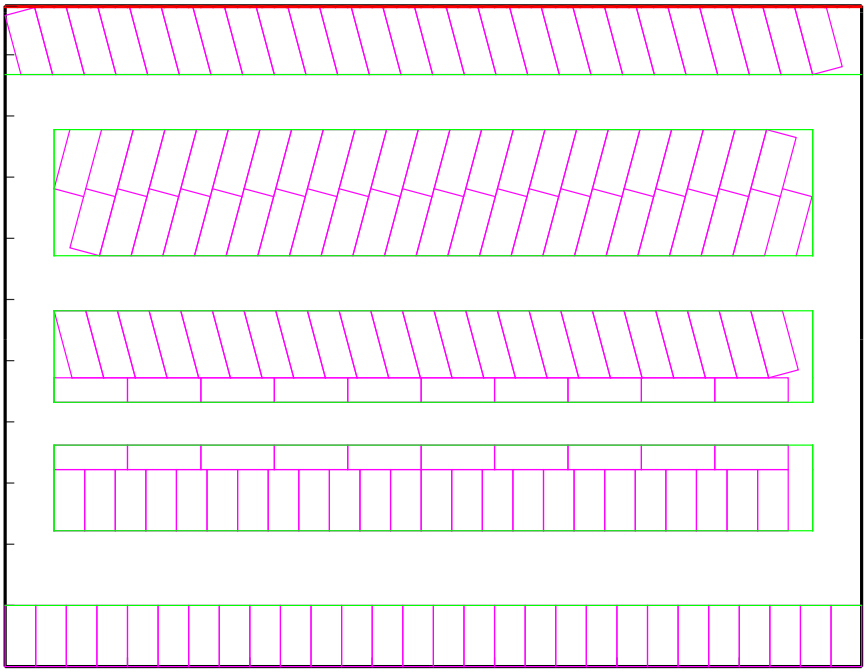
CHALMERS
UNIVERSITY OF TECHNOLOGY



Optimisation of Parking Layout

A Mixed Integer Linear Programming Formulation for Maximum Number of Parking Spots, Applicable for Evaluation of Autonomous Parking Benefits

Master's Thesis within the division of Systems, Control and Mechatronics

MALIN KARLSSON
RICHARD PETERSSON

EX033/2016

Department of Signals and Systems (S2)
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis

Optimisation of Parking Layout

A Mixed Integer Linear Programming Formulation for Maximum Number of Parking Spots, Applicable for Evaluation of Autonomous Parking Benefits

MALIN KARLSSON
RICHARD PETERSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2016

Optimisation of Parking Layout
A Mixed Integer Linear Programming Formulation for Maximum Number of Parking Spots, Applicable for Evaluation of Autonomous Parking Benefits
MALIN KARLSSON
RICHARD PETERSSON

Supervisors: Martin Fabian, Department of Signals and Systems, Chalmers University of Technology;
Nenad Lazic, Active Safety, Volvo Car Group
Examiner: Martin Fabian, Department of Signals and Systems

Master's Thesis EX033/2016
Department of Signals ans Systems (S2)
Division of Systems, Control and Mechatronics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Parking layout generated by pattern formulation program written in MATLAB, showing an unintuitive yet optimal parking layout.

Typeset in LaTeX
Gothenburg, Sweden 2016

Optimisation of Parking Layout
A Mixed Integer Linear Programming Formulation for Maximum Number of Parking Spots, Applicable for Evaluation of Autonomous Parking Benefits
MALIN KARLSSON
RICHARD PETERSSON
Department of Signals and Systems
Chalmers University of Technology

Abstract

One of motorism's greatest challenges is that it takes up a lot of urban space per passenger. Volvo Car Group is working together with the City of Gothenburg to investigate how autonomous cars and autonomous parking can affect cars' impact on urban environments. The goal of this thesis is to, with mathematical methods, optimise the distribution of parking spots in a two-dimensional parking area in order to maximise the number of spots. To facilitate comparison between regular and autonomously parking cars, a program to design parking layouts is developed.

The thesis considers two independent approaches to parking spot placement. The first approach is a Mixed Integer Linear Programming (MILP) formulation of the optimisation of spot placement. In this approach, each parking spot is placed individually. However, this turns out to be computationally expensive to such an extent that a second approach is required. This second, and main, approach is a pattern-based MILP formulation that considers rows of parking spots as items. Optimality is found for the vertical distribution of horizontal rows. The pattern-based approach also includes heuristics in order to choose the order of parking rows with respect to vertical roads. The pattern-based approach is analysed for potential benefits of autonomous parking with values based on standard dimensions used by the Gothenburg City Parking Company. The spot placement formulation is not analysed.

Using data from the Gothenburg City Parking Company, we show that although narrower parking spots require more manoeuvre space, there is an increase in area utilisation from using narrower parking spots.

Keywords: optimisation, mixed integer linear programming, MILP, parking layout, autonomous parking, packing problem, permutations analysis, two-dimensional packing problem.

Dedicated to Adolph Lohström,

who showed the importance of camaraderie in successful school attendance, through his performance as a student at Chalmers University of Technology. Without owning the physical conditions of his comrades, he managed to achieve exemplary attendance and also had time to engage in the student union-life. His example is proof that an active social life and good study presence can exist in harmony, at least in theory.

Acknowledgements

We would like to sincerely thank our supervisor and examiner at Chalmers University of Techonlogy, Martin Fabian, for pleasant meetings and patient guidance throughout the whole thesis. We would also like to thank Nenad Lazic, our Supervisor at Volvo Car Group, for his support and joyful comments.

Further, we would like to give Oskar Wigström a special thank, for guidance during a period when we were stuck. Also, Marcus Rothoff from Volvo Car Group deserves to be acknowledged, as well as Stefan Gröndahl, Maria Berntsson, Anna Svensson and the rest of all the helpful people at the Gothenburg City Parking Company and the City Planning Office of Gothenburg.

Our squad of opponents, Albin Hjalmarsson, Henrik Johansson, Madeleine Yttergren and Anton Zita, we thank for their helpful contribution in reviewing and improving our work.

The project would not had been this fun without the fika group of thesis workers at Volvo; Erik Henriksson, Viktor Kardell, Pontus Petersson and again Anton Zita. Finally, we thank our family and friends, who helped us reach here and all the way through the thesis.

Malin Karlsson and Richard Petersson, Gothenburg, June, 2016

Contents

Acronyms xv

Variable Notations xvii

List of Figures xix

List of Tables xxi

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Problem Definition . . . 2
- 1.3 Purpose and Objective . . . 2
- 1.4 Delimitations . . . 4
  - 1.4.1 General Delimitations . . . 4
  - 1.4.2 Program Delimitations . . . 4
- 1.5 Solution Overview . . . 5
- 1.6 Thesis Organisation . . . 6

2 Theory 7
- 2.1 Computational Complexity . . . 7
- 2.2 Optimisation Methods . . . 8
  - 2.2.1 Mixed Integer Linear Programming . . . 8
  - 2.2.2 Constraint Programming . . . 9
  - 2.2.3 Stochastic Optimisation . . . 10
- 2.3 Optimisation Theory . . . 10
- 2.4 Classical Problems . . . 11
- 2.5 Technical Specifications . . . 13

3 Related Work 17

I Spot Formulation 21

4 Spot Formulation 23
- 4.1 Overview . . . 23
- 4.2 Delimitations . . . 25
- 4.3 Optimisation Method . . . 26

5 MILP Formulation **27**
5.1 Objective Function . . . 27
5.2 Car Park Bounds . . . 27
5.3 Shelfing . . . 28
5.4 Lowest Index Placement . . . 29
5.5 Overlap Avoidance . . . 29
5.6 Manoeuvre Space . . . 29

6 Computational Complications **33**

II Pattern Generation **35**

7 Pattern Formulation **37**
7.1 Overview . . . 37
7.1.1 Manoeuvre Rows . . . 38
7.1.2 Double Rows . . . 38
7.1.3 Result Generation . . . 40
7.2 Optimisation Method . . . 40
7.3 Indata . . . 40
7.4 XML Reading . . . 41
7.5 Solvers . . . 42

8 MILP Formulation **43**
8.1 Sets . . . 43
8.2 Objective Function . . . 43
8.3 Total Length . . . 43
8.4 Manoeuvre Rows . . . 44
8.5 Double Rows . . . 44

9 Post Processing **47**
9.1 Permutations . . . 47
9.2 Mirroring Tilted Parking Rows . . . 49
9.3 Vertical Roads . . . 49
9.4 Presentation of Results . . . 50
9.5 Iteration . . . 51

10 Results and Analysis **53**
10.1 Result Generation Data . . . 53
10.2 Parking Spot Reductions . . . 53
10.2.1 Horizontal Layout - Skeppsbron . . . 55
10.2.2 Vertical Layout . . . 59
10.2.3 Square Layout . . . 61
10.2.4 Impact of Orientation . . . 61
10.2.5 General Trends . . . 64
10.3 Car Park Dimension Iteration . . . 64
10.4 Counterintuitive Layouts . . . 65

11 Conclusion and Discussion **69**
11.1 Objectives 69
11.1.1 Mathematical Methods 69
11.1.2 Applicability for Autonomous and Manual Parking 69
11.1.3 Program Output 70
11.2 Results 70
11.3 Future Work 71
11.3.1 Adapting to Heuristics 71
11.3.2 Additional Functions 71
11.3.3 Obstacles 72
11.3.4 Spot Formulation 72
11.3.5 Ethical and Sustainability Matters 73

Bibliography **77**

A Appendix 1 **I**

Acronyms

CP	Constraint Programming
IP	Integer Programming
LP	Linear Programming
MILP	Mixed Integer Linear Programming
NP	Non-deterministic Polynomial time
SO	Stochastic Optimisation

Variable Notations

Pattern Formulation

$length$	vertical length of the parking spot[1]
$lengthAcross$	length of the long side of the biggest possible rectangle within the parallelogram constituting the parking spot[1]
$manLength$	vertical length of the manoeuvre space[1]
φ	angle between the long side of a parking spot and the manoeuvre spaces[1]. Takes values in [0°,90°]
$spotsLost$	number of spots the solution is decreased with, when adding vertical roads
$type$	vector containing the types of spots, e.g. the different angles
$unusedLength$	vertical waste for spots where its angle is not 0° or 90°. In double rows this can be used to save vertical space[1]
$unusedWidth$	horizontal waste for spots where its angle is not 0° or 90°. Horizontal space that can not be of use for any spot[1]
$width$	horizontal width of a parking spot[1]
$widthAcross$	width measured as a line perpendicular to the long sides of a parking spot[1]

Spot Formulation

h_i	binary variable that takes value 1 if spot i is rotated by 90° (horizontal), and takes value 0 if spot i is rotated 0° (vertical)
l	length of a single spot
LW	the ratio of the width of a parking spot divided by the width of the whole parking area
M	Big M, a great enough value to activate or deactivate constraints
n_i	binary variable that takes value 1 if spot i is present in the solution, and takes value 0 otherwise

[1] see Figure 7.2

w	width of a single spot
$xGreat_{ij}$	binary variable that takes value 1 if the rightmost side of spot j is to the left of the leftmost side of spot i, and takes value 0 otherwise
x_i	x-value of spot i
$xLess_{ij}$	binary variable that takes value 1 if the leftmost side of spot j is to the left of the rightmost side of spot i, and takes value 0 otherwise
x_{wl}	x-value of the left wall
x_{wr}	x-value of the right wall
$yGreat_{ij}$	binary variable that takes value 1 if $xGreat_{ij}$ and $xLess_{ij}$ are both 1 (i.e. they are overlapping horizontally), and takes value 0 otherwise
y_i	y-value of spot i
y_{wb}	y-value of the bottom wall
y_{wt}	y-value of the upper wall

List of Figures

1.1 Black box illustration of the program. 3
1.2 System overview of inputs, functions, external tools and outputs. . . . 5

2.1 Example considering point (3,2), for which the constraint is inactive. 11
2.2 A closed solution space: The triangle enclosed by the constraints lines. 12
2.3 Convention for angle definition in parking design. 14
2.4 Illustrations of width and length for different car park orientations. . 15

3.1 Comparison of guillotine cuttable pattern, and non-guillotine cuttable pattern. 18

4.1 Single spot, $\varphi = 90°$. 23
4.2 Parking spots with different rotations. 23
4.3 Parking spots with different rotations and minimum distance to wall on the left side. 24
4.4 Five spots placed in the desired order, spot n_{i+1} is to the right of spot n_i if possible, otherwise above and to the far left. 25
4.5 Parking spots n_i and n_{i+1} placed with minimum distance to each other, with respect to the x-axis. 25

5.1 Shelfing. Three spots are placed, where spot 2 is above the dashed, grey line created from spot 1, and spot 3 is above the dashed line created from spot 2. 28
5.2 Variables $xLess_{ij}$, $xGreat_{ij}$ and $yGreat_{ij}$ depend on the horizontal placement of spots i and j. 30
5.3 Each spot is assured manoeuvre space on at least one side. This fails for parking spot 5 which is blocked by parking spot 6. 31

7.1 Example of a resulting parking layout, using the pattern formulation. 37
7.2 Parameter used to denote the dimensions of a tilted parking spot. . . 39
7.3 Vertical space saved by placing two spots of the same type adjacent to each other. 39
7.4 A double row, with single spots at both ends. 41

8.1 Two double rows, both requiring two halves of manoeuvre rows, yet can only share one manoeuvre row. 45

9.1 Tilted parking rows must create the same driving direction. 49

9.2 Two combinations of the same car park, 32 m by 51 m, with vertical roads. 50
9.3 Car park, 107 m by 87 m, with vertical roads, spots = 430. 51

10.1 Layout of the planned car park Skeppsbron, Göteborg. 55
10.2 A piece of the Skeppsbron layout, with no obstacles but columns. . . 56
10.3 Small Skeppsbron layout, using same dimensions as existing layout. . 56
10.4 Four resulting output layouts for the Small Skeppsbron layout. 57
10.5 Increase in spots and decrease in area per spot in Small Skeppsbron for decreasing values on *widthAcross* and *manLength*. 57
10.6 Two resulting output layouts for the Big Skeppsbron layout. 58
10.7 Increase in spots and decrease in area per spot in Big Skeppsbron for decreasing values on *widthAcross* and *manLength*. 58
10.8 Three layouts for the Vertical Layout when *widthAcross* is reduced and with *manLength* ±0° for all layouts. 59
10.9 Three layouts for the Vertical Layout. 60
10.10 Increase in spots and decrease in area per spot in the Vertical Layout for decreasing values on *widthAcross* and *manLength*. 60
10.11 Two resulting output layouts for the Square Layout. 61
10.12 Increase in spots and decrease in area per spot in the Square Layout for decreasing values on *widthAcross* and *manLength*. 62
10.13 Comparison between vertical and horizontal orientation for the Small Skeppsbron layout, vertical roads included and set to 4 m. 64
10.14 Percentual change in fitness values for varying values on *widthAcross* and *manLength*, a comparision between the four layouts. 65
10.15 Change in total number of spots, depending on the dimensions of the car park. 66
10.16 Change of length efficiency, *spots / required length*, for car park widths 1 m - 200 m. 67
10.17 Comparison between intuitive and counterintuitive layout for car park of size 20 m by 11.1 m. 68
10.18 Comparison between intuitive and counterintuitive layout for car park of size 60 m by 58 m. 68

A.1 Full drawing of the Skeppsbron layout. II

List of Tables

2.1 Comparison of computation time, depending on time complexity function, for different input sizes. 8

6.1 Binary decision variables generated for different values of n_{spots}. . . . 33

8.1 Variables and parameters used in the pattern formulation. 46

10.1 Values applied on the variables changed in the result generation. . . . 54
10.2 Dimensions for each parking spot in the Skeppsbron layout. 55
10.3 Results using the Vertical Layout, in its original orientation and a corresponding horizontal layout. 63
10.4 Results using the Small Skeppsbron layout, in its original orientation and in its corresponding vertical layout. 63
10.5 Dimensions on parking spot types used in this chapter. Data are taken from [22]. 67

1
Introduction

Urbanisation is globally growing and by 2045 the urban population is expected to surpass six billion [1]. As cities grow bigger and denser, more and more people are questioning the distribution of urban space between cars and people [2], [3]. Parking takes up a great amount of city area and Litman [2] estimates that modern cities devote at least as much area for roads and off-road parkings as for housing. When areas devoted to cars are to decrease, those existing need to be efficiently used. Optimising these spaces is a subject of interest today and with that comes the parking areas.

The task of distributing parking spots in a car park is an optimisation problem today solved manually. As the task is highly complex, a resource to support the architect could improve the result. A human might miss a good solution if it is not intuitive, relating to the persons own preferences. A well-formed computer program designed to evaluate all possible parking spot locations can find solutions a human would miss, since it lays nothing but mathematical valuation in its calculations. Thus, a computer can provide the human architect with new perspectives or even better solutions, potentially in less time than the architect would require. However, this does not mean it is a menial task for a computer. For a person it might come naturally that adjacent parking spots probably should be parallel, while a computer might see each parking spot individually.

In a not too distant future, cars will possibly drive and park themselves. Volvo Car Group envisions a concept called *Valet parking reimagined*. In this concept, a driver could exit his or her car outside a car park and let the car do the full search and park procedure on its own. If this is possible, it would also be conceivable to build car parks dedicated to such cars, in which the spots could be tighter and roads narrower, effectively packing more cars onto a smaller area. In such a scenario, a computer could be of aid for comparing layouts for different dimensions of parking spots.

1.1 Background

One of motorisms greatest challenges is that it takes up a lot of urban space per passenger. A modern, dense and growing city simply does not have the ability to lease the large areas that traditional traffic requires. In many places around the world discussions are ongoing regarding car free city centers and the desire to let cars take less space and be less visible in the cities [3]. In Gothenburg there is a vision that higher shares of parking spots should be relocated from the roadsides to

bigger car parks [4].

Volvo Car Group are working on a project called Drive Me. It is a cooperation between Volvo Car Group, the Swedish Transport Administration (Trafikverket), Swedish Transport Agency (Transportstyrelsen), Lindholmen Science Park, Chalmers, Autoliv, Gothenburg City Parking Company (Göteborgs Stads Parkerings AB) and the city of Gothenburg. The purpose is to study the social benefits of autonomous driving and for Volvo Car Group to become the leading company on sustainable mobility [5]. Within this project is also to investigate new possibilities gained from the abilities of autonomous cars, to better be able to face the future needs and challenges in building effective and sustainable car parks.

To facilitate the analysis of parking differences between different cars, a tool for automated parking spot distribution would be useful. Such a tool would allow for unbiased comparisons, since there is little or no risk that the car park design is influenced by chance. Beyond the use of comparison, such a tool would facilitate overall design of new car parks. Car park design is currently done manually by experienced architects. This is sometimes time consuming work where human thinking is wasted on calculations that could be done faster by a computer.

The programming problem of fitting items within bounds is a classic in combinatorial optimisation, operations research and theoretical computer science. The problem has many names and variations such as the knapsack problem, the cutting stock problem and the bin packing problem. They are all similar in the way that they revolve around fitting objects within given bounds, whether it is volume, weight or area. These are further described in Section 2.4.

1.2 Problem Definition

For the companies designing car parks some data are known, e.g. the layout of the area, specified sizes of the parking spots etc. Desired is a solution, given these parameters, that gives the highest number of spots. The solution must as well meet all requirements, such as reachability for all spots.

The problem faced in this thesis is given as the System part in Figure 1.1. The result of this thesis shall enable a person to give inputs according to the figure and within a reasonable amount of time receive the results to evaluate. The maximum limit of a reasonable time is decided to be 24 hours for an area of $1000\,\mathrm{m}^2$. The input values are subject to variation from the user but the program should always be able to compute an optimal solution, or if that is not possible, a near optimal solution. The result should be presented with coordinates and graphically, as well as with one or several *fitness values*. Fitness values are measures from the solution proposed used in order to compare different solutions, for instance, how many spots that fit in the solution proposed.

1.3 Purpose and Objective

The goal of the thesis is to, with mathematical methods, optimise the distribution of parking spots in a parking area in order to maximise the number of spots. The

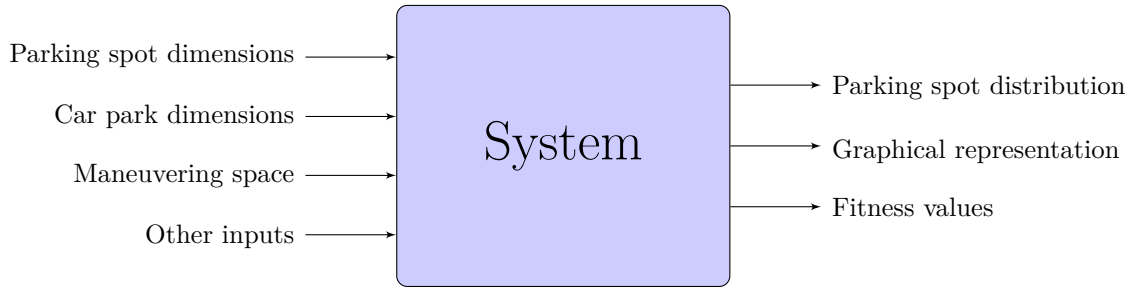


Figure 1.1: Black box illustration of the program.

optimisation methods should be applicable for use with both autonomous and manual vehicles. The purpose is to allow for better area utilisation of car parks, and also to enable analysis of potential benefits of valet parking and assisted parking solutions.

In order to achieve the set goal, a functioning tool is developed in MATLAB. The tool's purpose is to optimise the distribution of parking spots in car parks. The user should be able to specify measurements such as walls and parking spot sizes.

The objectives of this thesis can be stated as:

- Develop a program that can interpret given inputs. This Master's Thesis does not include work on a Graphical User Interface (GUI). It is therefore sufficient that the input is managed in a text environment, such as XML. The end user is assumed to be familiar with such an environment.
- Develop another program that defines the constraints, formulates an optimisation problem and sends it to a solver. It is of importance that the program can handle the inputs given and that it produces output that the solver can handle.
- Choose an appropriate solver for the optimisation problem.
- Develop a program that handles the result given by the solver and presents it in a clear way. It should present the result as coordinates of each parking spot, as well as graphically for easy comparison. Fitness values for comparison of result should be presented as well.

For the distribution of parking spots to make sense, there is a list of criteria each spot has to fulfil. The program must take into account the following constraints:

- Parking spots can only be placed at allowed areas. Due to this, no parking spot can be placed outside of the area, or in a way that it is not compromised by other spots.
- All parking spots should have enough manoeuvring space in order to allow cars to park on them, via manual or autonomous parking.
- Parking spots cannot intersect with each other or manoeuvring areas.
- All parking spots are accessible by roads, i.e. the road should be continuous so that every spot is reachable from everywhere, without leaving the road.

1.4 Delimitations

The problem of packing items is highly complex. This thesis is therefore delimited in a number of ways. The delimitations are divided into general delimitations and program delimitations. The general delimitations concerns which areas work is executed in, for example that no vehicle modeling is included. Programming delimitations concerns what inputs the program developed can handle.

1.4.1 General Delimitations

The thesis and the final program only considers the two-dimensional case of car parks. This means that no multistorey parking houses are considered, nor is the height of cars of any interest.

This Master's Thesis is solely focused on distribution of parking spots and the space they require. The program developed can be used to evaluate the potential benefits of new driving systems, such as autonomous cars. This thesis does not however include any investigation of the functionality of autonomous parking systems, nor of their capabilities. One of the purposes of this thesis is to allow for investigation of potential gain in number of spots, by shrinking parking spot dimensions for autonomously parked cars. With the technique of today, the cars park themselves with marginally better precision than an average driver. However, as the technique develops, systems of significantly higher precision will arise. Instead of adjusting this thesis to what can be achieved today, the user is provided with a tool that gives a result depending on what input values are estimated for the car to handle. The results in Chapter 10 are examples of potential improvements, based on dimensions that are estimated to be manageable in the future.

As mentioned in Section 1.3, the thesis does not include work with a Graphical User Interface, GUI, of the final product. The end user needs to be familiar with the environment used, such as MATLAB. Lastly, the thesis does not include any modelling of vehicle movements, but takes parameters such as turning radius and required manoeuvre space from developers and regulations.

1.4.2 Program Delimitations

For the programming problem to be of adequate complexity, the program is also limited in several ways. The program is limited to only function with the following inputs:

- car park dimensions, excluding obstacles such as columns, where the car park is of rectangular shape
- parking spot size, limited to one size for each run
- manoeuvring requirements, i.e. dimensions of open space around parking spot

The program does not take into account any placement of entrance or exit, this is left to the user to implement manually.

1.5 Solution Overview

This thesis suggests two, unrelated, approaches to parking spots placement. The first, *spot formulation*, places each spot individually, but computes in reasonable time only for a small number of parking spots.

The second, *pattern formulation*, places horizontal rows of parking spots and manoeuvre rows. This approachs works for large car parks. It also includes a function for iterating lengths in a small interval, useful for checking if there is something to gain from increasing or decreasing the car park area.

Both solutions have approximately the same architecture, as illustrated in Figure 1.2. The input is entered through a XML-file which is then read by the program. The XML data is then used to write a .lp file which is sent to an external solver, CPLEX or Gurobi.

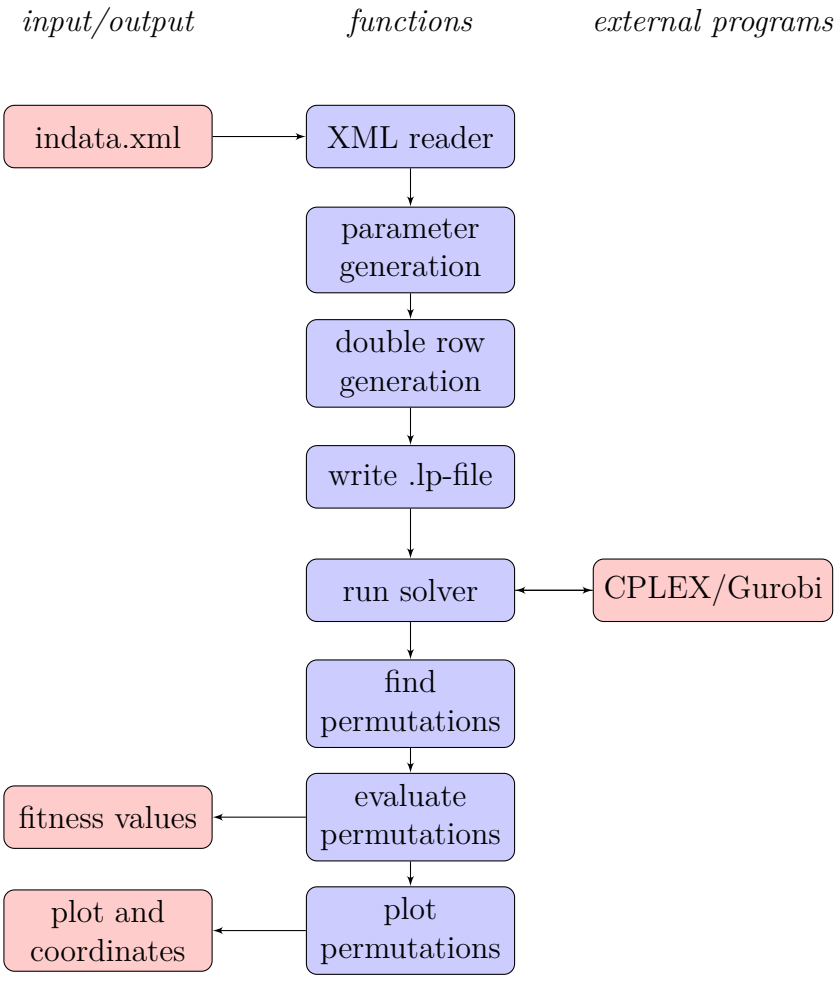


Figure 1.2: System overview of inputs, functions, external tools and outputs.

For the spot formulation, the program ends by plotting the parking spots within the car park. For the pattern formulation, the results from the solver are used to find all possible permutations of the pattern proposed. The patterns are evaluated with respect to access of roads, a process placed outside of the optimisation. Finally

the result is given as a plot of a suggested layout, coordinates of the spots and fitness values to evaluate the solution are provided as well.

The two approaches are unrelated to each other. Only the second, pattern based formulation is analysed and evaluated. It is not required to understand the spot formulation in order to understand the pattern formulation.

1.6 Thesis Organisation

Chapter 2 provides theory related to the subject; complexity of the problem, similar problems and optimisation theory, as well as technical specifications. In Chapter 3, related work is presented. The thesis is thereafter divided into two parts for the different approaches. The parts are independent of each other, and one is not needed to understand the other.

In Part I, a spot placement formulation to the problem is presented. Chapter 4 describes how the formulation is constructed. Further on, its MILP formulation is presented in Chapter 5. In Chapter 6, the computational challenges of this method are explained.

Part II proposes a pattern based formulation of how the problem can be expressed. First, in Chapter 7, a system overview is given, explaining how the formulation is structured. Secondly, Chapter 8 defines the corresponding MILP formulations and lastly, Chapter 9 describes all heuristics that processes the solution proposed by the solver.

In Chapter 10, results from the generated program in Part II are presented and analysed. Finally, conclusions from the results are given in Chapter 11, as well as discussions upon the results, and suggestions of future work for the second approach.

2

Theory

This chapter reviews the theory behind the methods described in this report as well as theory useful to understand why these methods are used. This chapter includes sections on computational complexity, 2.1, optimisation methods, 2.2 and classical problems 2.4. The last section of the chapter, 2.5, explains the technical specifications of the thesis.

2.1 Computational Complexity

Computational complexity is used to describe how much computational resources are required to solve a given task. In other words, to describe the size of the number of computations required. Knowing how much time a computation takes, this can be translated into how much time a given task takes.

One of the most fundamental classes of complexity is called Non-deterministic Polynomial time (NP). What is meant by a problem being of NP complexity is that it is verifiable in polynomial time. This means that given a proposed solution to the problem, the solution can be verified to be true or false in polynomial time.

Within the class NP are also classes P and NP-complete. A problem that is in P can not only be verified in polynomial time, but also be solved in polynomial time. A problem that is in NP-complete however, cannot[1]. The class NP-complete is the intersection between classes NP and NP-hard. The class NP-hard implies that the problem is at least as hard as the hardest problems in NP. To illustrate with an example, assume that the decision problem *"can value X be exceeded without exceeding limit Y?"* is NP-complete. In that case, the optimisation problem *"Maximise value without exceeding limit Y"* is NP-hard. A problem that is NP-complete cannot be solved in polynomial time. Instead, the calculation time for such problems are of sub-exponential, exponential or even greater size. Table 2.1 gives an overview of how calculation times vary depending on complexity function. The table shows how the computation time varies for polynomial and exponential time complexity functions, relating to the number of inputs, n.

To find out if a problem is NP-complete is a complex task. The common approach is to prove that it is equally hard as a similar problem which has been proven to be NP-hard already. The definition of when a problem is complicated enough to be classified as NP-complete originates from the Cook-Levin theorem, developed by Stephen Cook and Leonid Levin [6]. The theory was soon after expanded to a list

[1]This is on the assumption that $P \neq NP$.

of 21 NP-complete problems by Richard Karp in his paper "Reducibility Among Combinatorial Problems" [7].

Table 2.1: Comparison of computation time, depending on time complexity function, for different input sizes.

Time complexity function	Size n					
	10	20	30	40	50	60
n	0.00001 seconds	0.00002 seconds	0.00003 seconds	0.00004 seconds	0.00005 seconds	0.00006 seconds
n^2	0.0001 seconds	0.0004 seconds	0.0009 seconds	0.0016 seconds	0.0025 seconds	0.0036 seconds
n^3	0.001 seconds	0.008 seconds	0.027 seconds	0.064 seconds	0.125 seconds	0.216 seconds
n^5	0.1 seconds	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	0.001 seconds	1.0 seconds	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	0.059 seconds	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

2.2 Optimisation Methods

There are multiple ways to approach optimisation. MILP and Constraint Programming (CP) are two similar methods, both proving an optimum, although through different courses of actions. Stochastic Optimisation (SO) is another approach which includes stochasticity in the search for an optimum.

2.2.1 Mixed Integer Linear Programming

MILP is the combination of Linear Programming (LP) and Integer Programming (IP). LP and IP are methods of reaching the best possible answer in a mathematical model. In LP, problems are expressed as linear equalities or inequalities on continuous variables with a linear objective function to minimise or maximise. In IP, problems are expressed similarly, with the exception that all variables are restricted to be integers. MILP is when a model includes both linear and integer variables. This is useful to represent values that cannot be fractions, in an otherwise linear model. Examples of such variables are; the number of cars built, yes or no questions, or to model logical statements, such as if-statements. MILP, LP and IP are often solved using the simplex algorithm. The simplex algorithm bases its search for

the optimum on the fact that the optimal point must be in a point of intersection between two constraints, or on one constraint.

Large amounts of integers always result in a vast number of combinations and is often problematic. MILP is no exception. To use integers in linear programming, a LP relaxation is performed. In the LP relaxation, all integers are treated as continuous variables. The course of action is as follows:

- If the solution to the LP relaxation results in all original integer variables taking integer values, the optimal solution has been found. Such a solution is said to be *naturally integer*.
- Often however this is not the case, and the original integers in the LP relaxation are solved to be fractions.
- In general, it can be said that to round the solution of an LP relaxation to the nearest integer is almost certainly non-optimal and may be infeasible. The solution process is instead to use a tree search procedure.
- In the tree search procedure, two or more additional problems are generated in each iteration. One where the original integer is constrained to be less than or equal to the rounded down fraction, and one where the integer is constrained to be greater than or equal to the rounded up fraction.
- Both of these problems are then solved, and branched further if the solution is yet to be integer for all original integer variables.

The more variables to perform the relaxation on, the more time consuming the procedure becomes [8].

2.2.2 Constraint Programming

CP is a programming method where all variables are related to each other by constraints. CP differs from MILP by the way they search for a solution. CP searches for a solution by constraint propagation. What this means is that a solution is reached through pruning the domain for each variable, using inference.

Thorsteinsson [9] states that CP programs solve feasibility problems rather than optimisation problems. Typical feasibility problems are the n-queens problem and the map colouring problem. The goal of the n-queens problem is to place N queens on an N by N chess board, such that no queen can capture another. The goal of the map colouring problem is to colour all fields on a map such that adjacent regions have different colours, using as few colours as possible. This is known to always be possible with four colours.

Thorsteinsson also states that there is only a superficial difference between searching for an optimal solution and a feasible solution. By inserting a new bound on the objective function every time a new feasible solution is found, the feasible set gradually becomes smaller, finally only including the optimal feasible solution.

The objective function is better utilised in MILP however. There it is used to find a search direction. This hastens the search, since suboptimal parts of the search tree thereby can be eliminated.

2.2.3 Stochastic Optimisation

SO differs from both MILP and CP. MILP and CP both set out to search for the global optimum in a given set, investigating each node in theory. They are both methodical processes that only gives a result if they conclude it to be the best value achievable of all possible values. As the name implies, SO does not work in this structured way. SO often utilises an objective function much like MILP and CP. The difference is however that SO randomly chooses search direction within the feasible set. Randomly however is a truth with modification. SO often keeps an internal value of fitness for each solution, and tries new solutions as a combination of previous ones. The almost randomness of the procedure comes from the fact that high fitness solutions have a higher probability of being combined into a new solution. For non-convex or otherwise highly complex problems, the fact that all solutions have a chance of combination is the key to finding the optimum. Without this possibility, the method would often find a local optimum and stay there. Otherwise complex problems are for example, problems with large amounts of integer decision variables. This is where SO has an advantage over MILP and CP.

Where SO has a disadvantage is in the purpose of comparing results for different cases. Since stochasticity is involved, there is always a difference in computation time for SO. This is often handled by placing a time limit for the program on how long it is allowed to run. This however implies that there is a possibility that the global optimum is not reached, even for convex problems. When there is no guarantee that the global optimum is reached, the results depend on chance. This in turn, is not good when different cases are to be compared.

2.3 Optimisation Theory

Mathematical optimisation is a large field of science. To facilitate the reading of this report, basic concepts and terminology of mathematical optimisation are presented below.

An optimisation problem always includes certain elements. There is an objective function, parameters, variables and constraints.

- The parameters describe fixed values. Such as: *There are 2 guests; guests* $= 2$.
- The variables describe values to be determined during optimisation. E.g. *number of plates; plates*.
- The objective function described what is aimed to optimise. For example: *maximise the number of plates; Max plates*.
- Lastly, the constraints describe the relationships between all variables and parameters. E.g. *Each guest can have at most two plates;* $plates \leq 2guests$.

The constraints can be equalities or inequalities. Equality constraints are always active, while inequalities can be active or inactive. An active constraint is something that actively limits the solution, depending on where we are looking in the solution space. The example given above can be seen as a two dimensional graph in Figure

2.1. If the point $guests = 2$, $plates = 3$ is considered, Figure 2.1 shows that this point is not hindered by the constraint. The number of plates can be increased by 1 before hitting the constraint. The constraint is therefore said to be inactive in $(3, 2)$. If the number of plates is increased, so that there are 4 plates for the 2 guests, then the constraint hinders the adding of additional plates. The optimal solution is then reached at 4 plates.

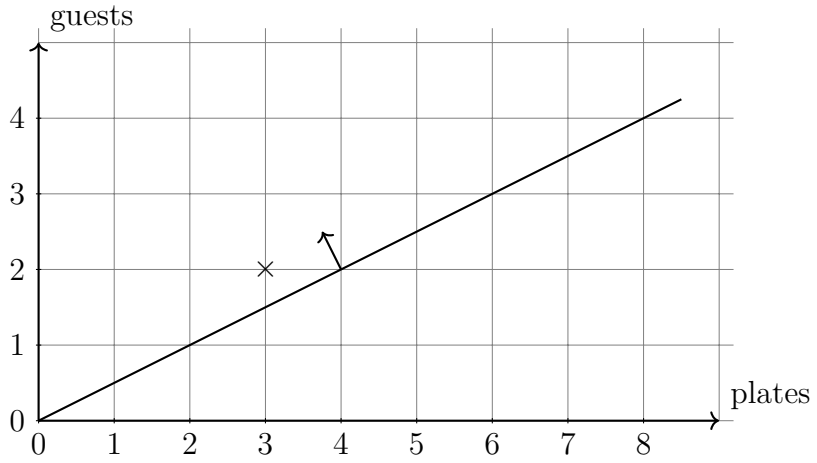


Figure 2.1: Example considering point (3,2), for which the constraint is inactive.

To illustrate the concept of solution space, consider the following extension of the previous example. If the number of guests is changed to being a variable instead of parameter and constraints are added saying that the number of plates must be non-negative and that at most there can be 4 guests, the solution space is the two-dimensional triangle enclosed by the constraints. This example is given in Figure 2.2. If starting in the same point as the previous example, $guests = 2$, and $plates = 3$, Figure 2.2 shows that 1 plate can be added before hitting the limit on number of plates for 2 guests. However the number of guests can still be increased. By increasing the number of guests and plates to both their limits, the optimal solution is reached at 4 guests and 8 plates.

2.4 Classical Problems

The task of maximising the number of parking spots in a car park is comparable to at least three classical problems, briefly mentioned in Chapter 1. The three problems are; the knapsack problem, the cutting stock problem and the bin packing problem. They are quite similar between themselves and are therefore described in further detail below.

The knapsack problem revolves around fitting items within a single container. In this problem, items often have different sizes and values. The goal is to pack items so that the total value inside the container is maximised. To exemplify, consider the following case:

> *Given a set of items, A, each item, $i \in A$, has a value, v_i, and a weight w_i. A binary variable x_i denotes if the item is packed or not. The task is*

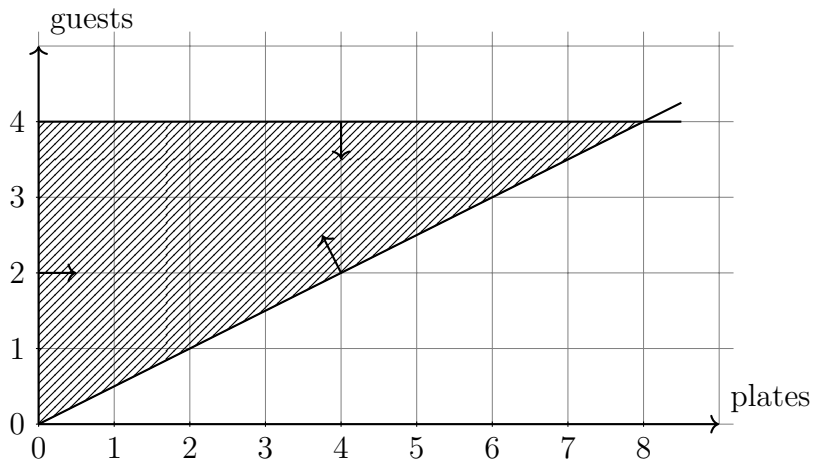


Figure 2.2: A closed solution space: The triangle enclosed by the constraints lines.

> *to maximise the value packed in a knapsack, $\sum v_i x_i$, while not exceeding the knapsacks weight limit; $\sum w_i x_i \leq L$.*

The bin packing problem revolves around fitting items in as few bins as possible. The items given must all be packed. Because of this, item values are of no matter. The number of item sizes are often many. If all item sizes are equal or few, the problem is more related to the cutting stock problem. The following is an example of a traditional bin packing problem:

> *Given a set of items, A, and an infinite set of bins, B. Each item, $i \in A$, has a length, l_i. Each bin, $j \in B$, has length, L, and a binary variable b_j which denotes if the bin is used. The binary variable x_{ij} denotes that item i is packed in bin j. The task is to minimise the number of bins used, $\sum b_j$, while not exceeding the length limit for any bin; $\sum_i l_i x_{ij} \leq L b_j, \forall j \in B$. Each item must be packed in one of the bins used, $\sum_j x_{ij} = 1, \forall i \in A$.*

The cutting stock problem originates from the problem of cutting pieces of stock material into smaller pieces of requested size to fulfill a demand, while minimising the needed number of stock pieces. The item sizes do not vary, and the number of items to fit can exceed the required minimum. The stock pieces can also alter between a number of sizes. To exemplify, consider the following case:

> *Given a set of item types, A, an infinite set of stock pieces, B, and a list of all possible combinations of how stock pieces can be cut into item types (often called "patterns"), P. Each item type, $i \in A$, has a length, l_i, and has a demand, d_i, which denotes how many are required to be made. The stock pieces all have length L. For each pattern, $j \in P$, an integer variable, x_j, denotes how many times pattern j is used. Each pattern also has a parameter, a_{ij}, which denotes how many times item type i is used in pattern j. The task is to minimise the number of stock pieces used, $\sum_j x_j$, while fullfilling the demand, $\sum_j a_{ij} x_j \geq d_i, \forall i \in A$.*

Bin packing and cutting stock are especially similar since they both attempt to minimise the number of bins, or stock pieces used. A way to understand the difference between bin packing and cutting stock is to examine the way their solutions look. The solution to a cutting stock problem will often be a list of packing patterns and how many times they are used. For example, if all item sizes are equal, there will be only one pattern. In the bin packing problem, each bin may have a unique packing pattern.

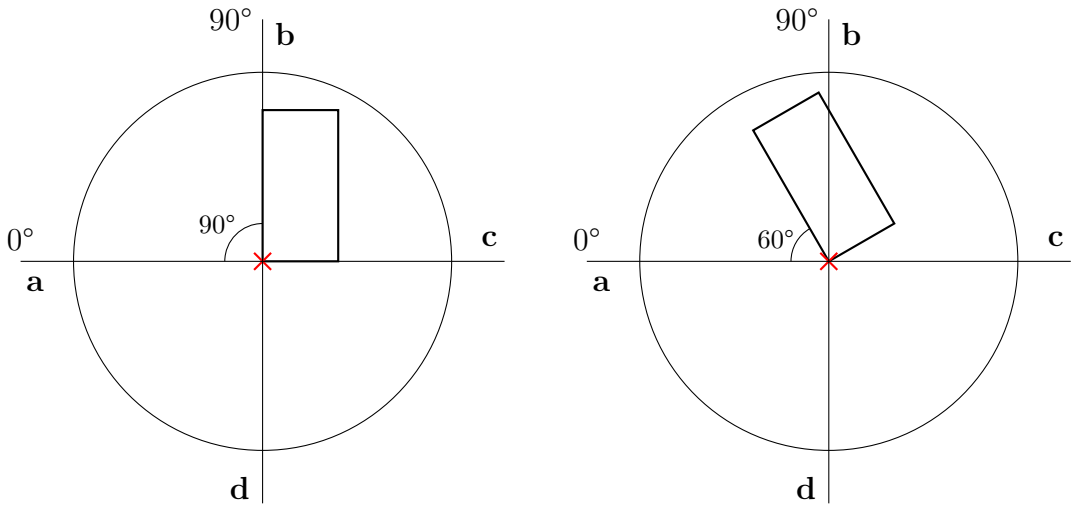
By viewing each parking spot as an individual rectangle and delimiting the problem to handle car parks only of rectangular shapes, it is clear that the problem of parking placement is very similar to these three classical formulations. Since the knapsack problem is also a maximisation problem, this is the closest relative. The individual parking spot optimisation problem can be viewed as a special case of knapsack problem, where all values and sizes are equal, however with added constraints. The added constraints are formulated to take into account the need for manoeuvre space next to each spot, and the fact that every spot must be accessible by a road from a given entry point in the car park. One can speculate whether these added constraints reduce the solution space to such a degree that the problem is no longer NP. The model presented in Chapter 4 does however not show any such tendencies. Since all of the classical optimisation problems have been proven to be NP-hard; it can be assumed that by extension, the parking spot optimisation problem is also NP-hard.

2.5 Technical Specifications

Traditionally, angles are given in the sense of the unit circle, with the polar coordinate system. In parking design however, angles are defined in a mirrored sense. Figure 2.3a shows the case of a parking spot with angle 90°. Note how the angle is measured, clock-wise from the left of the horizontal axis, labelled **a**. Thus, for a parking spot, the angle is measured from the axis **a** to the left long side of the parking spot, as illustrated in the figure. The parking spot is fixed in the origin by its lower left corner, also illustrated in the figure. In Figure 2.3b, the spot now has an angle of 60°, measured in the same way. Parking spots takes values in the range [0°,90°] as described here. Although, spots can be rotated in the other direction, but in that case it is mirrored, and it is the corner on the other side of the short edge that is fixed in the origin.

Values on dimensions for different parking spots are based on standard dimensions, given from the Gothenburg City Parking Company (Göteborgs Stads Parkerings AB). During the result generation these values are used as the basic dimensions, hence all results of improvements when changing any dimensions are correlated to them.

No evaluation is done to establish what dimensions are suitable for autonomous cars of today. This is partly because of the limitations in today's systems and the possibilities of that of tomorrow. In addition, it is regarded as a subject outside of the scope of this thesis, to examine the capabilities of the movement of an autonomously parking car. Instead, all dimensions chosen and their corresponding result are seen as what profit can be achieved, if the system handles the parameters chosen.



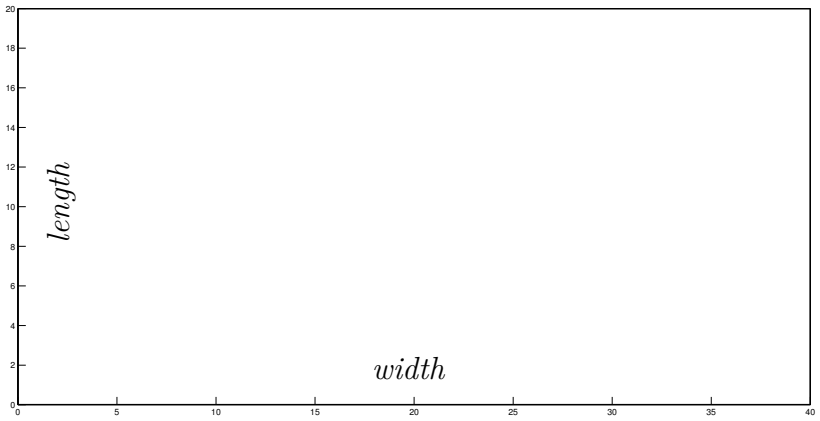
(a) Parking spot with $\varphi = 90°$. **(b)** Parking spot with $\varphi = 60°$.

Figure 2.3: Convention for angle definition in parking design.

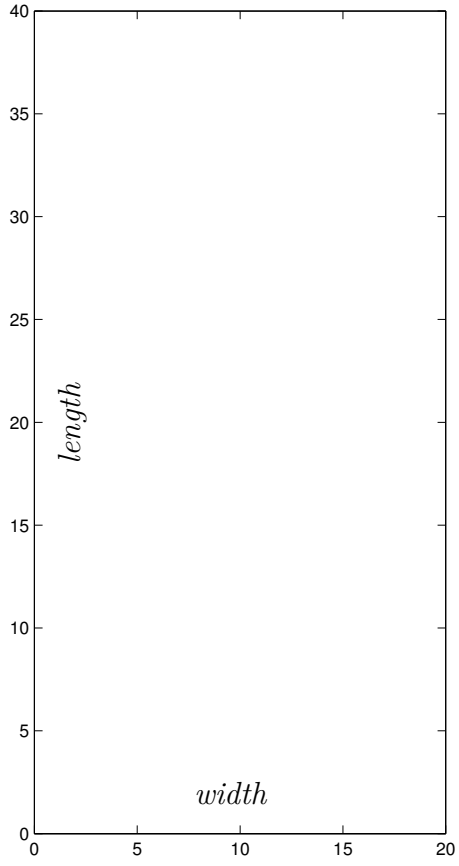
Considering the lines of the parking spots, they are seen as infinitely thin during calculations. This means that the cars are allowed to be positioned on the line, since lines for parking spots in reality of course have a width when painted.

When denoting the car park area, width is denoted correspondingly to the x-axis in a Cartesian coordinate systems. In a similar way, the length is denoted correspondingly to the y-axis in such a system. This is illustrated in Figure 2.4 and, these annotations are regardless of which dimension is larger.

When the *width* is larger than the *length* of the car park, the car park is said to be of horizontal orientation. When the *length* instead is larger than the *width*, the car park is said to be of vertical orientation.



(a) A horizontally oriented car park.



(b) A vertically oriented car park.

Figure 2.4: Illustrations of width and length for different car park orientations.

3
Related Work

In the literature there are many approaches to packing items within bounds. The most interesting related work comes from Porter et al. [10] in the report *Optimisation of car park designs*. In the report, the problem faced is to maximise the number of parking spots within any car park. The car parks are allowed to contain internal obstacles, but the perimeter must be of polygonal shape. Three approaches to the problem are presented; Tile and trim, optimising the road, and optimising over road networks. Theory regarding optimal parking angle for maximum parking spot density is presented in [10] with the result shown to be 90°, which means a perpendicular parking spot. This is however with the assumption that maneuver row length is a function of angle according to

$$L(\alpha) = w + lsin(\alpha), \tag{3.1}$$

where L stands for maneuver length, w denotes spot width and l spot length.

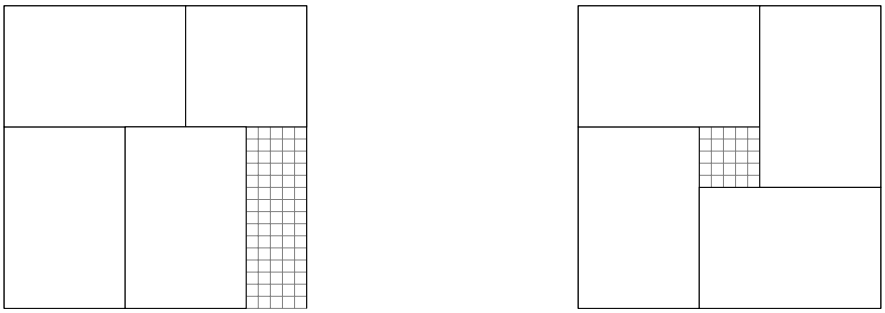
The *Tile and Trim* approach is to use the optimal parking angle and tile this to cover the complete car park area. The tile is then rotated until an optimal placement is found. The spots not completely inside the car park area are then trimmed away. This sometimes leads to unused portions of area large enough for other angles of spots. The authors attend to this by constructing a program in MATLAB which can optimise the number of cars in such a space.

The third approach in [10] describes a way to place nodes in road networks inside the area, based on the assumption that along every road there are parking spots to each side. It is not however described how this optimisation would operate but one could speculate that it would involve stochastic optimisation, rather than linear programming.

For one-dimensional multiple bin packing problems and cutting stock problems, column generation or pattern generation as described by Lundgren et al. [11] is a well known method. The pattern generation method involves creating a set of patterns of items, and then through iteration create and find the most efficient patterns that fulfill the given item requirements.

This method is excellent for one-dimensional problems but not as good for higher dimensions. Carvalho et al. [12] formulates one method for use in two-dimensional problems. This method is interesting, since the problem is similar to the parking spot packing problem faced in this thesis. As many other methods however, it relies on the guillotine cutting constraint which heavily shrinks the feasible set. The guillotine cutting constraint is not applicable to the problem faced in this thesis, since it would over-constrain and remove otherwise potential optimal solutions. The guillotine cutting constraint guarantees that all items are placed such that they can

all be separated using only orthogonal cuts that bisect one dimension of the sheet. Figure 3.1 illustrates the difference between a guillotine cuttable sheet, and a non-guillotine cuttable sheet. In Figure 3.1a, it is always possible to separate items by a single cut through the full area, then repeat the procedure for the resulting parts. In Figure 3.1b however, it is not possible to cut through all the area from edge to edge, without destroying another item.



(a) Guillotine cuttable pattern. **(b)** Non-guillotine cuttable pattern.

Figure 3.1: Comparison of guillotine cuttable pattern, and non-guillotine cuttable pattern.

Lodi et al. [13] surveys methods for two dimensional bin packing, based on strip packing. Such methods base their packing heuristics on the division of total area into strips that are searched for the optimal packing. The use of heuristics is a common approach. According to [14], and universally accepted, linear and dynamic programming approaches are capable of finding the optimum only for small-scale problems, and for larger problems heuristics is a must.

Saadi et al. [15] formulates another approach. Using exact branch and bound and precalculated lower bounds a solution is computed. Each node in the branch and bound corresponds to a feasible internal packing rectangle. The internal rectangles are obtained via vertical or horizontal builds of items. These vertical or horizontal builds however work in such a way that a new item is always placed above or next to the already existing rectangle of items.

Pisinger [16] has a similar approach. By using a heuristic which places items on the contour of predeceasing items, a solution is obtained. The solution is then described as a sequence pair. The sequence pairs describe in what order the items lies, from top left to bottom right, as well as from bottom left to top right. These sequence pairs can thus be permuted and the corresponding placements tested. One way of finding better permutations of the sequence pairs is by a simulated annealing algorithm.

Egeblad and Pisinger in [17] drafts a MILP formulation of the two and three dimensional knapsack problem. The formulation has $6n^2 + n$ binary variables and $3n$ continuous variables. Egeblad and Pisinger claims that even though the size of binary variables are not alarming, the problem is hard to solve. This is said to be mainly because of the use of conditional constraints which, during LP-relaxation, will lose their effect.

Huang and Korf [18] approaches the problem through assigning each item an x-coordinate first and its y-coordinate second. Huang and Korf claim to outperform all

other algorithms in the given benchmarks. The benchmarks however never exceed a limit of 32 items at most.

In conclusion, a lot of work has been done in the field of optimal packing, but not as much on the subject of packing cars, especially not on autonomous parking cars. Since the goal of this thesis is to develop a deterministic approach, useful for comparison of different parking cases, none of the previous approaches described in this chapter are applicable. They are however a source of inspiration.

Part I

Spot Formulation

4

Spot Formulation

In this chapter an exact, but incomplete, formulation of the optimisation problem is described. This formulation was the first attempt to find the optimal distribution of parking spots in a car park. However the search space grows quickly with the number of parking spots, making the computing time grow to unmanageable proportions even for 15 parking spots. This phenomenon is described in general in Section 2.1, and specifically for this case in Chapter 6.

4.1 Overview

The main components of the formulation are the parking spots and the key task is to place them in an optimal way in order to maximise the number of spots. The parking spots are described by their coordinates (x_i, y_i) and rotation φ. The coordinates describe the point located in the spots lower left corner when standing, as shown in Figure 4.1. All spots are the same size and they have length l and width w.

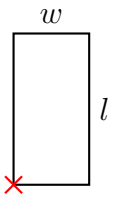
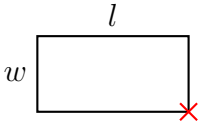
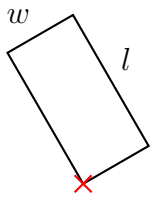


Figure 4.1: Single spot, $\varphi = 90°$.

The spots can also be rotated by an angle φ, with the location of the spot still in the same corner, see Figures 4.2a and 4.2b.



(a) Parking spot, $\varphi = 0°$.



(b) Parking spot, $\varphi = 60°$.

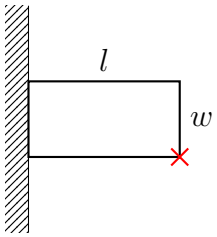
Figure 4.2: Parking spots with different rotations.

The various rotations of a spot changes the limitations on the spot coordinates in order to stay within the given area. The car park is defined by its outer coordinates.

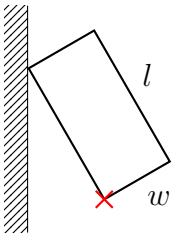
If the left wall of the car park is given as a straight, vertical line, the limitations on x_i will vary with the rotation on the spot according to (4.1a) - (4.1d).

$$x_i \geq l\cos\varphi + x_{wall}, \qquad \forall\varphi \in [0, 90] \tag{4.1a}$$
$$x_i \geq x_{wall}, \qquad \forall\varphi \in (90, 180] \tag{4.1b}$$
$$x_i \geq w(\cos\varphi + 1) + x_{wall}, \qquad \forall\varphi \in (180, 270] \tag{4.1c}$$
$$x_i \geq w\cos\varphi + l\cos\varphi - 270 + x_{wall}, \qquad \forall\varphi \in (270, 360) \tag{4.1d}$$

In the same way, upper limitations on x_i are formed, as well as upper and lower limitations on y_i. Figure 4.3a and 4.3b show parking spots with minimum distance to the left wall, for two different rotations. This means that the constraint (4.1a) is active.



(a) Parking spot, $\varphi = 0°$, minimum distance to wall on the left side.



(b) Parking spot, $\varphi = 60°$, minimum distance to wall on the left side.

Figure 4.3: Parking spots with different rotations and minimum distance to wall on the left side.

The parking spots have not only the outer bounds in form of walls to take into account, no spot must intersect with any other spot. This means that spot n_{i+1} must be either to the *right* of, to the *left* of, *above* or *below* spot n_i. This statement quickly increases the search space as the number of parking spaces increases, since both a solution where spot n_i is to the left of spot n_{i+1} as well as the other way around are true solutions. To decrease the number of possible solutions the spots are decided to be packed in numerical order, where spot n_{i+1} has to be to the *right* of spot n_i. If it is not possible to place spot n_{i+1} to the right of spot n_i, it can be placed *above* instead. Figure 4.4 shows five spots placed as described.

Due to the rotation of the spots the least distance allowed on the x-axis between two spots can vary as $0 \leq x_{dist} \leq l + w$. Figure 4.5a and 4.5b shows two possible cases, where the least distance is w and 0, respectively.

In order to design a functional (and traditional) parking space, the spots cannot be packed too densely - there must be room for a road as well as the manoeuvre space. In Figure 4.4, spots 1 and 2 are not reachable. A first step to this is that at least one side of each spot must have a certain space free, in this case, spot 4 and 5 would have to move.

Even if all spots in the system each have a manoeuvring space big enough for the car to move into the spot, this is not enough if the spot is not accessible from the entry. Thus, all manoeuvring spaces need to be connected, maybe via some free space.

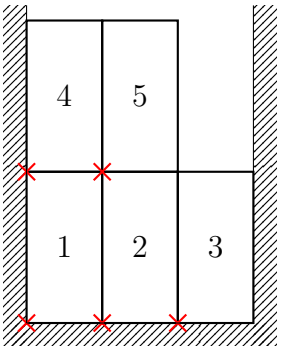
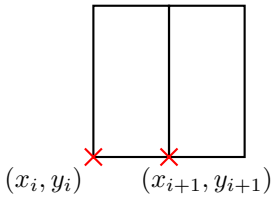
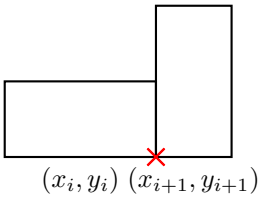


Figure 4.4: Five spots placed in the desired order, spot n_{i+1} is to the right of spot n_i if possible, otherwise above and to the far left.



(a) Two parking spots, $\varphi = 90°$, least distance on $x = w$.



(b) Two parking spots, $= 0°$ and $90°$, least distance on $x = w$.

Figure 4.5: Parking spots n_i and n_{i+1} placed with minimum distance to each other, with respect to the x-axis.

Obstacles can also affect the placement of the spots. Examples of such obstacles are columns, stairs and elevators. Adjustment of spot placement with regard to such obstacles can be made in the same manner as the constraints for no spot intersection.

4.2 Delimitations

In order to be able to solve the layout problem within reasonable time, a number of delimitations are made. These are mainly done to reduce the number of discrete variables.

- Spots can only alternate between two angles, 0° and 90°.
- The car park is defined as a rectangular space.
- All spots have the same dimensions.

The first delimitation is due to several reasons. The chosen optimisation method MILP only handles linear equations. Trigonometric expressions can therefore not be used with continuous variables. An alternative is a binary variable for each angle and spot. However, adding more of these integer or binary variables associated with each spot, would make the combinatorial challenge insurmountable for a lower number of spots. Since the spots are symmetric, angles 0° and 90° are identical to orientations 180° and 270°, which therefore need not be included.

The second delimitation is made to reduce the complexity of the wall constraints, since these are assumed not to be the most limiting constraints.

The last delimitation is made in order to reduce the complexity of the formulation. If multiple sizes of spots was available, more discrete variables would be needed which would make the approach of packing spots in numerical order obsolete. If multiple sizes of spots was available, the program would need to evaluate all permutations of the numerical order.

4.3 Optimisation Method

Three methods to optimise the formulation was investigated in Section 2.2. At the end, MILP was chosen as the method to use. This might seem like a peculiar choice of method, considering the statement in 2.4 that claims the parking optimisation problem to be NP-hard, and brute force solutions to those kinds of problems are time consuming. In these cases, SO can find a solution in less time, but it does not ensure global optimality. One of the main purposes of this thesis is to compare the number of spots fitting in an area for normal parking and for autonomous parking. Due to this, results that varies between runs makes comparison uncertain. Hence SO was out ruled as the optimisation method.

The choice between CP and MILP fell on MILP for practical reasons; it is a known method for the authors of this Master's Thesis and Chalmers University of Technology provides access to expertise within the field.

5

MILP Formulation

The constraints described in text in Section 4.1 are here translated into expressions formulated for use by a MILP solver. All constraints are applied to all spots $i \in I$ unless otherwise stated, where $I = \{1, 2, .., n_{spots}\}$. There are n_{spots} available, where n_{spots} is a number slightly greater than the number of spots that can possibly fit into the area. The value of n_{spots} is calculated from the total area of the car park and the area of a parking spot.

5.1 Objective Function

The objective function that describes the objective for the whole optimisation is given in (5.1). There, n_i is a binary decision variable that takes value 1 if the spot fits in the solution and 0 otherwise.

$$maximise \sum_{i=1}^{n_{spots}} n_i \tag{5.1}$$

5.2 Car Park Bounds

The constraints (5.2a)-(5.2c) make sure that each spot is to the right of the left wall, above the lower wall and below the top wall. The parameters x_{wl}, y_{wt} and y_{wb} are coordinates stating the x and y-bounds of the car park. The subscript w stands for wall, l for left, r for right, b for bottom and t for top. The decision variable h_i is active if the spot has a rotation of 0°, i.e. if it is horizontal.

The constraint (5.2d) says that a spot can be to the right of the right wall, but in that case $n_i = 0$, and n_i does not contribute to the objective function. Such constraints, of the type *if... else*, can in MILP be modelled using the *big M method* [19]. In these formulations, M is assigned some value big enough to activate or deactivate a constraint. If it activates it or not depends on the value of the decision variable it is combined with. In this case, M deactivates the constraint in (5.2d) if n_i take the value 0, thus the spot i can be outside of the car park, on its right side.

$$x_i - l \times h_i \geq x_{wl} \tag{5.2a}$$

$$y_i \geq y_{wb} \tag{5.2b}$$

$$y_i + w \times h_i + l(1 - h_i) \leq y_{wt} \tag{5.2c}$$

$$x_i + w(1 - h_i) \leq x_{wr} + M(1 - n_i) \tag{5.2d}$$

5.3 Shelfing

To avoid unnecessary permutations as described in Section 4.1, *shelfing* is implemented. The term shelfing refers to the fact that the parking spots are packed as if they were books in a shelf; by filling one shelf at a time, from left to right, as well as from bottom to top. A constraint is formulated to place the spots in this manner, illustrated in Figure 4.4. The constraint works as following. Each spot, i, creates a diagonal line in the upward and left direction. The subsequent spot, $i+1$, must lie above this line. In order for the constraint not to be overly limiting, the constraint is formulated such that the constraint is active when:

- Spots i and $i+1$ lie in the same point. Feasible only when spot i is horizontal and spot $i+1$ is vertical.
- Spot i is horizontal and in the bottom right corner, and spot $i+1$ is vertical and to the far left as illustrated in Figure 5.1.

In order to achieve this, the straight line equation of the diagonal is formulated in a particular way. The gradient, denoted LW (Least Width), is set to the width of a spot divided by the width of the car park, see (5.3a). The Y-intercept is set equal to the spot width. The straight line constraint describing spot i for the case when i is in the bottom right corner is shown in (5.3b), while (5.3c) is the constraint imposed on spot $i+1$ for the same case.

$$LW = \frac{w}{(x_{wr} - x_{wl})} \tag{5.3a}$$

$$y_i = -LW \times x_i + w \tag{5.3b}$$

$$y_{i+1} \geq -LW \times x_{i+1} + w \tag{5.3c}$$

Combining (5.3b) and (5.3c) thus gives the following relationship between the coordinates for any pair of spots i and $i+1$.

$$LW \times x_{i+1} + y_{i+1} \geq LW \times x_i + y_i, \quad \forall i = 1, 2, \ldots, n_{spots} - 1 \tag{5.4}$$

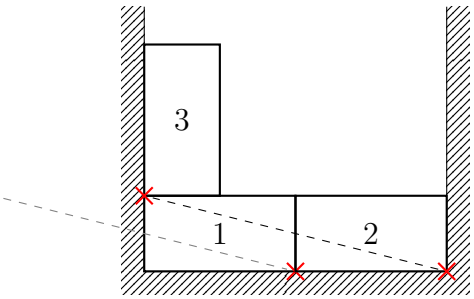


Figure 5.1: Shelfing. Three spots are placed, where spot 2 is above the dashed, grey line created from spot 1, and spot 3 is above the dashed line created from spot 2.

5.4 Lowest Index Placement

Another way to avoid unnecessary permutations of the solution is to make sure that the number of spots that fit in the area have the lowest index i possible. To enforce this, constraint (5.5) is formulated.

$$n_{i+1} \leq n_i, \quad \forall i = 1, 2, \ldots, n_{spots} - 1 \tag{5.5}$$

This make sure that if $n_i = 0$, then n_{i+1} must also be zero.

5.5 Overlap Avoidance

To make sure that the spots do not overlap each other, the constraints (5.6a) - (5.6d) are formulated. The constraint (5.6a) activates the binary decision variable $xLess_{ij}$ if the leftmost side of spot j is to the left of the rightmost side of spot i. In the same way the constraint (5.6b) activates the binary decision variable $xGreat_{ij}$ if the rightmost side of spot j is to the right of the spot i. If both $xLess_{ij}$ and $xGreat_{ij}$ are active, the spots i and j are overlapping horizontally. Therefore $yGreat_{ij}$ is also set to active by constraint (5.6c), effectively forcing spot j to be placed above spot i by constraint (5.6d). These three variables and the three relationships that two spots can have are illustrated in Figure 5.2.

With $yGreat_{ij}$ active, the constraint (5.6d) is also activated and creates a relation between y_i and y_j, making sure y_j is above y_i.

$$x_j - l \times h_j + M \times xLess_{ij} > x_i + w(1 - h_i) \tag{5.6a}$$

$$x_j + w(1 - h_j) < x_i - l \times h_i + M \times xGreat_{ij} \tag{5.6b}$$

$$xLess_{ij} + xGreat_{ij} - yGreat_{ij} \leq 1 \tag{5.6c}$$

$$y_i + l(1 - h_i) + w \times h_i \leq y_j + M(1 - yGreat_{ij}) \tag{5.6d}$$

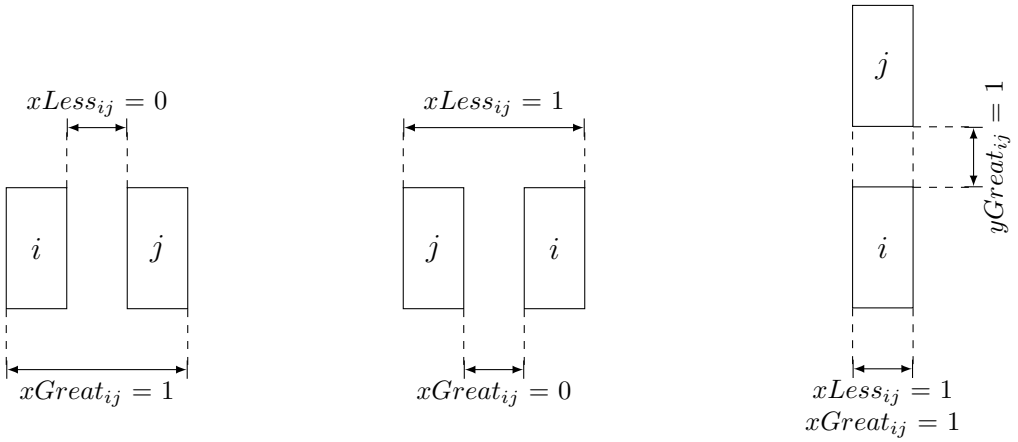
$$\forall i = 1, 2, ..., n_{spots} - 1, \quad j = i + 1, ..., n_{spots}$$

5.6 Manoeuvre Space

In a functional car park, all spots must be possible to move into and out of, without crossing any other parking spot. In other words, every parking spot needs a manoeuvre space adjacent to it. The constraints attempting to model this requirement are described in this section.

To realise the demand of free space next to each spot, new decision variables are introduced, various combinations of *acr* and *vert*; *across* and *vertical*. These are to decide upon weather there is free space next to the spot, horizontally and vertically. Each spot can have up to three sides that are next to a wall or another spot.

The constraints (5.7a) - (5.7d) sets value of $acrL_i$, $acrR_i$, $vertB_i$ and $vertT_i$ to 1 if spot i is next to the left, right, bottom or top wall. The variable man is a constant that describes the least value required between a spot and another spot or a wall.



(a) i is to the left of j. **(b)** i is to the right of j. **(c)** i and j overlap.

Figure 5.2: Variables $xLess_{ij}$, $xGreat_{ij}$ and $yGreat_{ij}$ depend on the horizontal placement of spots i and j.

$$x_i > x_{wl} + man + l \times h_i - M \times acrL_i \tag{5.7a}$$
$$x_i < x_{wr} - man - w(1 - h_i) + M \times acrR_i \tag{5.7b}$$
$$y_i > y_{wb} - man + M \times vertB_i \tag{5.7c}$$
$$y_i < y_{wt} - man + M \times vertT_i - w \times h_i + l(h_i - 1) \tag{5.7d}$$

To find out if a spot j is placed within distance *man* to the *right* side of another spot i, i.e. a neighbour on its *right* side, constraints (5.8a) - (5.8d) are formed. In these constraints, the four binary decision variables δ are used to turn constraints on or off. If all four δ's are set to value 1, spot j is indeed a neighbour of spot i, on its right side. The decision variable acr_{ij} is then set to 1. This is regulated in constraint (5.8e).

$$x_j \leq x_i + M \times \delta_{1ij} - 1 \tag{5.8a}$$
$$x_j - l \times h_j > x_i - M \times \delta_{2ij} + w(1 - h_i) + man \tag{5.8b}$$
$$y_j + l(1 - h_j) + wh_j \leq y_i + M \times \delta_{3ij} \tag{5.8c}$$
$$y_j \geq y_i - M \times \delta_{4ij} + l(1 - h_i) + w \times h_i \tag{5.8d}$$
$$\delta_{1ij} + \delta_{2ij} + \delta_{3ij} + \delta_{4ij} - acr_{ij} \leq 3 \tag{5.8e}$$

In the same way as in constraints (5.8), a spot j can be the *upper* neighbour of a spot i. The constraints (5.9a) - (5.9d) sets the value of the binary decision variables $\delta_5 ij$ - δ_{8ij}.

$$x_j + w(1 - h_j) < x_i + M \times \delta_{5ij} - l \times h_i \tag{5.9a}$$
$$x_j - l \times h_j \geq x_i - M \times \delta_{6ij} + w(1 - h_i) \tag{5.9b}$$
$$y_j < y_i + M \times \delta_{7ij} \tag{5.9c}$$
$$y_j > y_i - M \times \delta_{8ij} + l(1 - h_i) + w \times h_i + man \tag{5.9d}$$

If $\delta_5 ij$ - δ_{8ij} are set to 1, spot j is an upper neighbour of spot i and the variable $vert_{ij}$ is set to 1.

$$\delta_{5ij} + \delta_{6ij} + \delta_{7ij} + \delta_{8ij} - vert_{ij} \leq 3 \tag{5.10}$$

Instead of rewriting similar constraints for looking at left and lower neighbours, acr_{ji} are set to the same value as acr_{ij}, using symmetry. This also decreases the use of binary decision variables, δ. See constraints (5.11a) and (5.11b).

$$acr_{ij} = acr_{ji} \tag{5.11a}$$
$$vert_{ij} = vert_{ji} \tag{5.11b}$$

Finally, for each spot i, constraint (5.12a) is formulated to make sure at least one of the sides is without neighbour.

$$\sum_{j=1}^{n_{spots}} acr_{ij} + 2acrL_i + 2acrR_i + 2vertB_i + 2vertT_i \leq 6 \tag{5.12a}$$
$$\forall i = 1, 2, ..., n_{spots},\ j \neq i$$

These constraints, formulated to ensure free space next to every parking spot are however not functional. Figure 5.3 illustrates how the solver does what it is told, but it is an erroneous way of mathematically describing what is desired. In the figure, parking spot 5 is blocked by parking spot 6 but according to the constraints has its left side free. In order to assure a functional manoeuvre space constraint, these constraints would need to be adjusted. Because of the computational complexity problems discussed in Chapter 6, this way of modelling the problem is deemed unfit, since it cannot work for large sizes of car parks. The constraints are therefore not modified further.

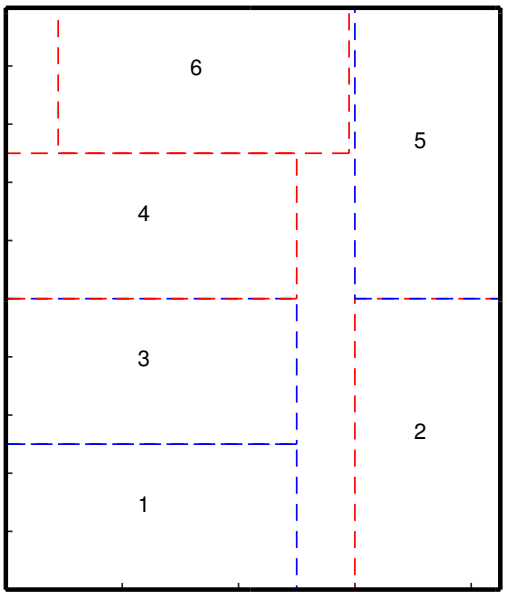


Figure 5.3: Each spot is assured manoeuvre space on at least one side. This fails for parking spot 5 which is blocked by parking spot 6.

6
Computational Complications

This formulation is not complete to solve the problem: it does not consider obstacles such as columns, does not describe how roads are to be placed and does not have a complete solution to ensure that each spot has a manoeuvre space big enough to allow for a car to park in the space. Even so, the formulated constraints result in an explosion of integer variables already, with only a few input variables. Specifically the number of binary variables increases drastically, which affects the computational time. The number of binary decision variables that are generated without including the constraints in Section 5.6, that make sure that some manoeuvre space is included for each spot in the solution, are described by (6.1a). Further, (6.1b) expresses how many binary decision variables that are generated with these decision variables for manoeuvre space included.

$$2n_{spots} + 3 \sum_{k=1}^{n_{spots}-1} k \tag{6.1a}$$

$$12n_{spots} + 15 \sum_{k=1}^{n_{spots}-1} k \tag{6.1b}$$

To get a perception of how many variables this is, Table 6.1 gives some examples. The left column, number of spots, shows the value of n_{spots}.

Table 6.1: Binary decision variables generated for different values of n_{spots}.

No of spots	Binary variables	
	without space	with space
5	40	210
10	155	795
50	3775	18 975
100	15 050	75 450
500	375 250	1 877 250

By adding further constraints and working with the existing ones a solution can be reached, however the computational time will be so high that it is not possible to solve in reasonable time for useful sizes of car parks. No solution is found for car parks including more than 20 parking spots.

Part II

Pattern Generation

7
Pattern Formulation

This chapter describes a formulation that, given a set of input parameters of the system, finds a solution with optimal placement of parking rows. This placement is modified with heuristics in order to make the car park functional with regards to accessibility by road. A final solution is proposed, with coordinates for each spot and fitness values to compare its efficiency, these are further explained in Section 9.4. The pattern formulation is separated from the spot formulation in Part I, thus no information given on the spot formulation is inherited, if not otherwise stated.

7.1 Overview

Instead of placing each spot individually as described in Part I, the pattern formulation treats entire rows of parking spots as items, as well as rows of manoeuvre spaces. An example of a layout is presented in Figure 7.1. In the figure, the horizontal free space is manoeuvre rows, while the corresponding vertical space is *vertical roads*, further explained below.

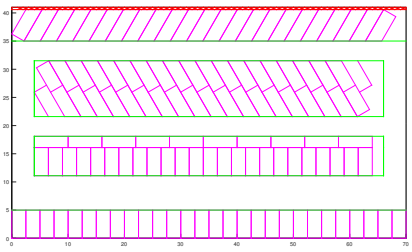


Figure 7.1: Example of a resulting parking layout, using the pattern formulation.

The program is given a list of parking angles to choose from, complete with dimensions and requirements on manoeuvre space. These different parking angles are commonly in this part referred to as *types*. The program calculates for each type how many spots can fit within the car park horizontally when placed in a row next to each other. Thus, a value is produced for each type. The rows are always placed horizontally, but the car park can be defined vertically or horizontally by the user. As in the spot formulation, the shape of the car park is limited to be rectangular, as stated in the general delimitations in Section 1.4.2. The optimisation problem is to decide how many rows of each type to include, given the length of the car park. The optimisation problem does not include placement of accessibility roads to travel between manoeuvre rows, this is adjusted for after the optimisation is done.

The solver outputs a solution with maximum number of spots for the given input parameters. The solution is a list of how many of each parking row type and each associated manoeuvre row to include. Furthermore, the solution contains the total number of parking spots and how much vertical space is unused. The output does not include in which order the rows should be placed.

After the optimisation is done, heuristics are executed to find the optimal ordering of the given rows, with respect to placement of vertical roads for connecting the manoeuvre rows. This is described in further detail in Section 7.1.3. Vertical roads are the roads added in the heuristics, connecting the horizontal manoeuvre rows in order to allow for circulation in the car park, further explained in Section 9.3. Heuristics is a set of commonsense rules, used to solve a problem in a good way, but not necessarily the best way. Heuristics are applied when placing vertical roads and choosing parking row ordering, since adding it to the optimisation model would increase the complexity of the optimisation model greatly.

Figure 7.2 illustrates a parking row with three parking spots. The dimensions are named accordingly to the figure, and these expressions are henceforth used in the thesis. It is assumed that the manoeuvre rows are dimensioned for one way drive. For cars to able to drive both ways, the manoeuvre rows might need to be longer. The length of manoeuvre rows is also a parameter set by the user.

7.1.1 Manoeuvre Rows

Each parking spot requires a manoeuvre space adjacent to it as described in Section 5.6. The same applies to parking rows, and in the same manner as parking spots combined to a row, manoeuvre spaces also combine to form manoeuvre rows. The width of these manoeuvre rows are the same as the width of the car park, see Figure 7.1. The length however, depends on the orientation, i.e. the angle of the associated parking spots, see Figure 7.2. Typically, vertical parking spots (90°) require the longest manoeuvre rows, while parking spots with low angles, such as 30° or parallel parking (0°) has the shortest requirements.

A single manoeuvre row can be used by two parking rows, since there can be one parking row situated on each side of the manoeuvre row. The parking rows need not be of the same type, however in such a case it is important that the manoeuvre row fulfils the highest requirements on length. For example, if a parking row of 90° shares manoeuvre row with a parking row of 0°, the manoeuvre row must be of length as required by the 90° parking row.

7.1.2 Double Rows

For parking spot rows with angles that are not 0° or 90°, the spot can be seen as a parallelogram. Still, the car using the spot is of rectangular shape and can not use all the area. Because of this, there will be some waste in the top and bottom of the spot. The wasted space is marked grey and named *tiltedWaste* in Figure 7.2. There is also an equally large triangle at the opposite side of each spot. This area is however not seen as waste, since it is passed during parking.

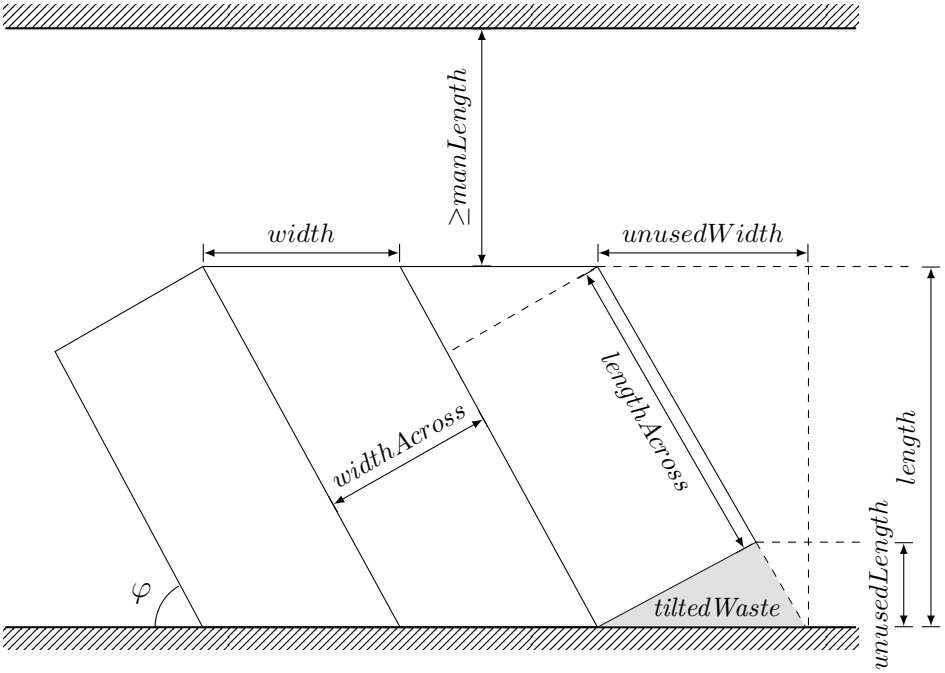


Figure 7.2: Parameter used to denote the dimensions of a tilted parking spot.

However, by placing two parking rows of the same type next to each other (vertically) this space can be utilised, as shown in Figure 7.3. This results in a reduction of row length, since it can be viewed as overlapping between the two rows. The reduction in length for the two rows combined is calculated according to (7.1). Two rows of the same type, combined to utilize the wasted space between them, are referred to as one double row.

$$unusedLength = widthAcross \times \cos(\varphi) \tag{7.1}$$

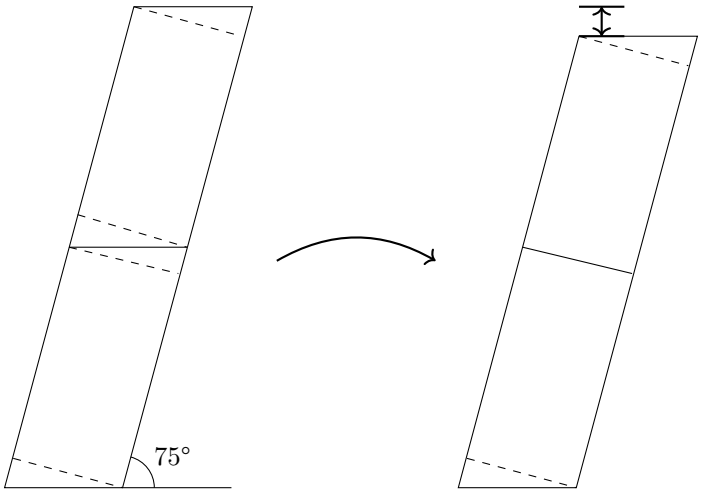


Figure 7.3: Vertical space saved by placing two spots of the same type adjacent to each other.

7.1.3 Result Generation

A MILP problem is formulated from the description in this chapter, in order to give a solver the information required to find the optimal distribution of parking rows. The MILP problem is described in Chapter 8.

After the optimisation is done, heuristics are used to evaluate what order the rows should be placed in, with respect to vertical roads. Then, the result is presented graphically, along with fitness values and coordinates. This process is described in Chapter 9.

7.2 Optimisation Method

The reasoning regarding optimisation method for the pattern formulation is the same as for the spot formulation, described in section 4.3. MILP is chosen as the optimisation method, mainly due to familiarity and access to expertise.

7.3 Indata

The indata that is entered by the user consists of four parts; main settings, car park, parking spot types, and iteration settings. The main settings to specify are:

- Which solver to use
- If to include double rows
- If to include vertical roads
- If to automatically flip the car park to a horizontal layout
- If to allow the program to calculate parking spot dimensions, or use the values given by the user

It is possible to calculate all parking spot dimensions for all angles, given only the length of 90° spots, the width across and parking spot angles. However, this might not always be desired. Hence the last setting. The user can specify as many types of spots as desired in the same optimisation, given that the following parameters are specified for each type:

- φ
- $widthAcross$
- $length$
- $manLength$

The indata to specify for the car park are its dimensions; width, *carParkWidth*, length, *carParkLength*, and the width of vertical roads, *roadWidth*. It is also possible for the user to iterate over the dimensions of the car park, and then perform another optimisation for that area. This is further explained in Section 9.5. For the iteration

settings the user must specify if to iterate over width or length, step size, and number of steps.

Given the indata specified by the user, the program calculates the row value for each type, with and without vertical roads. The value is the number of spots that fit on a row of each type, given the width of the car park. The program also calculates the area occupied for each spot type and the dimensions for double rows, if these are included. The row values with and without vertical roads are calculated for all non-double rows according to (7.2) and (7.3). In (7.3), $pValueRoad_i$ is the value when vertical roads are included.

$$pValue_i = \frac{carParkWidth - unusedWidth_i}{width_i}, \tag{7.2}$$

$$pValueRoad_i = \frac{carParkWidth - 2roadWidth - unusedWidth_i}{width_i}, \tag{7.3}$$

Double rows spots are created by combining two parking spots as described in Section 7.1.2. Each double spot in a double row counts as two parking spots. The value for double rows are calculated similarly to regular parking rows. The addition for double rows is that at both ends of the parking row, there is a possibility that a single spot can fit where a double spot does not. If these spots are present, they are included in the value for the double row. This is illustrated in Figure 7.4 where single spots at the ends of a double row are displayed.

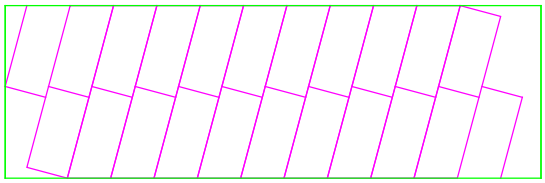


Figure 7.4: A double row, with single spots at both ends.

For the result generation, the area occupied by all spots is evaluated. This area is calculated for each rectangle or parallelogram shaped spot by multiplying spot width and spot length.

7.4 XML Reading

The indata is read by MATLAB through a program *xml_read.m*. The program was created by [20]. The reason for the XML format is that it is well known and that there are libraries for handling XML-files. Such libraries could be of use if the program developed is to undergo further development in the future, for example incorporating a GUI to facilitate input to the program.

7.5 Solvers

The solver is activated through writing a .lp-file in MATLAB and then calling a solver from MATLAB. The solvers used are CPLEX and Gurobi, however any solver that can use standardised .lp-files can work.

8

MILP Formulation

In Chapter 7, a formulation based on rows of parking spots is described. The rows are to be combined in such a way that a maximum number of spots fit in the given area. From here, a MILP problem is to be formulated, in order to let a solver find the optimal solution.

8.1 Sets

There are certain variables and parameters associated with each type of spot, these are listed in Table 8.1. All variables or parameters in the table indexed i are applied for all $i \in I$, if not stated otherwise. The vector *type* describes the types of spots and can take values between 0 and 90. Typical values are, $type = [90, 75, 60, 45, 30, 00]$.

Unless otherwise stated, all constraints are applied for all $i \in I$, where $I = \{1, 2, ..., n_{types}\}$ and n_{types} is the number of types. In the example with *type* above, $n_{types} = 6$. Important is that all vectors indexed over i are sorted so that *type* is always decreasing. It is assumed that types of angles 90° and 0° are always available. Regarding other angles, it is up to the user to define them and include them as types. Furthermore, all variables are constrained to be non-negative. All variables in this chapter are listed in Table 8.1, found at the end of the chapter.

8.2 Objective Function

The objective function to maximise is the total number of spots and is formulated as (8.1). The variable *totalValue* represents the number of spots in the car park in the final solution. The vector $pValue$ indicates how many spots that fit on one row of type i while pNr_i declares how many rows of type i is in the final solution.

$$\begin{gathered} maximise\ totalValue, \\ totalValue = \sum_{i=1}^{n_{types}} pValue_i \times pNr_i \end{gathered} \tag{8.1}$$

8.3 Total Length

The sum of all lengths of parking rows and manoeuvre rows must not exceed the length of the car park area. To enforce this, constraint (8.2) is formulated. This is

also where the unused vertical space, *wastedSpace*, is calculated, as this is the slack variable between summed row length and total car park length.

$$totalLength = wastedSpace + totalManLength + \sum_{i=1}^{n_{types}} pLength_i \times pNr_i \tag{8.2}$$

The variable *totalManLength* is defined by constraint (8.3), that sums the length of all present manoeuvre rows.

$$totalManLength = \sum_{i=1}^{n_{types}} mLength_i \times mNr_i \tag{8.3}$$

8.4 Manoeuvre Rows

To fulfil the demand that each parking row has access to a manoeuvre row of at least its required manoeuvre length, constraints (8.4a) - (8.4b) are formulated. As described in Section 7.1.1, manoeuvre rows of a type with higher angle are also accepted since they are assumed to be of equal or greater length.

Since a manoeuvre row can be shared by two parking rows of different type, an integer variable $mOpen_i$ is included. If $mOpen_i$ is nonzero, this denotes that there exists a manoeuvre row, of type i, that is only used on one side. It is thus open to use for any lower order parking type. The reason for it being constrained according to (8.4b) is that for any type, it is not allowed to include a manoeuvre row that is completely unused by its corresponding parking row type.

$$mOpen_i = 2mNr_i - pNr_i + mOpen_{i-1} \tag{8.4a}$$
$$mOpen_i \leq 1 \tag{8.4b}$$
$$mOpen_0 = 0$$

8.5 Double Rows

When double rows as described in Section 7.1.2 are allowed, some variables change. A double row is seen as one single item. This means that for each type, other than 0° and 90°, a sibling is created. The vector *type* thus grows according to (8.5).

$$type = [90, 75D, 75, 60D, 60, 45D, 45, 30D, 30, 00] \tag{8.5}$$

Thus, also I grows to $I' = \{1, 2, ..., n'_{types}\}$. Correspondingly, all other vectors spanning over I also grow. Introduced is also the set J, a subset of I, which denotes the indices of types that are double rows. As an example, from the current vector *type*, J would be

$$J = \{2, 4, 6, 8\}. \tag{8.6}$$

Since one double parking row requires two manoeuvre rows, additional constraints are formulated. For regular parking rows, constraint (8.4) applies as usual. For

double rows however, when $i \in J$, constraint (8.4a) is replaced by (8.7a) and (8.4b) is replaced by (8.7b)

$$mOpen_i = 2mNr_i - 2pNr_i + mOpen_{i-1} \tag{8.7a}$$
$$mOpen_i \leq 2 \tag{8.7b}$$

When double rows are present, the formulation for sharing manoeuvre rows between different parking types becomes significantly more complicated. This is due to the fact that double rows require a manoeuvre row on each side. Consider the example illustrated in Figure 8.1. Two double parking rows of different type each require two manoeuvre rows. For each parking row, the manoeuvre rows are only used from one side. According to how the ordinary parking rows are constrained, this would mean that they could share two manoeuvre rows. However, since the manoeuvre rows orientation around the double parking rows are fixed, only one manoeuvre row can be shared in practice. In the figure, manoeuvre rows are marked with m and are coloured blue while parking rows are marked with p. The dashed line of the manoeuvre row indicates that it is open for use to other parking rows. The filled line adjacent to the parking row indicate that the manoeuvre row is used by the parking row in question. The figure shows how two double rows with four manoeuvre rows that all have one side open for use can be combined such that they all together use three manoeuvre rows, leaving two manoeuvre rows open for use.

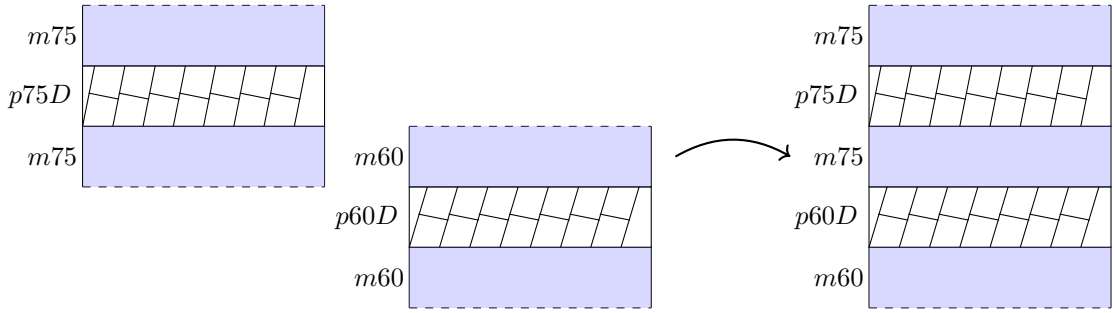


Figure 8.1: Two double rows, both requiring two halves of manoeuvre rows, yet can only share one manoeuvre row.

To model this situation, additional constraints are added, (8.8a) - (8.8c). Constraint (8.8a) is added to all types, while (8.8b) and (8.8c) are only added for types where $i \in J$. These constraints include variables $mRowOpen_i$ and $pExist_i$. Variable $pExist_i$ is used to balance constraint (8.8a) and as a slack for transferring open manoeuvre rows, in the case when the current type i is non-existent. Constraint (8.8a) is the key expression where variable $mRowOpen_i$ is used to constrain the double rows to behave in the way described above.

$$mRowOpen_i = 0.5mOpen_i + 0.5mOpenSlack_i \tag{8.8a}$$
$$pExist_i + pNr_i = mNr_i + mRowOpen_{i-1}, \forall i \in J \tag{8.8b}$$
$$pNr_i \leq M \times pExist_i, \forall i \in J \tag{8.8c}$$

Variable $mRowOpen_0$ does not need to be defined. Constraint (8.8b) is only used for double rows, i.e. when $i \in J$. Since it is assumed that 90° rows are always included, $i \in J$ will never occur for $i = 1$.

Table 8.1: Variables and parameters used in the pattern formulation.

Variable	Description
mNr_i	Number of manoeuvre rows of type i in the solution
$mOpen_i$	Number of manoeuvre rows of type i that are open for use
$mRowOpen_i$	Variable used to ensure manoeuvre rows of type i are shared correctly between rows, when double rows are activated
$pExist_i$	Binary variable saying if type i exist, only applied for all $i \in J$. Used to ensure manoeuvre rows of type i are shared correctly between rows, when double rows are activated
pNr_i	Number of parking rows of type i in the solution
$totalManLength$	The combined length of all manoeuvre rows in the solution
$totalValue$	Total number of spots in the solution
$wastedSpace$	Length of the car park that is not used
Parameter	**Description**
$mLength_i$	Length of the manoeuvre row of type i
n_{types}	Number of types, defined by the user (also including double rows, if they are included)
$pLength_i$	Length of the parking row of type i
$pValue_i$	Number of spots of type i that fits on one row, given the width of the car park
$totalLength$	Length of the car park
$type_i$	Angle of type i, also indicating if the type is a double row or not

9
Post Processing

The optimal solution for the distribution of horizontal rows is obtained, without taking into account placement of vertical roads to connect the manoeuvre rows. This is adjusted for by heuristics.

The solver proposes a solution with a maximum number of spots, *totalValue*, for the given input parameters. The solution is a list of quantities of each parking row and manoeuvre row, along with the number of spots they generate and how much vertical space is unused, the value of *wastedSpace*.

9.1 Permutations

The output from the solver says nothing about how the rows are placed; not on which coordinates they are placed or even in which order. The only thing the output declares is the number of rows, of each type. The first thing to investigate with the result is how the rows can be placed, and in how many ways. To do so, *combinations* are extracted. A set of parking rows and manoeuvre rows in a certain order is denoted a combination. The vector $combination_1$, as shown in (9.1) is one combination of the existing rows. In the combination vector, $p90$ indicates a parking row of 90° and $m90$ indicates a manoeuvre row of corresponding angle.

$$combination_1 = [p90, m90, p60, p60, m60, p00] \tag{9.1}$$

A combination vector is read in such a way that the first index indicates the row type placed at the bottom of the car park. The subsequent index is placed above it, and so forth. Another combination could be

$$combination_2 = [p90, m60, p00, p60, m90, p60]. \tag{9.2}$$

However, this is not a feasible solution since the $p90$ row only has access to a $m60$ row, which is too small. Finding all possible permutations of a vector generates a large number of combinations. As an example, there are 3 628 800 possible permutations of a vector $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. For this reason, only unique combinations are extracted. This would mean that a vector $[1, 2, 2]$, only can be permutated as $[2, 1, 2]$ and $[2, 2, 1]$. The possibility to switch places of the two 2's is theoretically also a permutation of the original vector, but since the result is the same it is not of interest. Going back to the former example but changing the last number of the vector to a nine, $[1, 2, 3, 4, 5, 6, 7, 8, 9, 9]$, would still generate 3 628 800 combinations if non-unique solutions was included. However, if only unique combinations are

allowed, the number of permutations is decreased to 1 814 400, i.e. they decreased by a factor 2.

The expressions for calculating the number of permutations is $n!$ for non-unique permutations, where n is the amount of items. For unique permutations the amount of permutations is expressed as

$$\frac{n!}{n_1!\ n_2!\ ...\ n_k!}. \tag{9.3}$$

Still, for reasonably large car parks in this thesis, the number of rows become big enough to create a high number of permutations, causing the computational time to increase substantially. Since permutations grow factorial with the inputs, the result of decreasing the number of inputs is noticeable even for small changes. Due to this, the manoeuvre rows are excluded from the combinations vectors and saved elsewhere. The $combination_1$ vector from before would now take the form:

$$combination_1 = [p90, p60, p60, p00]. \tag{9.4}$$

Since it is optimal for two parking rows to share one manoeuvre row, at least one third of the values in the combination vectors will be removed. Removing at least one third of the elements in the vectors result in a significant decrease of permutations.

After all possible combinations have been generated, there will probably be combinations that does not fulfil the criteria for being a valid combination. A combination where a parking row of type 90° only has access to a manoeuvre row of type 60°, as shown in (9.2), is an example of a false combination. To exclude all false combinations, the known numbers of different manoeuvre rows, saved from before, are used. The permutation vectors are expanded with zeros in the positions for the manoeuvre rows to retake its original size. That would make the $combination_1$ vector take the form:

$$combination_1 = [p90, 0, p60, p60, 0, p00]. \tag{9.5}$$

For each permutation generated, the combination is run through and for all positions that are zero, the neighbouring positions with parking row types decides what type of manoeuvre row needs to be inserted. For the first zero in the $combination_1$ vector of (9.5), the manoeuvre row must be of type 90, since its neighbour with highest value is of type 90. After iterating through all elements in the combinations, the manoeuvre rows included are compared to the manoeuvre rows available. If the included manoeuvre rows do not match the available, the solution is not valid and the whole combination is discarded.

Before finishing the permutation process, another addition is made. If the last position in the combinations vector is a manoeuvre row, all valid combinations are duplicated and the duplicates flipped, in the sense that the last position becomes the first, and so on. This is necessary since the manoeuvre rows are not included in the process of finding all possible combinations, and when inserting manoeuvre rows, it is assumed that the first row is a parking row. To illustrate this, two new vectors, $combination'_1$ and $combination'_2$, are created, see (9.6). Since they have a manoeuvre row in the end of the vector, only $combination'_1$ is found by earlier

mentioned processes. By flipping them, both combinations are found.

$$\begin{aligned} combination'_1 &= [p90, m90, p60, p60, m60] \\ combination'_2 &= [m60, p60, p60, m90, p90] \end{aligned} \tag{9.6}$$

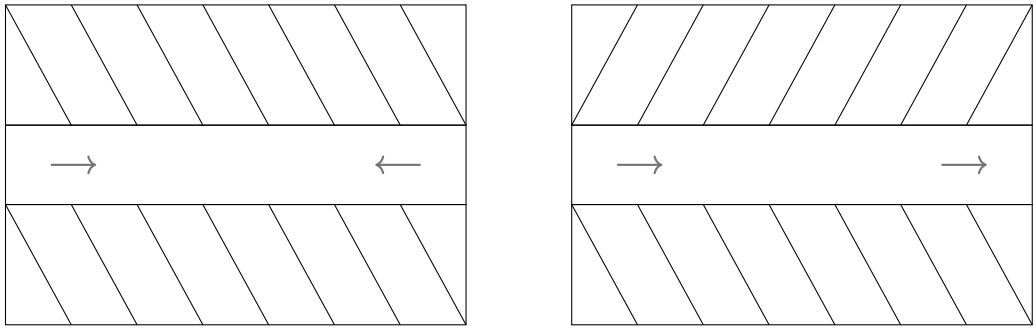
When double rows are included in the result from the solver, the whole process of finding permutations is done by considering any double rows as two single rows. After the process, all double rows are recreated. Although the process of eliminating invalid combinations also takes into account those combinations where it is not possible to create the right number of double rows from the temporary single ones.

9.2 Mirroring Tilted Parking Rows

When all possible combinations are extracted, the rest of the heuristics make the solution feasible for actual parking, and within that causing as little loss of spots as possible.

The rows that are tilted can not all be tilted the same way since their manoeuvre rows are dimensioned for one-way drive. If the tilted parking rows are all tilted the same way, a contradiction would occur, illustrated in Figure 9.1a. The spots in the lower parking row are made for entering from the left, as shown by the leftmost arrow. At the same time, the upper parking row is made for entering from the right, as shown by the rightmost arrow.

By simply mirroring the upper row vertically, a result as in Figure 9.1b is created, where both parking rows are made for being entered from the left side, as both arrows in the figure show.



(a) Tilted spots requiring different driving directions.

(b) Tilted spots requiring the same driving directions.

Figure 9.1: Tilted parking rows must create the same driving direction.

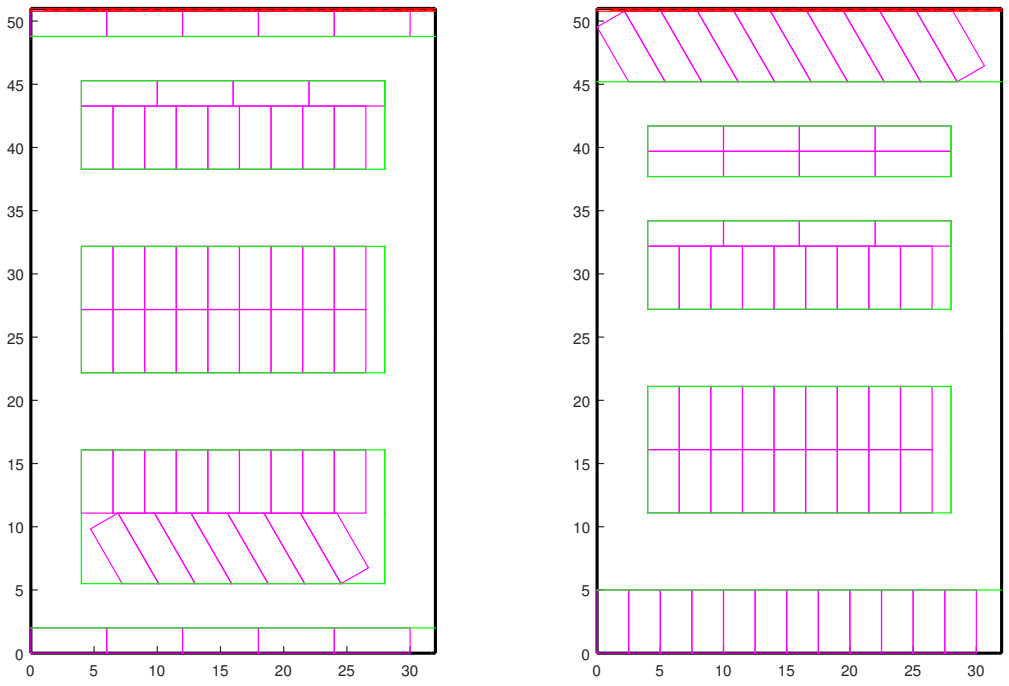
9.3 Vertical Roads

As mentioned in Section 7.1, the solution is modified by adding vertical roads to connect the manoeuvre rows and make them function as roads as well, thus making all spots in the solution reachable. The roads are added in the far left and far right.

The vertical roads could really be added anywhere. However, to avoid unnecessary waste space, as illustrated in Figure 7.2, they are placed in the margins.

By using the width of the vertical roads, as specified by the user as an input parameter, a new attribute is calculated for each road type. This is the number of spots fitting on one row if the row has a vertical road on each side, making the width of the row shrink by two times the width of the vertical road.

Combining this information with the allowed combinations, the best combination (or combinations) can be found, since they lose less spots. Logically it is better to have row types of a lower angle shortened by the vertical road, since they lose less spots than types of higher angle. Figure 9.2 illustrates how two different combinations of the same solution can differ due to the vertical roads. The dimensions of the car park is 32 m by 51 m. The combination in Figure 9.2a contains 57 spots and lost 16 spots from the solution without vertical roads, the variable *spotsLost* is used to describe this number. The combination in Figure 9.2b only lost 12 spots, and contains 61 spots.



(a) $spotsLost = 16$.

(b) $spotsLost = 12$.

Figure 9.2: Two combinations of the same car park, 32 m by 51 m, with vertical roads.

9.4 Presentation of Results

When all feasible combinations of the solution from the solver has been found, potentially flipped and vertical roads have been added (if so requested), the heuristics are done. The next step is to plot the result and calculate fitness values. To start with, fitness values are calculated. The fitness values are measures for how good the

solution is and are calculated to allow for comparison between different solutions, since this is one of the purposes of the program. Except for the already available total number of spots in the solution, another value is calculated. This is *areaPerSpot* and as the name suggests, it is a measure of the total car park area over the number of spots, as:

$$areaPerSpot = \frac{totalArea}{totalValue - spotsLost}. \tag{9.7}$$

Since these values depend on the combination of the solution, the fitness values are saved for each combination.

Finally, the solution is to be presented in a graphical way, easy for the user to understand. This is done using the plotting tools in MATLAB. Within the functions of plotting is also the function of saving the coordinates of each individual spot, the coordinates are the four corners of the spot. The coordinates are calculated from their angle and dimensions given as input, in combination with information about which way the row tilts and if there are vertical rows. The spots are always placed as far to the left as possible. Figure 9.3 shows a plot of a car park with a proposed solution containing double rows.

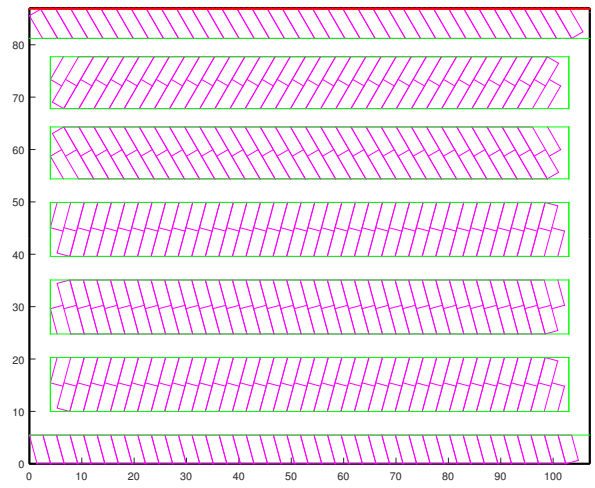


Figure 9.3: Car park, 107 m by 87 m, with vertical roads, spots = 430.

9.5 Iteration

An iteration function can be turned on or off in order to evaluate results from modifying the dimensions of the car park slightly. The whole optimisation process will run as many times as specified in the input, changing the size of the length or width of the car park accordingly to the specification in the input. When iteration is used, a plot displaying how the number of fitted spots change for the different car park dimensions is also output. Such a plot may show that for a small change in

car park dimension, there is a large change in number of spots, which might be of interest.

Another possibility is to compare the results from changing the dimensions of the parking and the manoeuvre rows. This is useful for analyses of autonomous parking effects, if the dimensions are decreased, maybe more spots fit into the solution.

10
Results and Analysis

The goal of this thesis has been to develop a tool for optimal distribution of parking spots, while allowing modification of input parameters as stated in Section 1.3. The tool also allows for comparison of potential increase in number of spots if the dimensions of the spots are reduced, e.g. for autonomous parking. All within the delimitations stated in Section 1.4.

In this chapter, results from the pattern formulation of the problem, as described in chapters 7-9, are presented and analysed. First, three examples of car parks are analysed, with regard to spot dimensions. One is a real car park in Gothenburg, called Skeppsbron, with a horizontal layout. The second car park has a vertical, layout while the third is a larger, square layout. These two layouts are fictive layouts, with dimensions set to test the program in various situations. Thereafter, an example of how the number of spots varies with the length or width of the car park is presented. Lastly, general data on different types of parking spots is analysed, showing how intuitive solutions can differ from optimal solutions.

10.1 Result Generation Data

Throughout this chapter, six angles are given the solver to optimise the solution, resulting in 10 types. The types are

$$type = [90, 75D, 75, 60D, 60, 45D, 45, 30D, 30, 00]. \tag{10.1}$$

For all results in this chapter, no value is given to how the solution affects the time to perform an actual parking. Neither is any evaluation done on how the driver (or the autonomous car) experiences the difficulty level of parking. However, all solutions are estimated to be feasible, as long as they contain vertical roads. In order to compare changes in parameters, vertical roads are not always included in the solution presented.

10.2 Parking Spot Reductions

In this section, three different car park layouts are used as input parameters, returning results to evaluate. Two of the spot dimensions are modified for comparison of possible increase in number of spots, as potentially usable by autonomously parking cars. These dimensions are *widthAcross* and *manLength*. The values that are tested are given in tables 10.1a and 10.1b. Every combination of different measurements, within the given sets, is evaluated in 12 different cases. One

of Volvo's widest cars, the Volvo XC90, has a width of 2.01 m including folded rear view mirrors [21]. Considering this, 2.1 m is regarded a suitable lower value of *widthAcross*. The dimensions used during analysis are taken from a parking layout guidebook [22], used by the Gothenburg City Parking Company. According to this source, a decrease in *widthAcross* requires an increase of *manLength*. So for lower values of *widthAcross*, *manLength* is first scaled up to fit that value, and the upscaled value possibly reduced, by a maximum of 15 %. In [22], graphs and expressions for angles in the range [0°, 90°] shows this connection between the two variables, for *widthAcross* taking the values 2.5, 2.4, 2.3 m. To extract more extreme cases, a linear connection between the three is assumed and corresponding values for *widthAcross* of 2.1 m are calculated.

In this chapter, the combination where *widthAcross* takes the value 2.5 and *manLength* is scaled accordingly, without any decrease, is referred to as the *standard dimensions*. The full list of measurements for the angles included in the analysis are shown in Table 10.5. Regardless of which case of *widthAcross* and *manLength* is considered, the length of all parking spots are always such that the *lengthAcross* is 5 m, except for the 0° spot type. The dimensions for 0° spots are somewhat divergent. The parking rows of 0° have a width of 6 m, instead of 5 m, in order to allow for parallel parking manoeuvres. The type also has a length of 2 m, instead of 2.5 m, like the others, since parking spots of this type do not block each other where the car doors would open.

In the following subsections, it is shown that depending on the size of the car park, reduced manoeuvre rows leads to one of two results; either an extra row of spots fit or the existing rows will "straighten up" to be rows of higher angles. The first case mostly applies for big car parks where many rows fit. Then the collective length from all reduced manoeuvre rows results in extra rows. If there are not many rows to start with, it is more likely that the result will be that the rows take types of higher angles instead, since no extra rows can be fitted into the solution efficiently.

Table 10.1: Values applied on the variables changed in the result generation.

(a) Values assigned to *widthAcross*..

widthAcross [*m*]
2.5
2.3
2.1

(b) Reduction on *manLength*.

manLength reduction [%]
-0
-5
-10
-15

In order to decrease the number of parameters affecting the result, vertical roads are not included in most of the results generated. This allows for fair comparison between results, since the program cannot handle obstacles.

Also noteworthy is that for each of the results presented, only one combination (see Section 9.1) will be displayed. For each layout there might be more combinations, the more type of rows in the solution the more possible combinations. Since

the results are extracted without vertical roads, the order of placement of parking rows does not affect the number of spots fitting in the car park and for this reason different combinations of layouts are not presented.

10.2.1 Horizontal Layout - Skeppsbron

A planned car park in Göteborg, Skeppsbron, has the layout according to Figure 10.1. The figure is scaled to fit the thesis page, for a upscaled design, see Appendix A. Although the design of Skeppsbron is already decided, it is a part of the Drive Me project and is evaluated for autonomous parking.

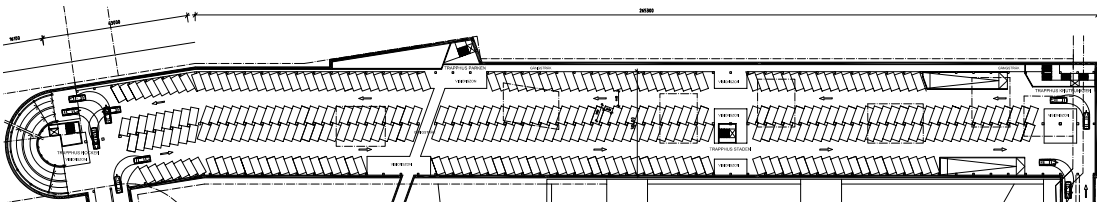


Figure 10.1: Layout of the planned car park Skeppsbron, Göteborg.

The dimensions of the spots in the existing layout are shown in Table 10.2.

Table 10.2: Dimensions for each parking spot in the Skeppsbron layout.

Variable	Value [m]
lengthAcross	5.0
manLength	4.4
widthAcross	2.4
φ	70°

The program created in this thesis does not handle obstacles such as elevator shafts, that are included in the existing layout. Therefor, a smaller part of the layout is chosen as input values on the car park. This part is illustrated by the red rectangle in Figure 10.2, with dimensions 66.75 m by 30.74 m, making it a total of 2052 m^2. This layout will be referred to as *Small Skeppsbron*.

Ignoring the fact that this piece of layout contains columns, it is suitable for comparison with a layout generated by the program. A comparison between the existing Small Skeppsbron layout and the layout the program generates, when given the values listed in Table 10.2, is thus done. The layout from the program is seen in Figure 10.3. Since the original layout includes 70° rows, this is included in this single comparison as well. For all other results, including the reduction analysis of Small Skeppsbron, 70° rows are not included. The layout from the program includes three such rows; one double 70° row, one single, and also one 75° row. The existing layout contains 100 spots, and the layout from the program generates 101 spots. This indicates that the program produces a layout that is not only the same as an existing layout, designed by an architect, but also gives a corresponding amount of

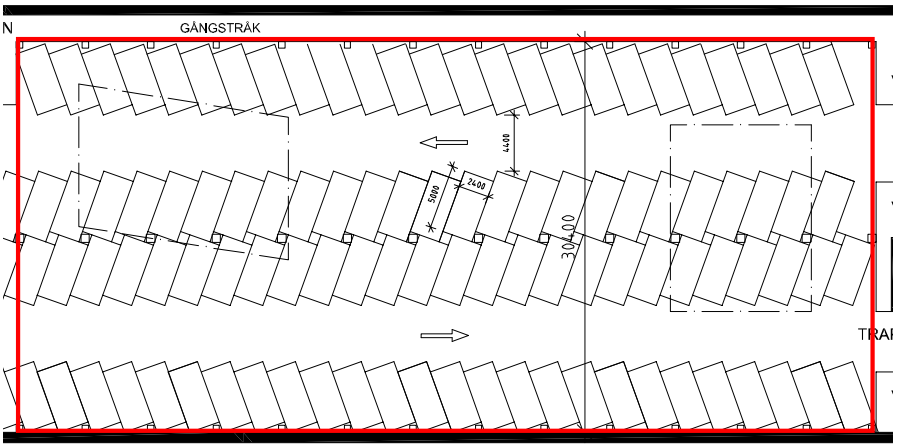


Figure 10.2: A piece of the Skeppsbron layout, with no obstacles but columns.

spots in its solution. The solution of the program is one spot better, although the actual Small Skeppsbron contains columns.

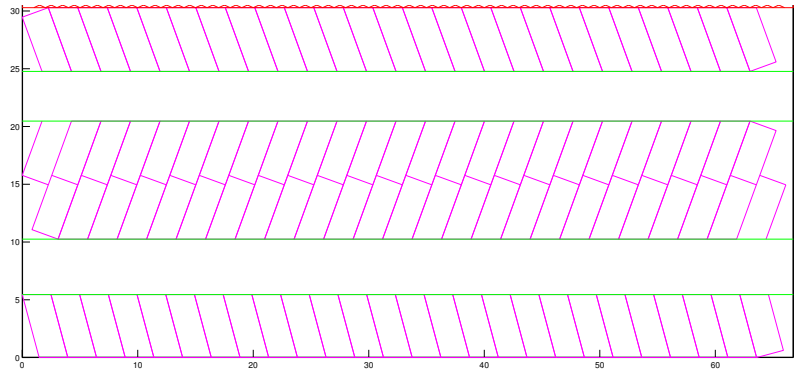
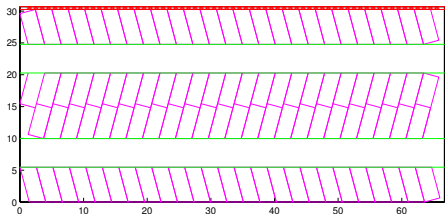
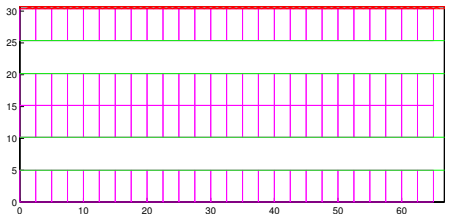


Figure 10.3: Small Skeppsbron layout, using same dimensions as existing layout.

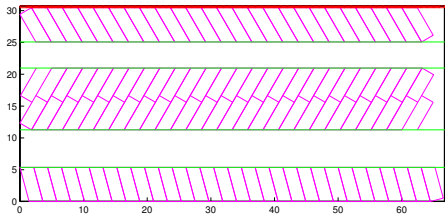
Analysis of the Small Skeppsbron with regard to reduction of *widthAcross* and *manLength* is also performed. The layout when using the standard dimensions of *widthAcross* and *manLength* includes in 99 spots, one less than the number in the existing layout. Note that the dimensions in the existing layout have a smaller value on *widthAcross*, which results in more spots even without reducing the *manLength* value, as will be shown in this chapter. When decreasing the values of these parameters, the largest number of parking spots fitting in the same area was 114 spots. Figure 10.4 shows four output results from the program, where the red part in the top of each figure is the wasted space, not claimed by any type of rows. In Figure 10.4a and 10.4b the value for *widthAcross* is the same, namely 2.5 m. The reduce in their *manLength* value on the other hand differs; 0 % compared to 15 %. As shown in 10.4b, the manoeuvre length is reduced enough to allow for rows of 90° spots, which are most efficient, in terms of width. In the same way, the results in Figure 10.4c and 10.4d have the same value on *widthAcross*, 2.1 m, but have different reduction on their *manLength* values, 0 % compared to 15 %. Even though the reduction is not big enough for allowing for rows of only 90° the program fits two such rows in Figure 10.4d.



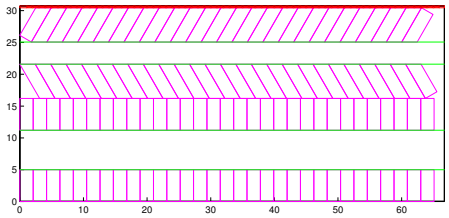
(a) $widthAcross = 2.5\,\text{m}$, $manLength$ $\pm 0\,\%$, spots = 99.



(b) $widthAcross = 2.5\,\text{m}$, $manLength$ $-15\,\%$, spots = 104.



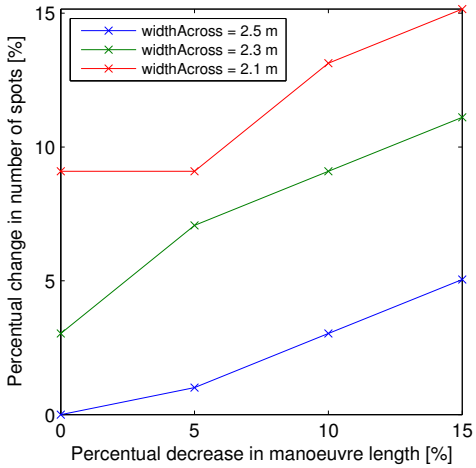
(c) $widthAcross = 2.1\,\text{m}$, $manLength$ $\pm 0\,\%$, spots = 108.



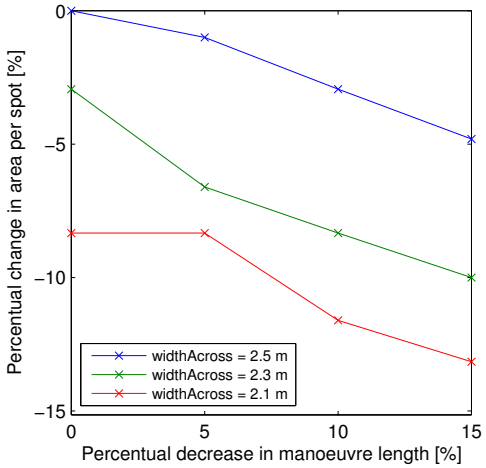
(d) $widthAcross = 2.1\,\text{m}$, $manLength$ $-15\,\%$, spots = 114.

Figure 10.4: Four resulting output layouts for the Small Skeppsbron layout.

The graph in Figure 10.5 shows the increase of number of spots from the decrease of the parameters. The values in the graph has been normalised to compare all values with the standard dimensions.



(a) Percentual change in number of spots.



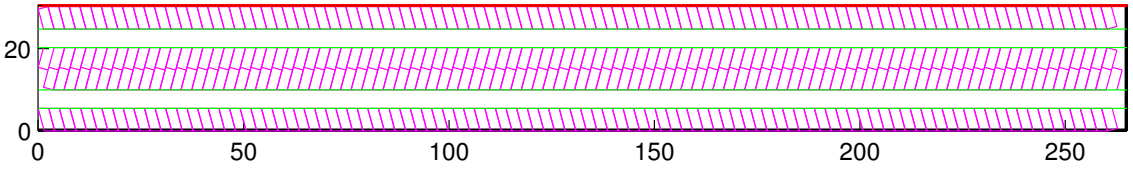
(b) Percentual change in area per spot.

Figure 10.5: Increase in spots and decrease in area per spot in Small Skeppsbron for decreasing values on $widthAcross$ and $manLength$.

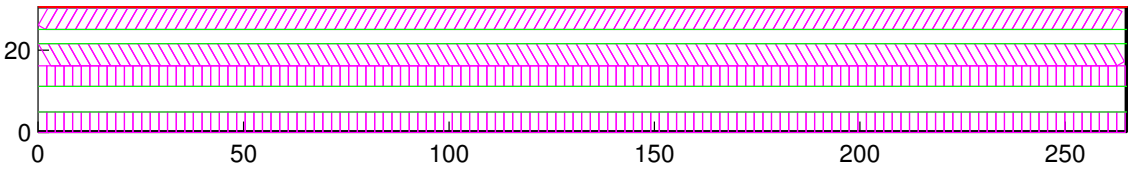
To get an estimate of how many spots that fit in the full Skeppsbron layout, a car park of dimensions 265 m by 30.74 m, making 8146 m^2, is evaluated in the same way in the program. These dimensions are referred to as *Big Skeppsbron* from here

on, and the width is chosen from the dimensions in the layout; the width of the oblong, rectangular part. As given in Figure 10.1, there are many obstacles in this part, thus a comparison of the existing and the resulting layout is not fair.

The resulting types of rows are the same for Big Skeppsbron as for Small Skeppsbron, two examples of the Big Skeppsbron layouts are shown in Figure 10.6a and 10.6b.



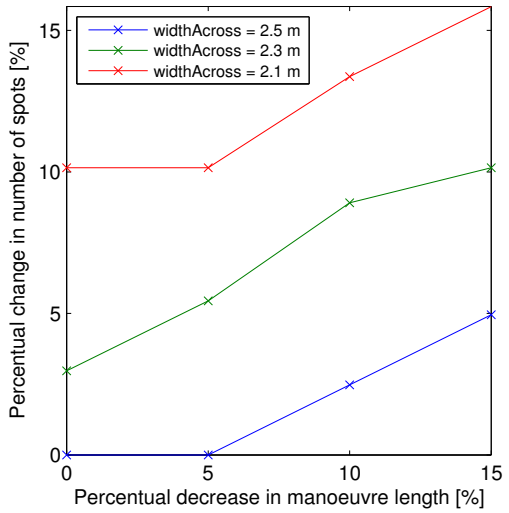
(a) *widthAcross* = 2.5 m, *manLength* ±0 %, spots = 404.



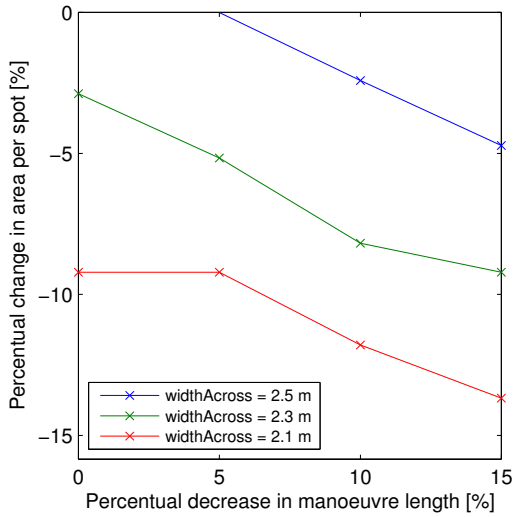
(b) *widthAcross* = 2.1 m, *manLength* −15 %, spots = 468.

Figure 10.6: Two resulting output layouts for the Big Skeppsbron layout.

The increase in number of spots is shown in Figure 10.7. The fact that the same types of rows are generated for both layouts makes the percentual graphs similar. The first figure shows 404 spots while the second shows 468, thus an increase of 15.8 %.



(a) Percentual change in number of spots.



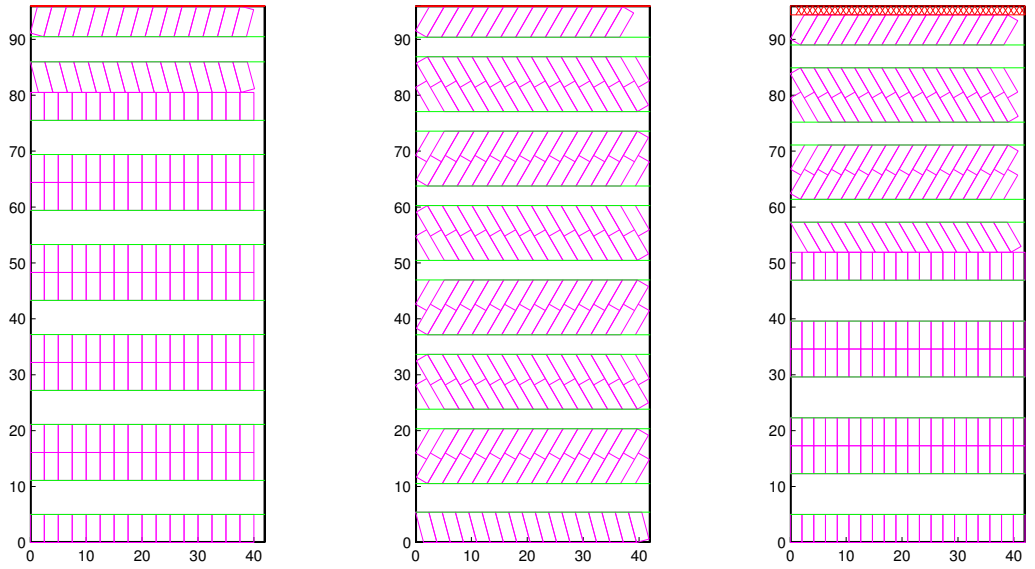
(b) Percentual change in area per spot.

Figure 10.7: Increase in spots and decrease in area per spot in Big Skeppsbron for decreasing values on *widthAcross* and *manLength*.

10.2.2 Vertical Layout

The Skeppsbron layout is horizontal and with such a low value on the car park length, the solver is limited to few possible solutions. What happens when the car park has a vertical layout instead? This section presents the result from the solver, given a car park with dimensions 42 m by 96 m and a total area of 4032 m^2, henceforth referred to as the *Vertical Layout*.

Three results of letting *widthAcross* taking values 2.5 m, 2.3 m and 2.1 m while *manLength* is not reduced at all are presented in Figure 10.8. The layouts differ in types of angles. Using standard dimensions results in almost exclusively 90° rows. When *widthAcross* is reduced, more rows are added, but of lower angle that requires shorter values on *manLength*. In the case with 2.1 m on *widthAcross*, some of the rows straighten up, increasing the value on the angles. Their resulting numbers of spots are 190, 211 and 216, respectively.

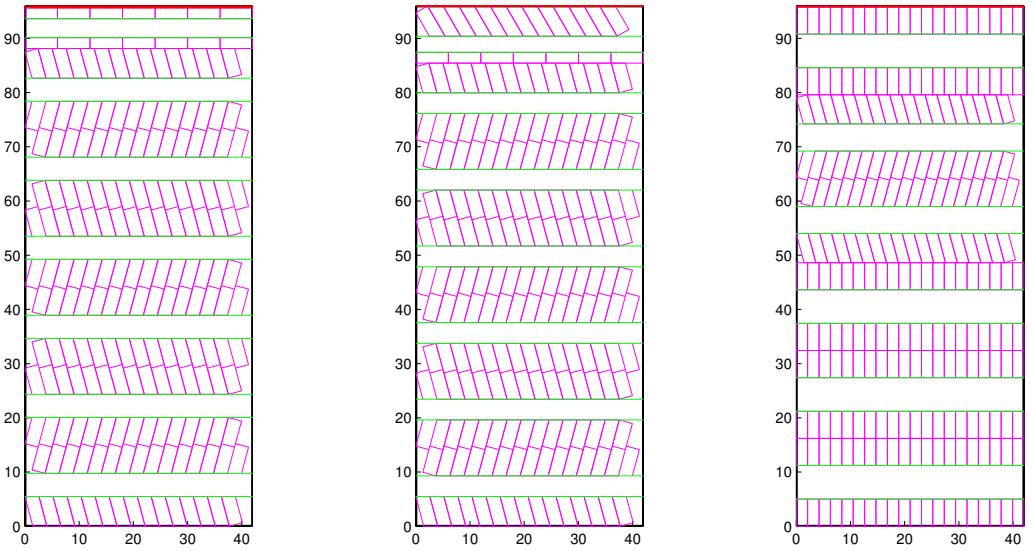


(a) *widthAcross* = 2.5 m, spots = 190. **(b)** *widthAcross* = 2.3 m, spots = 211. **(c)** *widthAcross* = 2.1 m, spots = 216.

Figure 10.8: Three layouts for the Vertical Layout when *widthAcross* is reduced and with *manLength* ±0° for all layouts.

Figure 10.9a and Figure 10.9b shows the case where the value on the variable *widthAcross* is fixed, 2.5 m, but decrease on *manLength* differs; 5 % and 15 %. These can be compared to the case with the standard dimensions, in Figure 10.8a.

The case where *manLength* is reduced by 5 % results in two rows of 0° parking rows, that probably is not the intuitive choice of row type. The choice of counterintuitive types is further analysed in Section 10.4. As in previous examples, the values of the angles is reduced somewhat for reduced dimensions, in order to fit more rows. Comparing the standard dimensions to cases where the *manLength* is reduced result in solutions containing 190, 194 and 200 spots, respectively. Figure 10.9c shows the case with highest reduction on both variables, *widthAcross* is 2.1 m and *manLength* is reduced by 15 %. The increase in number of spots between the



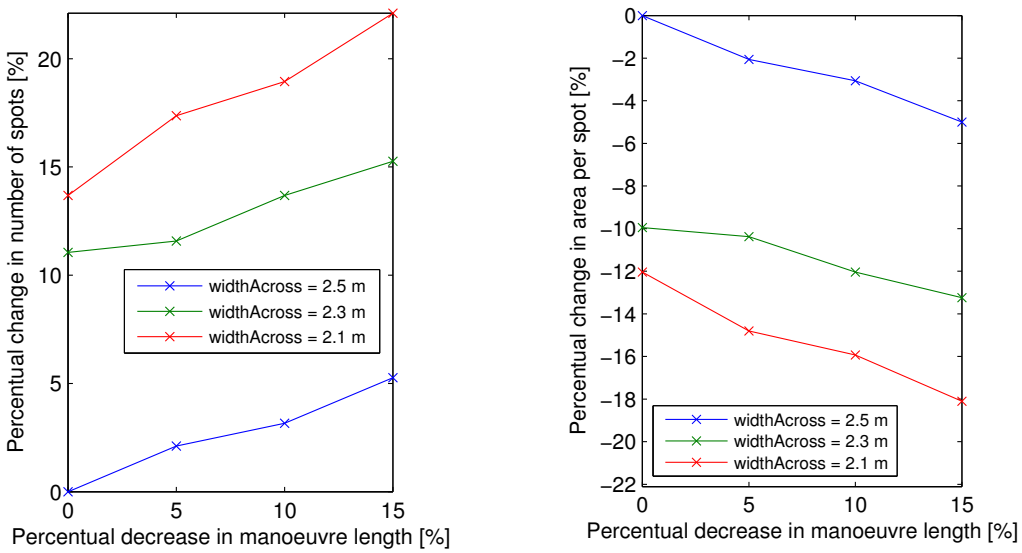
(a) *widthAcross* = 2.5 m, *manLength* −5 %, spots = 194.

(b) *widthAcross* = 2.5 m, *manLength* −15 %, spots = 200.

(c) *widthAcross* = 2.1 m, *manLength* −15 %, spots = 232.

Figure 10.9: Three layouts for the Vertical Layout.

standard dimensions and that in Figure 10.9c is 22.1 %. This yields an increase from 190 spots to 232 spots. The percentual increment is illustrated in Figure 10.10.



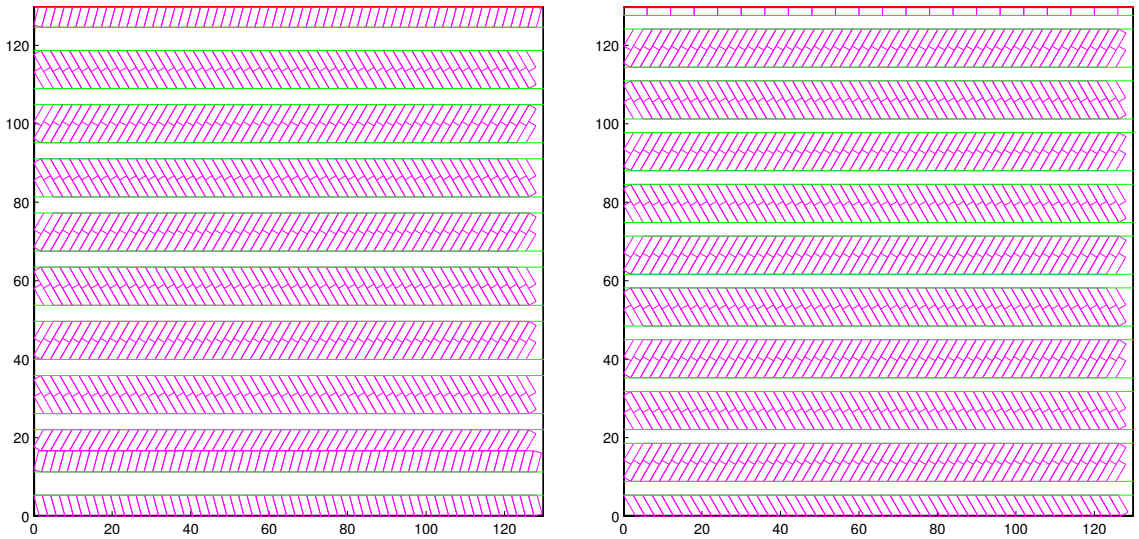
(a) Percentual change in number of spots.

(b) Percentual change in area per spot.

Figure 10.10: Increase in spots and decrease in area per spot in the Vertical Layout for decreasing values on *widthAcross* and *manLength*.

10.2.3 Square Layout

To complete the results from previous sections, a square layout is evaluated as well. The dimensions are chosen as 130 m by 130 m, resulting in a total area of 16 900 m^2. As such, this layout is the biggest in terms of area, and is referred to as the *Square Layout*.



(a) *widthAcross* = 2.1 m, *manLength* ±0 %, spots = 957.

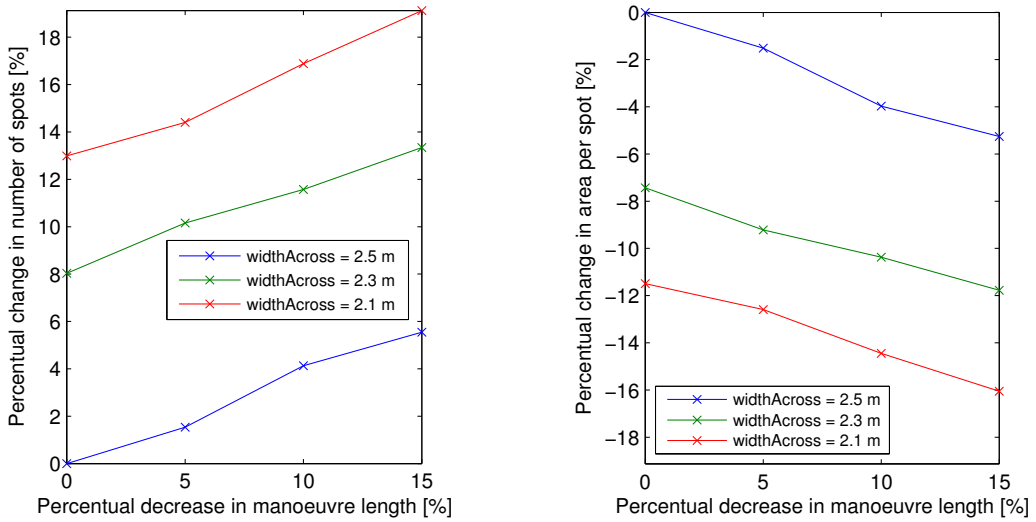
(b) *widthAcross* = 2.1 m, *manLength* −15 %, spots = 1009 spots.

Figure 10.11: Two resulting output layouts for the Square Layout.

As in the case of the Vertical Layout, the number of rows in the result with standard dimensions are enough to create new rows when shrinking the value of *manLength*. This is exemplified in Figure 10.11. In Figure 10.11a, *widthAcross* takes value 2.1 m and the manoeuvre length is reduced by 0 %. The number of parking rows in the layout is 18, when double rows are counted as two rows, and the total number of spots is 957. In the case where *widthAcross* has the same value but *manLength* is reduced by 15 % the corresponding layout shows 20 parking rows and a total 1009 spots. Thus, reducing the length of the manoeuvre rows by 15 % in this case renders 52 extra spots, or a percentual increment by 5.4 %. The overall percentual increase is 19.1 %, comparing the standard dimensions that results in 847 spots to the dimensions in Figure 10.11b, that yields 1009 spots. The complete data for the fitness values of the *Square Layout* are shown in Figures 10.12.

10.2.4 Impact of Orientation

The program created does not evaluate the orientation of the car park. For a square layout, as the one in Section 10.2.3, this obviously does not matter. In a case where the car park layout is rectangular and not square, the orientation will affect the resulting number of spots that fits in the solution. Intuitively, a horizontal car park would generate better results, since there are less parking rows that will lose spots



(a) Percentual change in number of spots.

(b) Percentual change in area per spot.

Figure 10.12: Increase in spots and decrease in area per spot in the Square Layout for decreasing values on *widthAcross* and *manLength*.

due to the vertical roads added after the optimisation, compared to its vertical counterpart.

A comparison is made by running the program for two of the layouts in previous sections, comparing its counterpart in orientation. First, the Vertical Layout is evaluated. Table 10.3 shows the result from evaluating the Vertical Layout, described in Section 10.2.2, as well as for the same layout rotated by 90°. The most extreme cases with parameters *widthAcross* and *manLength* are also compared. The table shows that a horizontal layout results in more spots than its vertical counterpart when vertical roads are added to the solution. Without vertical roads, the ordinary Vertical Layout contains four spots more than its horizontal counterpart. When the roads are added however, the vertical layout result in 184 spots, compared to 216 for the horizontal Layout. This applies when using a width of 4 m for the vertical roads, as stated in Table 10.3. The difference is 32 spots, or a 17.4 % increase.

In the case of Small Skeppsbron, described in Section 10.2.1, the horizontal layout generates more spots than its corresponding vertical layout, with or without vertical roads. Table 10.4 shows the corresponding results for Small Skeppsborn. A vertical version of Small Skeppsbron can fit up to 112 spots without vertical roads, while the horizontal layout fits 114. Adding vertical roads reduces spots for both cases. The vertical layout of Small Skeppsbron fits 88 spots with vertical roads, while the corresponding horizontal layout fits 107 spots. This is illustrated in Figure 10.13. Contrary to the results from the vertical layout, given in Table 10.3, the percentual increase in number of spots when reducing *manLength* and *widthAcross* is here bigger for the horizontal layout, considering the case of vertical roads included in

[1]Using the standard dimensions as described in Section 10.2.

[2]Using least value on *widthAcross* and highest reduction on *manLength*.

Table 10.3: Results using the Vertical Layout, in its original orientation and a corresponding horizontal layout.

Layout	Road width [m]	Spots min[1]	Spots max[2]	Spot incr. [%]	Area / spot max[1] [m^2]	Area / spot min[2] [m^2]
Vertical	0	190	232	22.1	21.68	17.38
Vertical	4	160	195	21.9	25.20	20.68
Horizontal	0	196	228	16.3	20.57	17.68
Horizontal	4	184	216	17.4	21.91	18.67

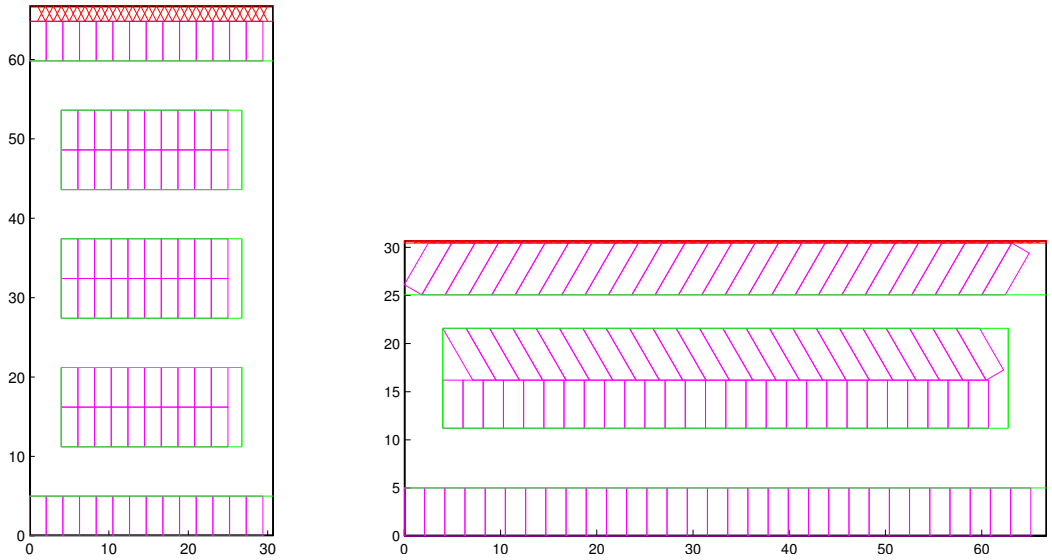
the solution.

Table 10.4: Results using the Small Skeppsbron layout, in its original orientation and in its corresponding vertical layout.

Layout	Road width [m]	Spots min[1]	Spots max[2]	Spot incr. [%]	Area / spot max[1] [m^2]	Area / spot min[2] [m^2]
Vertical	0	96	112	16.7	21.37	18.32
Vertical	4	78	88	12.8	26.31	23.32
Horizontal	0	99	114	15.2	20.72	18.00
Horizontal	4	93	107	15.1	22.06	19.18

The orientation also affects the computational time, since higher numbers of rows leads to a possibility of more unique permutations if there are different types of rows. The case of Big Skeppsbron is an example of this. When rotating its layout into a vertical one, the solver returns the following types of parking rows (manoeuvre rows are not included since they do not affect the total number of permutations): 30 rows with angle 90°, 2 rows with angle 75° and 2 rows with angle 0°. Even though it is only three types of rows, it generates 278 256 unique permutations, according to (9.3). The number of permutations in total is $n!$, resulting in 2.95×10^{38} permutations, when counting non-unique permutations. This requires a lot of computer power, and when trying to generate all permutations, MATLAB ran out memory[3]. Even if a more powerful computer is used for the task, 2.95×10^{38} is still an unreasonable number of permutations and preferably another way to find all unique permutations is required.

[3]Using these computer specifications : Intel Core i7-4810MQ CPU @ 2.80GHz, Windows 7 Enterprise, 64-bit.



(a) *widthAcross* = 2.1 m, *manLength* −15 %, spots = 88.

(b) *widthAcross* = 2.1 m, *manLength* −15 %, spots = 107.

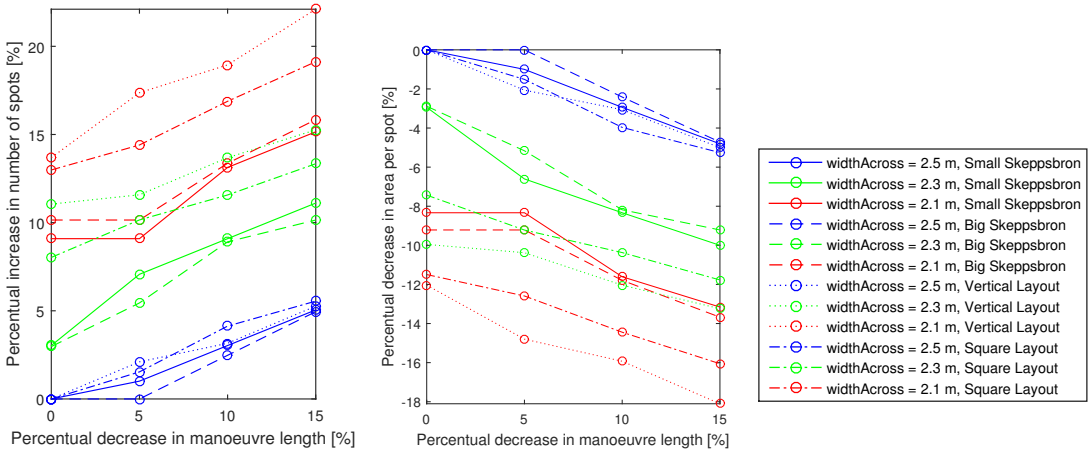
Figure 10.13: Comparison between vertical and horizontal orientation for the Small Skeppsbron layout, vertical roads included and set to 4 m.

10.2.5 General Trends

Comparing the percentual increment in number of spots for the evaluated layouts shows that the trend is similar for all four. In Figure 10.14, fitness values of all layouts evaluated are merged for comparison. Figure 10.14a shows that both reductions in *manLength* and *widthAcross* generate an increase in number of spots. The increase from *widthAcross* reduction however is slightly larger. Figure 10.14a shows that the percentual number of extra spots generated from reducing *manLength* by 10 % yields at least the same results as reducing *widthAcross* by 8 % (0.2 m). By reducing *widthAcross* by 16 % (0.4 m) compared to the standard dimensions gives a further increment in percent of number of spots. There is no general answer to the question of which dimension modification that is most efficient, however, these results would suggest that *widthAcross* is slightly more favourable.

10.3 Car Park Dimension Iteration

During real world creation of car parks, the dimensions of the car park are often variable, within some interval. It can then be of interest to evaluate how the parking spot layout changes, if one dimension of the car park is slightly modified. To allow for such investigations, an iterative functionality is included in the program, as described in Section 9.5. This iterative functionality is illustrated in Figure 10.15. This figure show iteration of car park dimensions, width and length, with a step size of 0.25 m and 10 iterations in both positive and negative direction from the original size, 60 m by 60 m. Figure 10.15a illustrates how the number of spots change with car park width. A pattern is shown in Figure 10.15a, illustrating that the value



(a) Percentual change in number of spots in all four layouts.

(b) Percentual change in area per spot in all four layouts.

(c)

Figure 10.14: Percentual change in fitness values for varying values on *widthAcross* and *manLength*, a comparision between the four layouts.

changes in ramps when the width changes. This shows the fact that the value does not change until a certain width is reached, at which, many rows simultaneously fit one additional spot. To maximise how the area is used, it is preferable to be placed just above a ramp.

As for the length, the increase in number of spots varies in a more irregular way. This depends on the fact that the program changes the type of rows used in the solution, with the varying length of the car park. In this case, as in Figure 10.15b, the original dimensions 60 m by 60 m are well placed on the graph, just above a steep part.

10.4 Counterintuitive Layouts

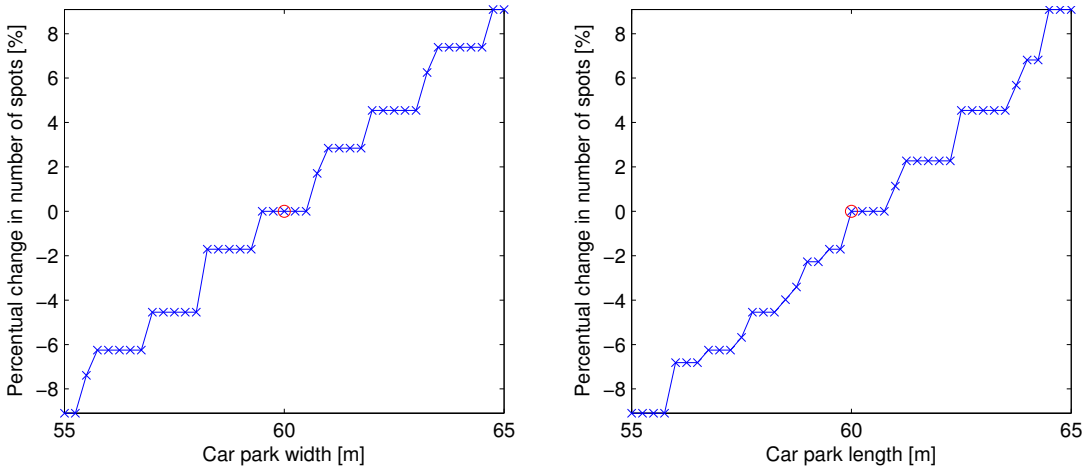
Intuitively, 90° rows are considered most efficient when it comes to planning the car park. This is due to the fact that the spots create no waste space; not in the actual spot or at the edge as $unusedWidth$, see Figure 7.2.

The approach *Tile and Trim* as described by Porter et al. [10], follows this intuitive assumption that 90° parking spots are optimal and tiles such spots in each car park as a rough estimate of maximum number of parking spots.

When investigating which angle of parking is most efficient, one way of looking at efficiency is according to (10.2). In this expression, i denotes the different parking types in the same way as throughout this chapter, with the exception that no double types are included. The term $pValue_i$ is calculated as specified in (7.2).

$$efficiency_i = \frac{2pValue_i}{2pLength_i + manLength_i} \tag{10.2}$$

Assuming a length precisely matching the length of two parking rows of the same



(a) Number of spots depending on car park width.

(b) Number of spots depending on car park length.

Figure 10.15: Change in total number of spots, depending on the dimensions of the car park.

type, and a corresponding manoeuvre row, efficiency is calculated. Since $pValue_i$ depends on the width of the car park, the efficiency is given regarding that.

This way of describing efficiency is interesting because it resembles the way the program in this thesis works, which is: checking number of fitted spots horizontally, and then fitting combinations of rows vertically. As mentioned in Section 10.2, standard dimensions for a parking spot are $widthAcross$ and $manLength$, such that the spot always fits a rectangle of 2.5 m by 5 m. This data is also included in Table 10.5. Using this data, efficiency numbers have been calculated for car park widths between 1 m and 200 m. For each car park width, the number of spots fitted horizontally has been calculated, and is then divided by the length of two parking rows of the corresponding angle, and one manoeuvre row. The results are presented in Figure 10.16.

To allow for easier comparison between angles, the graph has been normalised by division with the current car park width. As given in the figure, angles 90° and 75° are most efficient in this aspect. Again, one could then assume that these angles of spots would be best for any car park, this is however not true. Unless the car park dimensions are open for manipulation in order to exactly fit rows of these angles, the true optimum parking angles will depend on the length of the car park.

One example of when the optimal packing includes counterintuitive parking angles is shown in Figure 10.17. In this figure, a car park of dimensions 20 m by 11.1 m is input to the program. The length is precisely enough to fit one row of 90° parking spots with corresponding manoeuvre length. As shown the figure however, the program instead chooses to include one row of 60° and 0° spots each. This leads to one additional spot being fitted into the car park.

Another, larger, example of counterintuitive packing is shown in Figure 10.18. In this figure, two examples of packing for a car park of dimensions 42 m by 41 m are displayed. Figure 10.18a shows the layout when angles of {90, 75, 60, 45, 30, 00} degrees are allowed. Figure 10.18b shows the layout when only using 90° spots. The

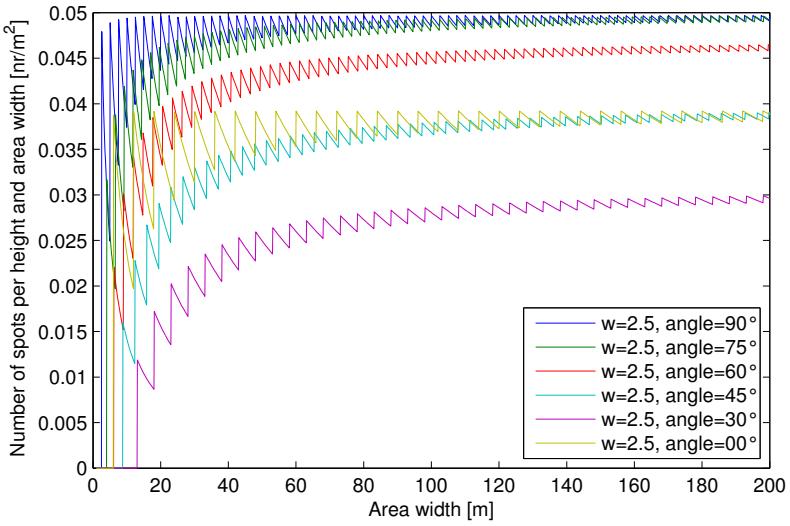
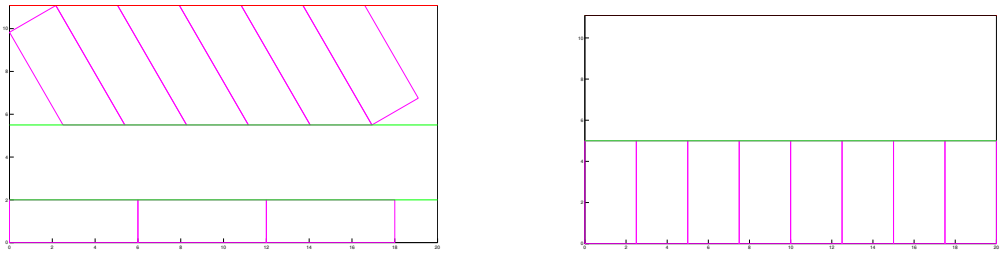


Figure 10.16: Change of length efficiency, *spots / required length*, for car park widths 1 m - 200 m.

Table 10.5: Dimensions on parking spot types used in this chapter. Data are taken from [22].

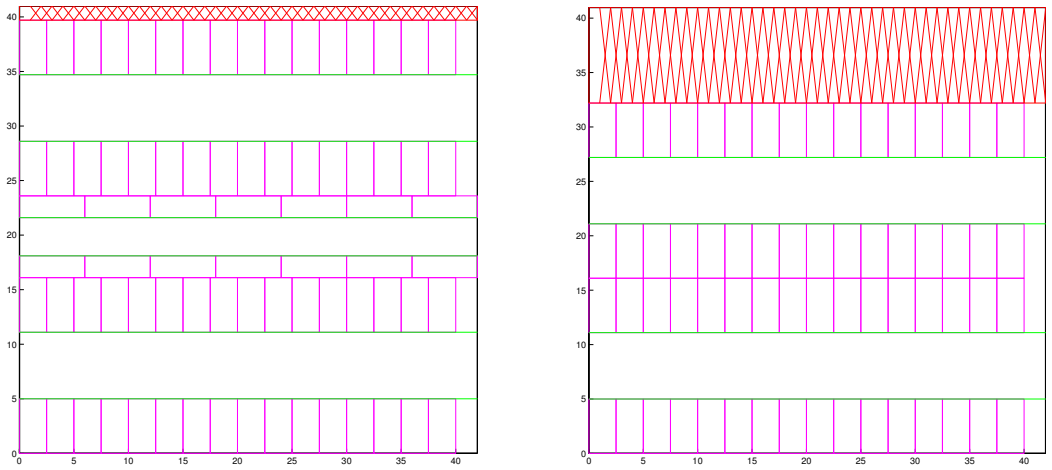
Corresponding dimensions [m]	Parking spot angle					
	90°	75°	60°	45°	30°	0°
width	2.50	2.59	2.89	3.54	5.00	6.00
widthAcross	2.50	2.50	2.50	2.50	2.50	2.00
unusedWidth	0	1.47	3.22	5.30	8.08	0
length	5.00	5.48	5.58	5.30	4.66	2.00
manLength	6.10	4.50	3.50	3.50	3.50	3.50

parking spot dimensions used are according to the parking layout guidebook [22]. The number of spots for the two is 78 for Figure 10.18a and 64 for Figure 10.18b.



(a) Counterintuitive layout, spots = 9. **(b)** Intuitive layout, spots = 8.

Figure 10.17: Comparison between intuitive and counterintuitive layout for car park of size 20 m by 11.1 m.



(a) Counterintuitive layout, spots = 78. **(b)** Layout including only 90° parking spots, spots = 64.

Figure 10.18: Comparison between intuitive and counterintuitive layout for car park of size 60 m by 58 m.

11
Conclusion and Discussion

This chapter will discuss the results of the thesis, compared to the goal stated in Chapter 1. It will also discuss the results of the analysis presented in Chapter 10 and finish up with a review on possible future work.

11.1 Objectives

Stated in Section 1.3 is the goal of this thesis:

> *The goal of the thesis is to, with mathematical methods, optimise the distribution of parking spots in a parking area in order to maximise the number of spots. The optimisation methods should be applicable for use with both autonomous and manual vehicles.*

This section will conclude how the developed program fulfils all parts of the goal that was set.

11.1.1 Mathematical Methods

The program which has been developed throughout the thesis is based on MILP, a sub-genre of mathematical programming and optimisation. It also utilises heuristics which use mathematics to efficiently find permutations. The use of MILP assures that whatever solution is reached, it is the optimum for the problem modelled. The MILP model is formulated as to maximise the number of parking spots within a given area, thus returning an optimal distribution of parking spots in this sense.

11.1.2 Applicability for Autonomous and Manual Parking

The program's input parameters are car park dimensions, parking spot dimensions, including manoeuvre requirements, and parking spot angles. These parameters are sufficient to model different kinds of parking vehicles such as manually driven and autonomously parking vehicles, as shown in Chapter 10 when the parking spots are made narrower and manoeuvre rows shorter. The only requirements on the inputs is that 90° spots and 0° spots are included, because of the way the program is written. Adding a correct number of types of spots is important. The user might include all integer angles within the range [0°,90°], but the more angles added, the longer the computational time will be. This is partially on account of the solver, that has more decision variables to take into account. Secondly, the more types of parking

rows included in the solution from the solver, the more unique combinations for the heuristics to find, which by extension also leads to longer computation times.

When defining the car park dimensions, the user can specify them as to model the car park vertically or horizontally. As we show in Chapter 10, it is often favourable to layout a horizontal car park, since they are both faster to calculate and give better result, especially when including vertical roads.

11.1.3 Program Output

The program generates results for all car parks tested, of which the largest was 130 m by 130 m. The upper limit on how large car parks the program can handle is hard to say. Partly because it depends on how many angles are included as inputs, but mainly because it depends on the solution that the solver generates. It is not necessarily the decision of which parking rows to use that is the hard part, but in which order they should be placed, with respect to vertical roads.

It is in some cases possible to achieve the same number of spots through more than one set of rows. For example, for some size of car park, the same number of spots might be achieved by using one 90° parking row, as the number of spots when using one 60°, and one 0° parking row. In such cases, the rows that the solver outputs depend on circumstances, such as the computer or solver used. Such cases has been found by using the program on two different computers. In its current state, the solver only outputs one set of rows, and it is not possible for the user to know if there are other possible sets that achieve the same result. From a maximum parking spot point of view, different sets are of no interest. However, from an architects point of view, who might further manipulate the layout to include obstacles for example, it may well be of importance.

11.2 Results

The results from the analysis in Chapter 10 show a number of things. First and foremost they show that the program can be used to generate practical parking layouts. The comparison with an existing layout done in Section 10.2.1 show that the layout generated by the program is in level with the layout created by an architect. They are similar both in number of spots, and in choosing which parking rows to include.

The analysis done in Section 10.2 show that the program can be used to compare different kinds of parking models, such as autonomously parking and manually parking vehicles. The results show that there is potential benefits to be had if autonomously parking vehicles can operate on spaces with the dimensions used during the analysis. The trends for different parking layouts also show that there is more advantages to be achieved by reducing the measurement on *widthAcross*, than reducing the measurement correspondingly on *manLength*.

How the program can be used to evaluate different orientations and measurements of car park dimensions is also exhibited, in sections 10.2.4 and 10.3. In Section 10.2.4 we see that, for the dimensions evaluated, it is more advantageous to let the car park assume a horizontal orientation. Section 10.3 show that the relation between car

park dimensions and number of spots is non-linear. With that stated, it is clear that the program can be used to evaluate potential benefits of changing aforementioned dimensions.

Lastly, Section 10.4 presents especially unintuitive layouts, further establishing the fact that sometimes the optimal solution is very hard to guess. The program however has no problem finding it for car parks of such size that the heuristics can handle the rows included. As stated in Section 11.1.3, it is hard to know where this limit is.

As previously stated, no general mappings for relationships between any input and number of spots is done. This is because of the complexity of the problem. The number of spots depend on all variables simultaneously.

11.3 Future Work

The program in its current state works well and fulfils the requirements specified for the thesis. It is a good tool for analysing the effect of parking spots with decreased dimensions, maybe due to capabilities of autonomously parking cars, and to get a draft of the design to work on. To get even better results, that requires less after-treatment by the user, some functions could be implemented. These functions are discussed in this subsection.

11.3.1 Adapting to Heuristics

The optimisation is performed with regard to the full width of the car park, even if vertical roads are included. If vertical roads are included, the rows in the solution are optimised for the full car park width and then cut off if they are not placed in the top or bottom of the car park, as shown in Figure 9.2. A better way might be to estimate how many rows will be in the solution. If this number is large enough that most spots will be in a row next to the vertical roads, the optimisation is performed with regard to the width of the car park minus the two vertical roads instead, adding spots for the rows in the top and bottom. In this way, the solution would be closer to the true optimum, since it is less modified by heuristics than the solution in this thesis.

11.3.2 Additional Functions

Multiple sizes of parking spots, together with desired fractions of each size could be of use to generate layouts which include both spots for autonomous parking and manual parking. This could be used to include handicap parking spots as well.

A lot of focus during parking layout design is put on not making the roads include large, unnecessary, driving distances. This has not been included in the thesis, but could be implemented as a heuristic after optimisation. Important to consider when including more heuristics is that the more heuristics, the higher the likelihood that the result deviates from the optimal solution.

A function that would increase the usefulness of the program is if it could be designed to handle car park areas that are not of rectangular shape. As it is designed

now, it is possible to split an area into smaller rectangles and find the best solution for each of them. This might give results that do not match up well, for example if the parking rows are placed on different lengths, making a manoeuvre row match up with a parking row.

11.3.3 Obstacles

Possibilities to include obstacles in the formulation could help architects find new ways to fit parking spots around obstacles. In the current formulation however, our guess is that optimising around such obstacles would be difficult, but including them in heuristics would be possible. The reason for this is that the MILP formulation is modelled in one dimension, the length dimension, since all rows share the same width. If obstacles like columns where to be included, the solver has to handle another dimension, the width dimension. Also, the solver does not handle any positions in the current formulation, it only sees to that the rows fit in some combination. An obstacle would have a position, thus positions must be included in the formulation as well.

Including obstacles in the heuristics instead would mean that the optimal solution is modified and hence introducing a risk of a non-optimal solution. This might be acceptable, as in the case with the vertical roads, and with the different combinations the best can be found. The implementation of including obstacles is still difficult to implement, since it requires taking into account the position of all spots, in two dimensions. In the program, this is not taken into account before the presentation of the result. Furthermore, as complicated as it is to calculate the impact of obstacles interfering with parking spots, it may be even more complicated to evaluate how obstacles interfere with manoeuvre rows. This would introduce more decisions regarding roads to bypass the obstacle.

11.3.4 Spot Formulation

The computational complexity of the spot formulation from Part I shows how complicated optimisation problems of this kind can be. In its current state, the spot formulation is not useful, since it cannot generate layouts for car parks of interesting size.

The problem with the current formulation is that it generates large numbers of integer variables. In order to make the spot formulation work for larger sizes of car parks, one would have to find some way of formulating the problem in order to avoid these integer variables. Especially, some way of avoiding the need for variables for each pair of parking spots would improve the formulation.

If it is possible to formulate the problem in such a way that its computation time is decreased enough to solve the problem, its solution would be at least as good as that of the pattern formulation. Assuming that it would be possible to formulate the spot formulation as to allow any degree on parking spot, the feasible region of the spot formulation would be a superset of the feasible region of the pattern formulation. If the spot formulation would not be able to handle other angles than $0°$ and $90°$, it would not necessarily be better than the current pattern formulation.

11.3.5 Ethical and Sustainability Matters

When building and travelling an area, to save space is to save money. Less required space means a smaller lot to occupy and less material to expend into building the structure. Shorter distance to travel means less fuel to consume, assuming that the speed at which vehicles can operate within the area is uncorrelated to the dimensions. Maximising the number of vehicles within an area is equivalent to minimising the needed area for a given number of vehicles. To minimise the needed area is to conserve money. Assuming that a parking space has a regular width of 2.5 meters and assuming that the average car is approximately 2 meters wide, it is clear to see that just the matter of stepping out the car before driving into the spot, can save up to 20 % of space [21] [22]. In the local environment, less space occupied by unwanted structures, such as parking areas, means more space left for people. The continued possibility to use cars increases our independence and freedom.

It could be argued that by making cars take up less space, we enable the use of more cars. From an environmental viewpoint, this is not sustainable. Cars currently account for 30 % of Sweden's carbon dioxide emissions [23]. Even if all cars were made to run on electricity completely, it is still not sustainable to build such large constructions as cars for each person, considering our ever increasing population.

Automation in large can be used to perform tasks currently handled by people. One can argue for both positive and negative impacts of this. The program created in this thesis in its current state is nowhere near replacing a human car park architect. In its current shape, it is a helpful tool for giving an architect a good start to work from, and also useful for comparing different cases, when such options exist.

Bibliography

[1] United Nations Department of Economic and Social Affairs, *World's population increasingly urban with more than half living in urban areas*, [Accessed 18 January 2016], Jul. 2014. [Online]. Available: `https://www.un.org/development/desa/en/news/population/world-urbanization-prospects.html`.

[2] Todd Litman, *Land for vehicles or people?*, [Accessed 11 February 2016], Nov. 2014. [Online]. Available: `http://www.planetizen.com/node/72454/land-vehicles-or-people`.

[3] J. Nyhus, *I framtidens stadstrafik tar bilen mindre plats*, [Accessed 11 February 2016. Article in Swedish], Dec. 2015. [Online]. Available: `http://www.gp.se/nyheter/debatt/1.2920477-i-framtidens-stadstrafik-tar-bilen-mindre-plats`.

[4] G. Grahn-Hinnfors, *Och bilarna ska gömmas i p-hus*, Mar. 2010. [Online]. Available: `http://www.gp.se/nyheter/goteborg/1.324678-och-bilarna-ska-gommas-i-p-hus`.

[5] Lindholmen Science Park, *Volvo car group initierar världsunikt pilotprojekt med självkörande bilar*, Dec. 2013. [Online]. Available: `http://www.lindholmen.se/nyheter/volvo-car-group-initierar-varldsunikt-pilotprojekt-med-sjalvkorande-bilar`.

[6] S. Cook and L. Levin, "The complexity of theorem proving procedures", 1971.

[7] R. Karp, "Reducibility among combinatorial problems", *Complexity of Computer Computations*, 1972.

[8] A. Kobetski, "Optimal coordination of flexible manufacturing systems with automatic generation of collision- and deadlock-free working schedules", PhD thesis, Chalmers University of Technology, 2008.

[9] E. S. Thorsteinsson, "Hybrid approaches to combinatorial optimisation", PhD thesis, Carnegie Mellon University, May 2001.

[10] R. Porter, "Optimisation of car park designs", University of Bristol, Tech. Rep., 2013.

[11] J. Lundgren, M. Rönnqvist, and P. Värbrand, *Optimization*. Lund, Sweden: Studentlitteratur, 2010.

[12] R. Macedo, C. Alves, and J. M. V. V. de Carvalho, "Exact algorithms for the two dimensional cutting stock problem", in *Column Generation*, 2008.

[13] A. Lodi, S. Martello, and D. Vigo, "Recent advances on two-dimensional bin packing problems", *Discrete Applied Mathematics*, vol. 123, Nov. 2002.

[14] T.-Y. Yu, J.-C. Yang, Y.-L. Lai, and H.-Y. Chang, "Applying an enhanced heuristic algorithm to a constrained two-dimensional cutting stock problem", *Applied Mathematics & Information Sciences*, vol. 9, Feb. 2015.

[15] M. Hifi, R. M'Hallah, and T. Saadi, "Approximate and exact algorithms for the double-constrained two-dimensional guillotine cutting stock problem", *Computational Optimization and Applications*, vol. 42, Mar. 2009.

[16] D. Pisinger, "Denser packings obtained in $\mathcal{O}$(n log log n) time", *INFORMS Journal on Computing*, vol. 19, Jul. 2007.

[17] J. Egeblad and D. Pisinger, "Heuristic approaches for the two- and three-dimensional knapsack packing problems", Department of Computer Science, University of Copenhagen, Tech. Rep., 2006.

[18] E. Huang and R. Korf, "Optimal rectangle packing: An absolute placement approach", *Journal of Artificial Intelligence Research*, vol. 46, 2012.

[19] Brooks/Cole, *4.10 - the big m method*, `http://www.columbia.edu/~cs2035/courses/ieor3608.F05/david-bigM.pdf`, [accesed 2016-06-16], 2003.

[20] Jaroslaw Tuszynski, *Xml_read.m*, [Accessed 6 June 2016], 2006. [Online]. Available: `http://www.mathworks.com/matlabcentral/fileexchange/12907-xml-io-tools/content/xml_read.m`.

[21] Volvo Car Group, *Volvo car support - ägarmanual online (owner's manual online)*, [Accessed 30 May 2016], Feb. 2016. [Online]. Available: `http://support.volvocars.com/se/cars/Pages/owners-manual.aspx?mc=v526hbat&my=2016&sw=15w46&article=871e942e897ca77dc0a801511788660a`.

[22] Ingenjörsvetenskapsakademin, Transportforskningskommissionen, *Parkeringsanläggningar*. Stockholm, Sweden: Kommissionen, 1969.

[23] Trafikverket, *Vägtrafikens utsläpp*, Jun. 2013. [Online]. Available: http://www.trafikverket.se/om-oss/var-verksamhet/sa-har-jobbar-vi-med/miljo-och-halsa/klimat/transportsektorns-utslapp/vagtrafikens-utslapp/.

A

Appendix 1

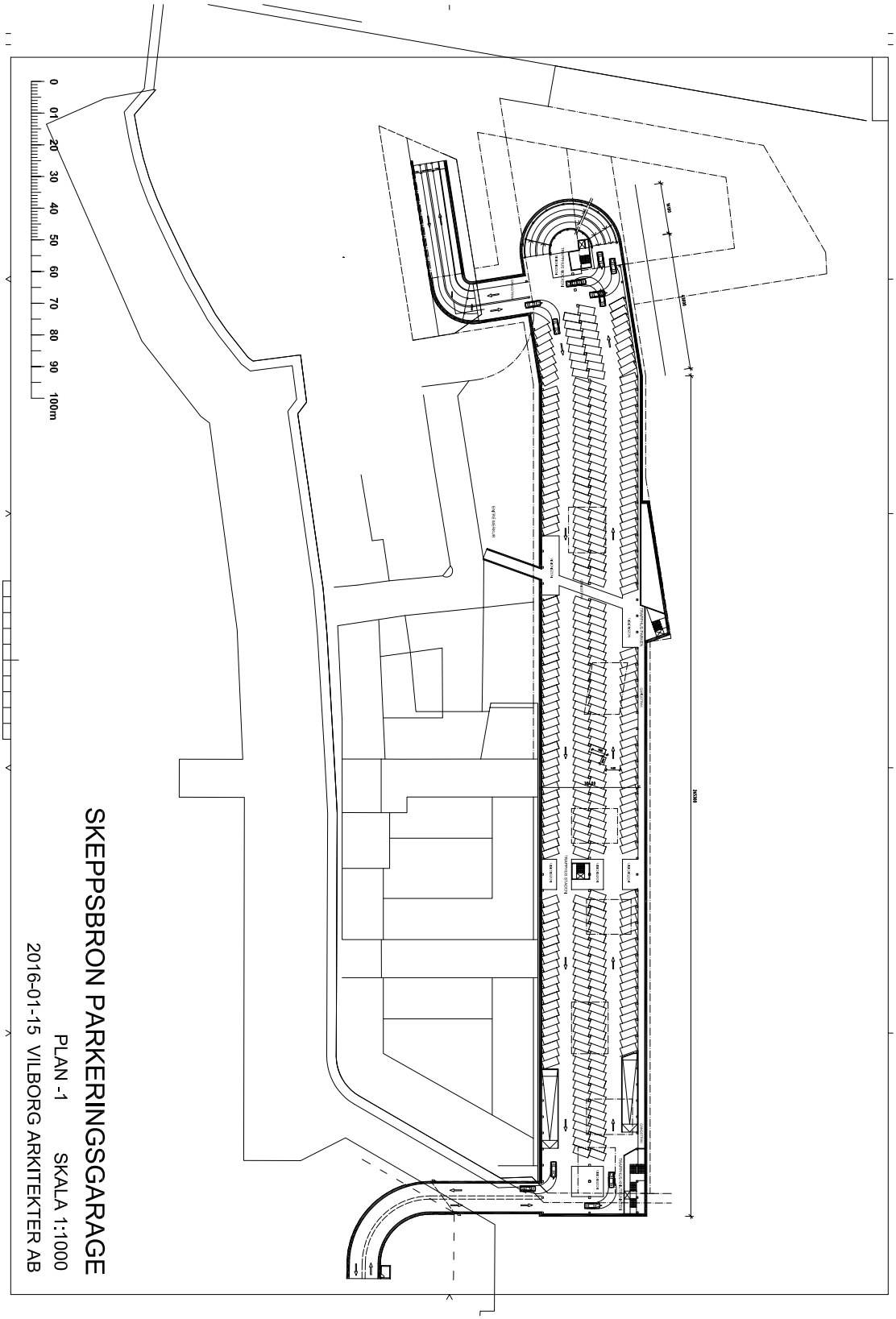


Figure A.1: Full drawing of the Skeppsbron layout.