rather than straight implications. The columns "**SO**" and "**MO**" report the results of single-objective and multi-objective optimization, respectively. In the single-objective case, considering equation (5.11), we set the free parameters α_1, α_2 and α_3 to 0.3 and α_4 to 0.1 in order to encode different priorities. In the multi-objective encoding, we set all weights to one.

Considering the results in Table 5.3, we see that the integer-based version of the full encoding is the least appealing option: while z3 performs slightly better than OptiMathSat on this version, other solutions yield faster runtimes. In particular, relaxing the encoding has a substantial impact both on z3 and OptiMathSat: solving time decreases by orders of magnitude in some cases with respect to the integer-based encoding. Finally, considering the addition of native SMT-LIB functions we see that the results are mixed, i.e., it is not so clear that choosing them improves the solving time. Noticeably, while OptiMathSat remains slower than z3, it never exceeds the time limit on this encoding. As for single vs. multi-objective encoding, we can see that the multi-objective approach performs better than the single-objective one. In spite of some exceptions, multi-objective optimization — specifically, with z3, relaxed encodings and native SMT-LIB function — seems to be the winning option overall. When it comes to comparing the heuristic engine of LIFTCREATE with the best results of the constraint-based approach, we should take into account that the former deals with the *complete* design cycle and not just with some subtasks. Given this initial bias, that in some cases the heuristic engine outperforms most constraint-based solutions, but it is overall slower than the best ones, it is fair to say that OMT solvers with relaxed encodings and multi-objective optimization provide a feasible replacement to heuristic search in the design subtasks that we considered here.

Chapter 6

Conclusions and future research

The outcome of this research is a set of enabling technologies to develop and test a novel design tool.
The research dealt with an multidisciplinary approach to the daunting task of design and verification. The objective to support the engineering process of a complex product was carried out by testing several different approaches.
All the process to define requirements, implement and test solution had a strong impact on efforts invested in this research. In this scenario the declarative techniques had an edge over, for example, meta-heuristics approach.
Considering the current research and industry scenario, where implementation of algorithms, tools, solvers and computational capability are a widespread available resource, possible and viable extension to this research can be the study of techniques to optimize the problem definition and encoding.
Professional engineers strive for a computational approach to design but, except for some very advanced solutions, tools are somehow related to technical legacies or to partial, non-integrated solution.
A further topic would be about the generalization of the techniques introduced, expanding the fields of applications to more than just elevator design. Ideally, starting from a specific normative, which represents a structured and reliable source of information, an interesting extension would be the research for an automated elaboration to obtain a first semantic definition of the problem.

Bibliography

1. Linda L Zhang. Product configuration: a review of the state-of-the-art and future research. *International Journal of Production Research*, 52(21):6381–6398, 2014.
2. Timo Soininen, Juha Tiihonen, Tomi Männistö, and Reijo Sulonen. Towards a general ontology of configuration. *AI EDAM*, 12(04):357–372, 1998.
3. Marco Menapace and Armando Tacchella. Ontologies in system engineering: a field report. *CoRR*, abs/1702.07193, 2017.
4. Juha Tiihonen, Alexander Felfernig, and Monika Mandl. Personalized configuration. *Knowledge-based Configuration–From Research to Business Cases. Morgan Kaufmann Publishers, Waltham, MA*, pages 167–179, 2014.
5. Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner. Modeling and solving technical product configuration problems. *AI EDAM*, 25(2):115–129, 2011.
6. Leopoldo Annunziata, Marco Menapace, and Armando Tacchella. Computer intensive vs. heuristic methods in automated design of elevator systems. In *31th European Conference on Modelling and Simulation, ECMS 2017, Budapest, Hungary, May 23 - May 27, 2017, Proceedings.*, pages 543 – 549, 2017.
7. Stefano Demarchi, Marco Menapace, and Armando Tacchella. Automating elevator design with satisfiability modulo theories. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 26–33. IEEE, 2019.
8. Robin T. Bye, Ottar L. Osen, Birger Skogeng Pedersen, Ibrahim A. Hameed, and Hans Georg Schaathun. A software framework for intelligent computer-automated product design. In *30th European Conference on Modelling and Simulation, ECMS 2016, Regensburg, Germany, May 31 - June 3, 2016, Proceedings.*, pages 534–543, 2016.

9. Louis A. Kamentsky and Chao-Ning Liu. Computer-automated design of multifont print recognition logic. *IBM Journal of Research and Development*, 7(1):2–13, 1963.

10. R Venkata Rao and Vimal J Savsani. *Mechanical design optimization using advanced optimization techniques*. Springer Science & Business Media, 2012.

11. F Zorriassatine, C Wykes, R Parkin, and N Gindy. A survey of virtual prototyping techniques for mechanical product development. *Proceedings of the institution of mechanical engineers, Part B: Journal of engineering manufacture*, 217(4):513–530, 2003.

12. Rebekka Volk, Julian Stengel, and Frank Schultmann. Building information modeling (bim) for existing buildings—literature review and future needs. *Automation in construction*, 38:109–127, 2014.

13. Aníbal De Almeida, Simon Hirzel, Carlos Patrão, João Fong, and Elisabeth Dütschke. Energy-efficient elevators and escalators in europe: An analysis of energy efficiency potentials and policy measures. *Energy and Buildings*, 47:151–158, 2012.

14. Lawrence Davis. *Handbook of genetic algorithms*. CUMINCAD, 1991.

15. Abdollah Homaifar, Charlene X Qi, and Steven H Lai. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–253, 1994.

16. Stefano Demarchi, Marco Menapace, and Armando Tacchella. Automating elevator design with satisfiability modulo theories. In *IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, Oregon, November 4-6, 2019, Proceedings*, 2019. Accepted for publication.

17. Susan E Carlson. Genetic algorithm attributes for component selection. *Research in Engineering Design*, 8(1):33–51, 1996.

18. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: from an abstract davis–putnam–logemann–loveland procedure to dpll (t). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

19. Roberto Sebastiani and Patrick Trentin. On optimization modulo theories, maxSMT and sorting networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 231–248. Springer, 2017.

20. Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

21. Kim Marriott, Peter J Stuckey, LD Koninck, and Horst Samulowitz. A minizinc tutorial, 2014.

22. Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

23. Kuang-Hua Chang. Chapter 19 - multiobjective optimization and advanced topics. In Kuang-Hua Chang, editor, *e-Design*, pages 1105 – 1173. Academic Press, Boston, 2015.

24. Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

25. Roberto Sebastiani and Patrick Trentin. Optimathsat: A tool for optimization modulo theories. *Journal of Automated Reasoning*, Dec 2018.

26. Geoffrey Chu. Improving combinatorial optimization. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

27. Laurent Perron and Vincent Furnon. Or-tools, 2020.

28. Joachim Schimpf and Kish Shen. Eclipse – from lp to clp. *Theory and Practice of Logic Programming*, 12(1-2):127–156, 2012.

29. IBM. Ibm ilog cplex optimization studio (2017) cplex users manual, version 12.7, 2017.

30. Zonghao Gu, Edward Rothberg, and Robert Bixby. Gurobi optimization, 2019.

Chapter 7

Appendix

Symbol	Description
x_{cf}, y_{cf}	Car frame base point coordinates
x_{cd}, y_{cd}	Car door base point coordinates
x_{ld}, y_{ld}	Landing door base point coordinates
x_{car}, y_{car}	Car base point coordinates
w_{car}, d_{car}	Car width and depth

Table 7.1: Explanation of the decision variables involved in the design process

Symbol	Description
x_{shaft}, y_{shaft}	Shaft base point coordinates
w_{shaft}, d_{shaft}	Shaft width and depth
$red_{[N,E,S,W]}$	Distance between shaft and car walls (North, East, South, West)
$cwt_{[N,E,S,W]}$	Car wall thickness (North, East, South, West)
w_{cf}	Distance from x_{cf} to the left car wall
d_{cf}	External distance between car frame rails
y_{gear}	Core gear placement with respect to the car frame base point
d_{br}	External depth of the car frame bracket from the base point
dcr	Distance between car rails
max_{oh}	Maximum car overhang that the car frame is able to sustain
$opening$	Doors opening
la_{cd}, ra_{cd}	Left and right axis size (car door)
la_{ld}, ra_{ld}	Left and right axis size (landing door)
$step_{cd}$	Car door step
$step_{ld}$	Landing door step
d_{step}	Distance between doors
w_{frame}	Landing door external frame width

Table 7.2: Explanation of the parameters involved in the design process

Shaft	TIME	CONF
1300 × 800	1271	0
1300 × 850	868	54
1300 × 900	731	54
1300 × 950	688	69
1300 × 1000	715	51
1300 × 1050	766	74
1300 × 1100	633	89
1300 × 1150	607	92
1300 × 1200	633	168
1300 × 1250	598	197
1300 × 1300	764	397
1300 × 1350	716	565
1300 × 1400	1062	679
1300 × 1450	953	819
1300 × 1500	966	859

Shaft size	TIME	CONF
1500 × 800	1213	0
1500 × 850	1304	54
1500 × 900	1250	80
1500 × 950	927	160
1500 × 1000	1330	160
1500 × 1050	1354	180
1500 × 1100	1268	198
1500 × 1150	1448	134
1500 × 1200	1544	414
1500 × 1250	1520	460
1500 × 1300	1742	920
1500 × 1350	1683	1118
1500 × 1400	1823	1179
1500 × 1450	1591	1253
1500 × 1500	1548	986

Table 7.3: Computing configurations with heuristic techniques (LIFTCREATE-HR): "TIME" is the total runtime in milliseconds, "CONF" is the total number of feasible configurations found (at most one for each prototype).

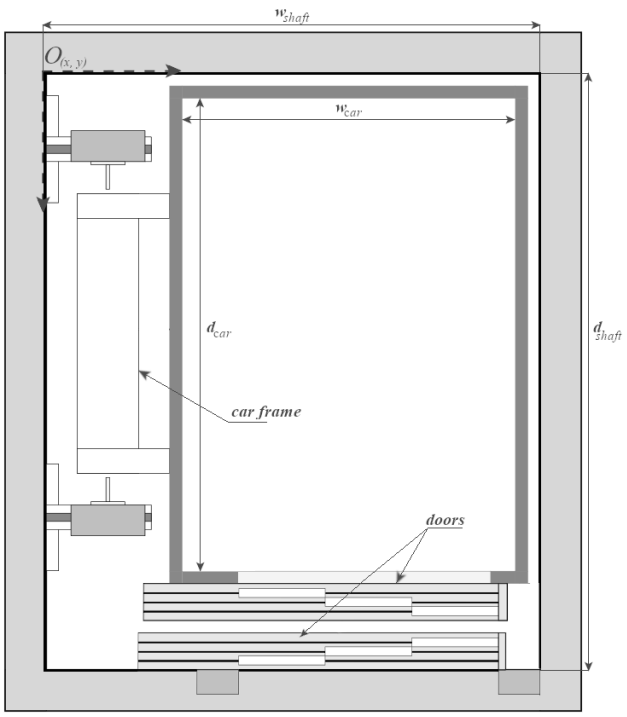

Figure 7.1: Cross-section (plan view) of a configured RHE. The shaft is the gray box surrounding the other components, the car frame is on the left side and doors at the bottom of the drawing.

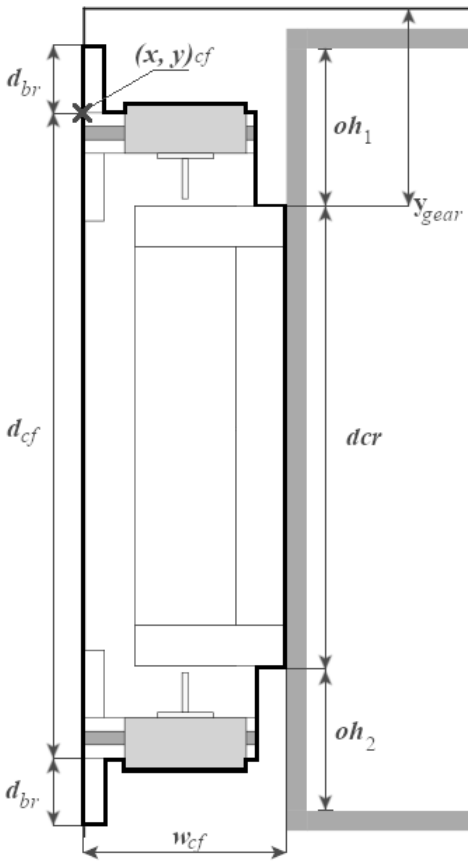

Figure 7.2: Detail of the car frame and related parameters.

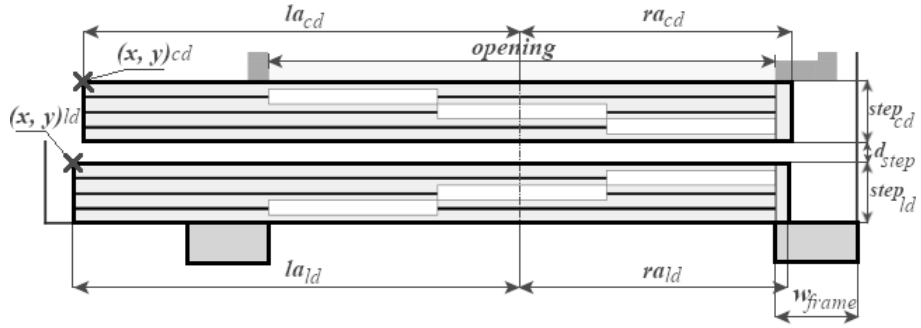

Figure 7.3: Detail of the car/landing door pair and related parameters.

Shaft	LIFTCREATE-BF							
	TTP	**TTE**	**TTV**	**TOT**	**E**	**V**	**C**	**C ∩ H**
1300×800	297	0	0	297	0	0	0	0
1300×850	706	445	24377	25528	5337225	187122	69	54
1300×900	484	871	53434	54789	13046550	618942	69	54
1300×950	369	1438	88256	90063	20953550	638756	84	69
1300×1000	621	2475	150499	153595	34197775	429037	56	51
1300×1050	598	2955	190942	194495	48035025	204131	79	74
1300×1100	671	4305	268022	272998	68988575	289941	95	89
1300×1150	684	5491	348839	355014	90732825	217301	95	92
1300×1200	780	7721	491018	499519	121372450	1048027	188	168
1300×1250	809	9620	632078	642507	153000450	1862813	217	197
1300×1300	982	12236	785448	798666	195302900	3196098	440	397
1300×1350	961	15071	954582	970614	238791400	5037710	623	565
1300×1400	1102	17352	1135655	1154109	282279900	6034361	724	679
1300×1450	1692	20692	1381459	1403843	331105625	7196113	876	819
1300×1500	1710	23609	1582480	1607799	380524375	8099000	886	859
1500×800	214	0	0	214	0	0	0	0
1500×850	452	674	46060	47186	11208915	311224	80	54
1500×900	624	1685	111249	113558	27399570	1154312	108	80
1500×950	485	2851	180057	183393	44005370	1580060	196	160
1500×1000	718	4355	293928	299001	71820085	1552741	175	160
1500×1050	959	6251	424036	431246	100880235	1037319	191	180
1500×1100	1382	8981	576262	586625	144885605	1311244	229	198
1500×1150	1171	12186	776171	789528	190551555	880694	152	134
1500×1200	1876	15588	973855	991319	254899030	3133714	535	414
1500×1250	1952	20302	1295708	1317962	321322230	5637692	536	460
1500×1300	2504	24886	1589546	1616936	410163260	9692307	1106	920
1500×1350	2447	31104	1963186	1996737	501495160	15565594	1268	1118
1500×1400	-	-	-	-	-	-	-	-
1500×1450	-	-	-	-	-	-	-	-
1500×1500	-	-	-	-	-	-	-	-

Table 7.4: Computing configurations with brute-force techniques (LIFTCREATE-BF): "TTP" is the time to compute prototypes, "TTE" is the time to compute early designs, "TTV" is the time spent for verification, and "TOT" is the total runtime (all times in milliseconds); "E" is the total number of early designs found, and "V" is the number of valid configurations; "C" is the number of clusters and "C ∩ H" is the number of clusters shared by brute-force and LIFTCREATE-HR; dashes indicate that the corresponding setting has exceeded the time limit of 30 minutes.

Shaft	LIFTCREATE-HR	LIFTCREATE-RS			LIFTCREATE-GA		
		$\mathbf{MED_T}$	$\mathbf{IQR_T}$	**SIM**[%]	$\mathbf{MED_T}$	$\mathbf{IQR_T}$	**SIM**[%]
1300 × 800	1271	85	9	–	0	0	–
1300 × 850	868	2600.5	42.5	78.26	725	209.5	100
1300 × 900	731	5815.5	124.25	78.26	1262.5	23.25	100
1300 × 950	688	9088	221	82.14	1976.5	35.5	97.37
1300 × 1000	715	15861.5	284.75	91.07	2627	26.75	94.12
1300 × 1050	766	21515	320.5	93.67	3328.5	59.5	100
1300 × 1100	633	30227	424.75	93.62	5312	380.5	100
1300 × 1150	607	39118	438	96.74	5983	401	100
1300 × 1200	633	54453	607.5	89.36	5398.5	84.75	100
1300 × 1250	598	69892	818.75	90.78	7166.5	161.5	100
1300 × 1300	764	86808.5	951.5	90.21	7554	208	96.67
1300 × 1350	716	104382.5	1119.75	90.69	8605.5	174	96.3
1300 × 1400	1062	125076	1083.75	93.78	8876	203.75	100
1300 × 1450	953	146872.5	1420	93.46	10452.5	287.25	100
1300 × 1500	966	168864	1527.5	96.9	11617	212.75	100
1500 × 800	1213	124.5	7.75	–	0	0	–
1500 × 850	1304	5300	97.75	67.5	1047	45	71.43
1500 × 900	1250	12448	250.75	74.07	1801.5	29.5	78.13
1500 × 950	927	19930.5	253.75	81.63	2537.5	35.75	83.78
1500 × 1000	1330	32566	595	91.43	3186.5	65	97.92
1500 × 1050	1354	46450	415.25	94.71	3821.5	49.75	84.21
1500 × 1100	1268	63574	839	86.46	5982	98.5	100
1500 × 1150	1448	78978	810.75	88.16	6561.5	84	100
1500 × 1200	1544	106945.5	954.25	77.53	7906	157.75	89.39
1500 × 1250	1520	139008.5	954	85.82	9088.5	240	98.55
1500 × 1300	1742	173950	1692.5	83.24	11264	373.75	100
1500 × 1350	1683	213649	1965	88.17	12597.5	230.75	100
1500 × 1400	1823	245340	2487.5	94.4	12134	290.75	100
1500 × 1450	1591	280193	2299.75	93.58	13213.5	419	100
1500 × 1500	1548	303919.5	2365.5	94.99	14653.5	703.5	100

Table 7.5: Comparison among LIFTCREATE-HR, LIFTCREATE-RS and LIFTCREATE-GA (mutation rate 10%) across different setups. For LIFTCREATE-HR we report the total runtime. For LIFTCREATE-RS and LIFTCREATE-GA we report median ("$\mathbf{MED_T}$") and interquartile range ("$\mathbf{IQR_T}$") of their runtime, as well as a similarity ("SIM[%]") measure with respect to LIFTCREATE-HR, i.e., the ratio $\frac{\text{MED}_\text{I}}{\text{MED}_\text{C}}$ reported in Table 7.6 (all runtimes in milliseconds).

Shaft	LIFTCREATE-RS								
	P	E	V		C		I=C ∩ H		$\frac{MED_I}{MED_C}$[%]
			MED_V	IQR_V	MED_C	IQR_C	MED_I	IQR_I	
1300 × 800	0	0	0.0	0.0	0.0	0.0	0.0	0.0	–
1300 × 850	414	533559	18723.5	177.5	69.0	0.0	54.0	0.0	78.26
1300 × 900	552	1304437	61937.5	295.5	69.0	0.0	54.0	0.0	78.26
1300 × 950	552	2095137	63868.0	229.5	84.0	0.0	69.0	0.0	82.14
1300 × 1000	966	3419396	42872.0	310.0	56.0	0.0	51.0	0.0	91.07
1300 × 1050	966	4803121	20375.5	163.25	79.0	1.0	74.0	1.0	93.67
1300 × 1100	1518	6898258	29032.0	226.75	94.0	1.0	88.0	1.0	93.62
1300 × 1150	1518	9072683	21733.5	174.75	92.0	1.0	89.0	1.0	96.74
1300 × 1200	2208	12136373	104783.5	329.75	188.0	0.0	168.0	0.0	89.36
1300 × 1250	2208	15299173	186307.0	489.75	217.0	0.0	197.0	0.0	90.78
1300 × 1300	3036	19529091	319652.5	790.0	439.0	1.0	396.0	1.0	90.21
1300 × 1350	3036	23877941	503584.0	680.75	623.0	0.0	565.0	0.0	90.69
1300 × 1400	3036	28226791	603468.0	812.75	723.0	1.0	678.0	1.0	93.78
1300 × 1450	3450	33109200	719755.5	687.5	872.0	2.75	815.0	2.75	93.46
1300 × 1500	3450	38051075	809984.5	1085.75	872.0	3.0	845.0	3.0	96.90
1500 × 800	0	0	0.0	0.0	0.0	0.0	0.0	0.0	–
1500 × 850	687	1120626	31124.0	191.25	80.0	0.0	54.0	0.0	67.50
1500 × 900	916	2739603	115395.5	458.75	108.0	0.0	80.0	0.0	74.07
1500 × 950	916	4400183	157998.0	367.5	196.0	0.0	160.0	0.0	81.63
1500 × 1000	1603	7181389	155253.0	388.5	175.0	0.0	160.0	0.0	91.43
1500 × 1050	1603	10087404	103745.0	486.0	189.0	1.0	179.0	1.0	94.71
1500 × 1100	2519	14487587	131107.0	493.75	229.0	0.0	198.0	0.0	86.46
1500 × 1150	2519	19054182	88101.0	336.25	152.0	0.0	134.0	0.0	88.16
1500 × 1200	3664	25488487	313411.0	562.25	534.0	2.0	414.0	0.0	77.53
1500 × 1250	3664	32130807	563689.0	721.25	536.0	0.0	460.0	0.0	85.82
1500 × 1300	5038	41014379	969265.0	1139.25	1104.0	1.0	919.0	1.0	83.24
1500 × 1350	5038	50147569	1556467.0	1299.25	1268.0	0.0	1118.0	0.0	88.17
1500 × 1400	5038	59280759	1660048.0	1416.25	1249.0	0.0	1179.0	0.0	94.40
1500 × 1450	5725	69534575	1784513.0	1377.5	1339.0	1.0	1253.0	0.0	93.58
1500 × 1500	5725	79913200	1666806.0	1580.0	1037.0	0.75	985.0	0.75	94.99

Table 7.6: Computing configurations with random sampling techniques (LIFTCREATE-RS): column "P" is the number of prototypes and "E" is the number of early designs sampled. The pair of columns "V" is the number of valid configurations found, "C" is the number of clusters, "I=C ∩ H" is the number of clusters shared by LIFTCREATE-RS and LIFTCREATE-HR. For each pair, column "MED_x" and "IQR_x" are the median and the interquartile range of value x, respectively. Column "$\frac{MED_I}{MED_C}$" is the ration between shared clusters and LIFTCREATE-RS ones.

Shaft	LIFTCREATE-RS							
	TTP		TTE		TTV		TOT	
	$\mathbf{MED_P}$	$\mathbf{IQR_P}$	$\mathbf{MED_E}$	$\mathbf{IQR_E}$	$\mathbf{MED_V}$	$\mathbf{IQR_V}$	$\mathbf{MED_T}$	$\mathbf{IQR_T}$
1300 × 800	85.0	9.0	0.0	0.0	0.0	0.0	85.0	9.0
1300 × 850	273.0	14.0	81.0	10.75	2248.5	44.75	2600.5	42.5
1300 × 900	350.5	15.5	199.0	16.5	5264.5	96.5	5815.5	124.25
1300 × 950	369.0	13.5	329.5	16.5	8392.0	217.25	9088.0	221.0
1300 × 1000	566.0	15.0	541.0	34.0	14753.5	262.75	15861.5	284.75
1300 × 1050	564.0	22.5	779.5	28.0	20161.5	312.25	21515.0	320.5
1300 × 1100	828.0	30.75	1111.5	27.25	28249.0	387.0	30227.0	424.75
1300 × 1150	825.0	27.75	1477.5	44.0	36818.5	427.0	39118.0	438.0
1300 × 1200	1123.0	27.25	2020.0	49.5	51294.0	572.5	54453.0	607.5
1300 × 1250	1127.0	38.25	2574.5	71.75	66177.5	756.25	69892.0	818.75
1300 × 1300	1481.5	25.5	3290.0	51.25	81986.5	904.5	86808.5	951.5
1300 × 1350	1485.0	21.0	4060.0	75.5	98861.0	1057.25	104382.5	1119.75
1300 × 1400	1485.0	27.0	4848.0	69.0	118728.5	1025.5	125076.0	1083.75
1300 × 1450	1670.0	38.5	5753.5	97.75	139428.0	1327.25	146872.5	1420.0
1300 × 1500	1668.0	36.5	6672.0	102.5	160559.0	1450.0	168864.0	1527.5
1500 × 800	124.5	7.75	0.0	0.0	0.0	0.0	124.5	7.75
1500 × 850	437.5	11.75	163.0	17.75	4699.5	88.25	5300.0	97.75
1500 × 900	570.0	26.25	421.5	28.5	11418.5	218.0	12448.0	250.75
1500 × 950	598.0	23.75	701.5	35.75	18654.5	271.5	19930.5	253.75
1500 × 1000	919.5	20.25	1157.5	31.5	30468.5	580.5	32566.0	595.0
1500 × 1050	925.0	14.75	1653.5	40.25	43891.5	432.5	46450.0	415.25
1500 × 1100	1346.0	29.75	2381.5	71.75	59904.5	823.0	63574.0	839.0
1500 × 1150	1350.0	26.5	3148.5	53.75	74468.0	780.25	78978.0	810.75
1500 × 1200	1841.5	53.25	4257.0	82.75	100842.5	875.5	106945.5	954.25
1500 × 1250	1850.0	39.5	5434.5	72.25	131644.0	887.25	139008.5	954.0
1500 × 1300	2456.0	48.75	6993.5	75.0	164561.0	1592.0	173950.0	1692.5
1500 × 1350	2448.0	40.5	8699.0	94.5	202574.5	1875.5	213649.0	1965.0
1500 × 1400	2455.0	52.75	10314.0	136.0	232618.5	2497.5	245340.0	2487.5
1500 × 1450	2764.0	52.5	12216.0	173.5	265170.5	2146.25	280193.0	2299.75
1500 × 1500	2754.5	53.0	14103.5	150.75	287098.5	2341.75	303919.5	2365.5

Table 7.7: Computing configurations with random sampling techniques (LIFTCREATE-RS): the column pair "TTP" is the time to compute prototypes, "TTE" is the time to compute early designs, "TTV" is the time spent for verification, and "TOT" is the total runtime (all times in milliseconds); for each pair, column "MED_x" and "IQR_x" are the median and the interquartile range of value x, respectively.

Shaft	Result	Time		Shaft	Result	Time	
		SMT	OMT			SMT	OMT
1300 × 800	U	150	226	1500 × 800	U	98	130
1300 × 850	S	112	361	1500 × 850	S	115	412
1300 × 900	S	106	434	1500 × 900	S	108	807
1300 × 950	S	91	691	1500 × 950	S	95	773
1300 × 1000	S	92	402	1500 × 1000	S	95	633
1300 × 1050	S	94	620	1500 × 1050	S	96	780
1300 × 1100	S	97	557	1500 × 1100	S	100	645
1300 × 1150	S	93	416	1500 × 1150	S	95	674
1300 × 1200	S	97	381	1500 × 1200	S	109	461
1300 × 1250	S	97	519	1500 × 1250	S	96	439
1300 × 1300	S	93	622	1500 × 1300	S	99	444
1300 × 1350	S	94	477	1500 × 1350	S	106	395
1300 × 1400	S	95	421	1500 × 1400	S	107	470
1300 × 1450	S	94	593	1500 × 1450	S	98	499
1300 × 1500	S	95	707	1500 × 1500	S	94	477

Table 7.8: Computing configurations with SMT techniques (LIFTCREATE-SMT): **"Result"** is the answer given by the solver and can be either SAT (S) or UNSAT (U). **"SMT"** time is related to feasibility checking and **"OMT"** time is related to optimality. All times are in milliseconds.

Shaft	LIFTCREATE-HR	LIFTCREATE-GA			LIFTCREATE-SMT	
		SIM%	Time		Result	Time
			MED_T	IQR_T		
1300 × 800	1271	–	0	0	U	226
1300 × 850	868	100	725	209.5	S	361
1300 × 900	731	100	1262.5	23.25	S	434
1300 × 950	688	97.37	1976.5	35.5	S	691
1300 × 1000	715	94.12	2627	26.75	S	402
1300 × 1050	766	100	332.5	59.5	S	620
1300 × 1100	633	100	5312	380.5	S	557
1300 × 1150	607	100	5983	401	S	416
1300 × 1200	633	100	5398.5	84.75	S	381
1300 × 1250	598	100	7166.5	161.5	S	519
1300 × 1300	764	96.67	7554	208	S	622
1300 × 1350	716	96.3	8605.5	174	S	477
1300 × 1400	1062	100	8876	203.75	S	421
1300 × 1450	953	100	10452.5	287.25	S	593
1300 × 1500	966	100	11617	212.75	S	707
1500 × 800	1213	–	0	0	U	130
1500 × 850	1304	71.43	1047	45	S	412
1500 × 900	1250	78.13	1801.5	29.5	S	807
1500 × 950	927	83.78	2537.5	35.75	S	773
1500 × 1000	1330	97.92	3186.5	65	S	633
1500 × 1050	1354	84.21	3821.5	49.75	S	780
1500 × 1100	1268	100	5982	98.5	S	645
1500 × 1150	1448	100	6561.5	84	S	674
1500 × 1200	1544	89.39	7906	157.5	S	461
1500 × 1250	1520	98.55	9088.5	240	S	439
1500 × 1300	1742	100	11264	373.75	S	444
1500 × 1350	1683	100	12597.5	230.75	S	395
1500 × 1400	1823	100	12134	290.75	S	470
1500 × 1450	1591	100	13213.5	419	S	499
1500 × 1500	1548	100	14653.5	703.5	S	477

Table 7.9: Comparison among the three techniques proposed, namely LIFTCREATE-HR, LIFTCREATE-GA (mutation rate 10%) and LIFTCREATE-SMT. All times are in milliseconds.