FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Kristýna Pantůčková

Optimization of a circulating multi-car elevator system

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis: RNDr. Jiří Fink, Ph.D.
Study programme: Informatika
Study branch: Obecná informatika

Prague 2018

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In date

I dedicate this thesis to my supervisor for his guidance and my family for their support.

Title: Optimization of a circulating multi-car elevator system

Author: Kristýna Pantůčková

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Jiří Fink, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Circulating multi-car elevator is a system holding multiple cars in two shafts, where cars move upwards in one shaft and downwards in the other shaft. This system is similar to the paternoster, but cars have to stop on floors and open doors to load and unload passengers. Besides many technical challenges, this system brings algorithmic problems regarding efficient control of all cars. This thesis studies an off-line optimization problem, where the most efficient elevator system is searched for a fixed set of passengers. For this purpose, we created a computer program, implementing a genetic algorithm for searching for the most efficient elevator control and a discrete event simulation for evaluation of the efficiency of the control. The program provides a graphical user interface for input of parameters, generating passengers and displaying the results.

Keywords: genetic algorithm optimization elevator systems

Contents

Introduction

One of the most important issues in the design of high buildings is an efficient vertical transportation system [1]. For this purpose, escalators, moving walks and elevators are used. Elevators are more suitable for travelling over large vertical distances. Moreover, escalators are not sufficient for transport of disabled persons [2]. In addition to classic single-car elevators, other types of elevators were invented to improve the efficiency of vertical transportation. For instance, double-decker elevators, consisting of two decks which can serve two adjacent floors at the same time [3], decrease the number of elevators needed to serve the building by increasing the elevator capacity.

Another type of elevator is the paternoster (see Figure 1), which holds multiple cars circulating in two shafts without stopping. The paternoster elevator can transport more passengers than conventional elevators [4]. However, the paternoster is not safe for transporting children, elderly persons and disabled persons. In the Czech Republic, paternosters are no longer subjects of standards (ČSN EN 81) and regulations (government regulation no. 122/2016 Sb.) related to elevators, which implement European standards (95/16/ES) [5]. Nowadays, paternosters are considered to be lifting devices.

In 2013, the Hitachi company proposed a new type of elevator, called circulating multi-car elevator (CMCE), which is inspired by paternoster, but its cars have closable doors and stop on floors to load and unload passengers [6]. CMCE holds multiple cars circulating in two shafts. Cars move upwards in the first shaft until they reach the top of the shaft. Then they move to the second shaft and continue to move downwards until they reach the bottom of the shaft. Afterwards they move back to the first shaft and continue to move upwards. The advantage of this type of elevator is that its two shafts with multiple cars require less space than multiple shafts of conventional single-car elevators. The Hitachi company developed a 1/10 scale prototype of CMCE holding six cars. In their prototype, opposite cars are connected to balance each other and thereby decrease the energy consumption.

This thesis is focused on optimization of the control of CMCE. A common way of improving efficiency of the vertical transportation is splitting floors to sectors and assigning these sectors to cars, both either statically or dynamically [1]. Another option is applying certain rules based on the current traffic type, for example prioritizing upwards or downwards calls. Efficient elevator control can also be achieved by using an optimization method, for example linear programming, neural networks or genetic algorithms, for searching for the best assignment of passengers to cars. While evaluating elevator systems, different types of traffic must be considered, because demands on elevators vary in different parts of a day and the elevator system should be efficient under all conditions.

In this thesis, we use an off-line optimization algorithm to search for the optimal assignment of passengers to cars. In contrast with on-line algorithms, which process problems without the knowledge of the future situation, off-line algorithms receive the whole problem and search for the solution using all the data [7]. The solution of an off-line algorithm can be used for assessing the efficiency of other algorithms. In this thesis, we use a genetic algorithm for off-line optimiza-

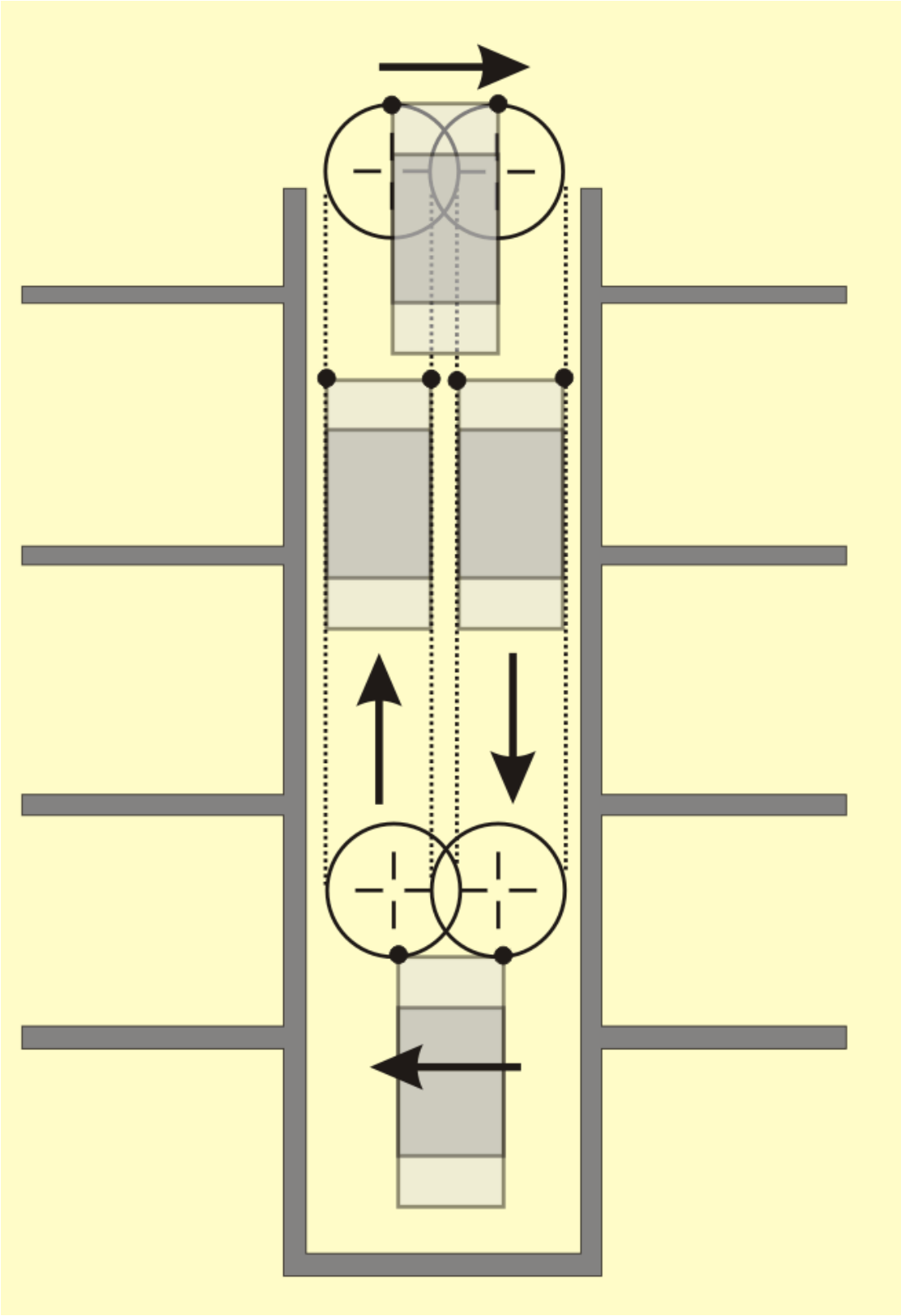

Figure 1: Paternoster. By Helfmann [Public domain], from Wikimedia Commons, `https://upload.wikimedia.org/wikipedia/commons/b/b2/Paternoster.png`

tion. Genetic algorithms are iterative algorithms inspired by biology [8].

The aim of this thesis is to study CMCE, understand its behaviour and search for the optimal CMCE system control. For this purpose, a computer program was created within this thesis. The program provides a graphical user interface for input of parameters of the building, elevator system and traffic and a simple animation of the elevator control.

This text contains a description of the subject studied in the thesis, explanation of the method used for optimization, a case study focused on optimization of concrete CMCE systems, examples of approaches to elevator system optimization and documentation of the program.

Because searching for the most efficient CMCE control using a genetic algorithm is computationally expensive and takes a lot of time, in the case study we compared CMCE with different properties in different traffic types in the trivial control, sending the nearest car to a passenger. Then we used a genetic algorithm to find an efficient CMCE control for a few sets of passengers and compared the results with the trivial control.

1. Model description

This chapter describes the technical background which is used to simulate and optimize a CMCE system. A model in a simulation consists of a building, an elevator system and passengers. An elevator system is a system consisting of one or more CMCE. One elevator holds one or more cars.

1.1 Building

The CMCE system is evaluated in office buildings, where on each floor, certain number of persons have their offices. Properties of a building include:

- the number of the lowest floor
- the number of the highest floor
- the height of a floor
- for all floors the number of persons who have their offices on that floor
- the time required for boarding and alighting of a passenger

We assume that each floor has the same height. A building must contain a lobby, which is assumed to be on the floor number zero. We assume that the time required for entering a car and the time required for leaving a car are equal and they are the same for all passengers.

1.2 Elevator system

1.2.1 Properties of an elevator system

An elevator system consists of one or more elevators. An elevator is characterized by:

- the number of cars in an elevator
- the capacity of a car
- the time required to open the door of a car
- the time required to close the door of a car
- the maximum velocity of a car
- the acceleration of a car
- the deceleration of a car
- the height of a car
- the minimum distance between cars

- the time required to move a car from one shaft to the other shaft

Properties of all elevators in the system are the same. The maximum velocity, acceleration and deceleration are assumed to be the same in both shafts. All elevators serve all floors.

1.2.2 Rules for the movement of cars

It is assumed that before the beginning of the simulation, the traffic is sparse. Therefore, at the beginning of a simulation, cars are staying on floors, they are not moving and their doors are closed. Half of the cars are situated, in one shaft and half in the other shaft with even distances between each other. When passengers arrive, cars start to move towards them. Passengers board a car in the order of their arrival. Immediately after the last passenger alights or enters, the car starts closing the door and does not wait for passengers, who have not arrived yet.

To move to the other shaft, a car must stop at the end of a shaft and after a certain time the car is placed at the end of the other shaft. Only one car at a time can occupy the top space of shafts used for moving cars form one shaft to the other shaft. The same holds for the bottom space of shafts. The minimum distance is always preserved between two cars.

In the situation where the path towards a car's next stop is blocked by another car, the car moves towards the next car, until it must stop to preserve the minimum distance. The blocking car is ordered to move away. After a forced stop, the car starts to move again, when the distance between cars is large enough. The blocking car notifies the waiting car, after it starts moving, with a delay of two seconds, because the future path of a car is computed from the current distance from the preceding car.

Cars serve calls assigned to them in the order in which they pass the floors. A car stops on all floors where any passengers want to exit the car and on all floors, where any passengers assigned to this car can enter the car. A fully occupied car only stops on floors, where passengers want to get off the car.

1.3 Traffic

1.3.1 Types of the traffic

CMCE system control is evaluated during periods, in which the demands on the elevator system are expected to be the highest. These periods include the up-peak, down-peak, lunch-peak and interfloor [1]. Up-peak occurs in the morning, when people arrive at the building and travel to their offices. Down-peak occurs in the evening, when people travel to the ground floor to leave the building. Lunch-peak occurs at noon, when some people travel to the floor where a canteen or a restaurant is situated and some people leave the building. Interfloor traffic consists of people travelling between various floors.

The passengers considered in the simulation are either loaded from a file or generated on the basis of the up-peak, down-peak, lunch-peak or interfloor traffic. Types of traffic are defined by total number of passengers, time period of arrival times of passengers and ratios of journeys to or from special floors and journeys

of general traffic. Special floors include floor number zero for up-peak, down-peak and lunch-peak traffic and the floor with a canteen for lunch-peak traffic. General traffic consists of journeys between various floors.

1.3.2 Generating passengers

In some types of traffic, many passengers travel to or from one special floor. When passengers are generated, firstly the requested number of passengers with the special floor at one end of their journey is generated. At the other end of their journey, there is the floor on which their office is located. In the up-peak traffic, these passengers are passengers travelling from the lobby to their offices. In the down-peak traffic, these passengers are passengers travelling from their offices to the lobby. In the lunch-peak traffic, these passengers include passengers travelling from their offices to the floor with a canteen, and passengers travelling from their offices to the lobby.

Secondly, the general traffic is generated. General traffic consists of passengers travelling from their offices to random floors and passengers travelling from random floors to their offices. The up-peak, down-peak and lunch-peak traffic include certain amount of general traffic. The interfloor traffic consists only of journeys of general traffic. If the number of passengers included in the traffic is lower than total number of persons in the building, a random subset of all possible passengers is selected. The arrival time of a passenger is a random number falling in the requested time period.

1.4 Optimization criterion

Quantities commonly used for measurement of passenger satisfaction include for example, the average waiting time, average travel time and average journey time [1]. In this thesis, we use the average waiting time (AWT), defined as the average time elapsed between the arrival of a passenger and opening of the door of the car to which the passenger is boarded, as the optimization criterion. This criterion was chosen, because it corresponds well with the satisfaction of passengers with the elevator system. The travel time, defined as the time spent in the elevator, and the journey time, consisting of both the waiting and travel time, are more interesting for employers, since lengthy times spent in elevators affect the productivity of employees.

2. Optimization

This chapter describes the method used for off-line optimization and details of the implementation. For searching for an efficient assignment of a fixed set of passengers to cars, we use a genetic algorithm.

2.1 Trivial control

In order to evaluate the efficiency of a CMCE system control found by our implementation of a genetic algorithm, we compare it with a trivial algorithm. In the trivial control, the car with the shortest distance from the required floor is sent to that floor, since it is expected to arrive first. If the car cannot stop on the floor due to its current position and velocity, the distance from the required floor involves the whole way around the elevator. If a car becomes fully occupied before an assigned stop, the next nearest car is called.

2.2 Genetic algorithm

This section contains a general description of genetic algorithms and details of our implementation.

2.2.1 Genetic algorithms

Genetic algorithms are iterative algorithms inspired by evolution in the nature. In a genetic algorithm, individual solutions are represented by chromosomes. Chromosomes consist of genes. Firstly, the initial population consisting of random chromosomes is created. From the population, a new population, representing the next generation of chromosomes, is created by crossbreeding and mutation [8]. For each chromosome, the probability of crossbreeding with another chromosome and passing its traits to the next generation is affected by its cost. The lower the cost of a chromosome, the better the solution represented by the chromosome is.

There are more methods for selecting parents of new chromosomes. Parents can be chosen randomly. Another option is to sort the population by cost and pair the best chromosome with the second best chromosome, the third with the fourth, and continue in this manner to the worst chromosome. Selection of parents can be also based on their costs or their ranks in the population. Another method of selection is the tournament selection method. In this type of selection, a small number of chromosomes is randomly selected from a population and the best of them is chosen as a parent.

Children are created from parents by a crossover method. One possible crossover method is the single-point crossover. This method chooses a point splitting a chromosome to the left and right part. One child inherits the left part from the first parent and the right part from the second parent and the second child inherits the left part from the second parent and the right part from the first parent. In the two-point crossover method, a chromosome is split to three parts

and a child inherits either the middle part, or the left and the right part from a parent. Another method of crossover is the uniform crossover, where a child inherits each item in the chromosome with the probability of 50 % from the first parent and with the probability of 50 % from the second parent.

A genetic algorithm can include mutation, which can add new useful traits to a population. Mutation is a random change of random genes in a population. It is characterized by the mutation rate, which is defined as the ratio of the number of mutated genes and the total number of genes in the population. The algorithm can exclude the best chromosomes, called elites, from mutation.

2.2.2 Genetic algorithm in our implementation

In our implementation, costs of chromosomes are calculated by a discrete event simulation. A chromosome is represented by a list of integers representing cars assigned to passengers. This list is sorted by arrival times of passengers in ascending order. This implementation uses the tournament selection method, which was chosen, because it is easy to implement. We select unique chromosomes for each tournament. We implemented the single-point crossover method. The first half of passengers in the new chromosome inherits cars from one parent and the second half from the other. Mutation is realized by changing random assigned cars in random chromosomes to different random valid car numbers. After the mutation, the worst chromosomes in the population are replaced by new random chromosomes. This action removes the worst solutions from the population and also brings new potentially useful traits to the population. The evolution is stopped after the requested number of iterations and returns the best solution from all iterations.

Since instances are evaluated by a discrete event simulation, which is computationally expensive, the single-point crossover method was chosen to take advantage of partially computed simulations. When children are created, they use partial simulations of their parents, since they share the first half of their assignments of passengers to cars with one of them. A partial simulation is created and saved during a simulation by creating a deep copy of a model, when half of the passengers arrive. To avoid computing costs of chromosomes with the same assignments again, costs of chromosomes are saved. When the number of saved costs reaches its bound, the oldest cost is deleted.

Parameters of the evolution include:

- the number of iterations
- the size of a population
- the percentage of elites in a population
- the percentage of chromosomes, which will be replaced
- the number of rivals in the tournament selection
- the mutation rate
- the value, by which the mutation rate decreases between generations

- the minimum mutation rate

Parameters of the evolution can be configured by a user. If the difference of the mutation rate is non-zero, the mutation rate decreases at the end of each iteration by this value, until it reaches the minimum mutation rate.

3. Results and discussion

This chapter contains results of simulations in two buildings with one CMCE with different properties. Because searching for the optimal CMCE control using a genetic algorithm takes a lot of time, we decided to compare CMCE under different conditions in the trivial control and then find a more efficient CMCE control for some sets of passengers. In both buildings, simulations of the trivial control were run with different number of cars in the up-peak, down-peak, lunch-peak and interfloor traffic with different number of passengers. We have searched for the ideal number of cars for CMCE with the trivial algorithm of control in each building regarding AWT. To obtain more representative results, we computed AWT for 10 different sets of passengers and compared the average value of AWT for different number of cars, types of traffic and number of passengers.

Then we have investigated for concrete sets of passengers, how efficient the trivial algorithm is, using the genetic algorithm. For this purpose, we selected a few examples whose AWT was high in the trivial control and tried to find a better elevator control with tolerable AWT. We expect, that waiting times of passengers arriving later will be higher in worse elevator control, because the elevator will be overloaded and will not be able to serve existing passengers together with incoming passengers. Therefore, we compared the dependence of AWT on the minute of arrival in the trivial control and the optimized control found by the genetic algorithm for these examples.

Properties of both buildings and their elevators are:

- the height of a floor: 4 meters [9]
- the time required for boarding or alighting of a passenger: 1 second
- the number of persons, who have their office on a floor: 30 (for all floors)
- the capacity of a car: 10 persons
- the time required for closing of the door: 2 seconds
- the time required for opening of the door: 3 seconds
- the maximum velocity: 3 m/s [10]
- the acceleration: 1 m/s^2 [11]
- the deceleration: 1.5 m/s^2
- the time required to move a car from one shaft to the other shaft: 5 seconds
- the height of a car: 2.5 meters [12]
- the minimum distance between two cars: 25 cm

The up-peak traffic consists of 90 % passengers travelling from the lobby to their office and 10 % general traffic. The down-peak consists of 90 % passengers travelling from their office to the lobby and 10 % general traffic. In the lunch-peak traffic, there are 20 % passengers travelling to the lobby, 60 % passengers

travelling to the floor number two, where the canteen is situated, and 20 % general traffic. The trivial control has been simulated with sets of 100, 200 and 300 randomly generated passengers arriving during the time period of 15 minutes.

3.1 Simulations in a ten-storey building

3.1.1 AWT in the trivial control with different number of cars and 300 passengers

In a 10-storey building, the average AWT was the highest in the down-peak (see Figure 3.1). The average AWT with 300 passengers decreases with increasing number of cars. With very high number of cars, where cars have less space for their movement, the average AWT slightly increases, especially in the up-peak (see Figure 3.2). The ideal number of cars seems to be around 12, where the average AWT is lower than 8 seconds in the up-peak, lower than 10 seconds in the lunch-peak and interfloor and lower than 18 seconds in the down-peak.

3.1.2 AWT for different number of passengers

With less cars, the average AWT is higher with higher number of passengers (see Figure 3.3, Figure 3.4, Figure 3.5 and Figure 3.6). With more cars, the average AWT is similar with different number of passengers except for the down-peak, where the average AWT for 300 passengers is the highest for all numbers of cars.

3.1.3 Improving AWT by increasing car capacity

Waiting times can be sometimes lowered by increasing the capacity of a car. For example, in the down-peak with 300 passengers and 6 cars, high waiting times are partly caused by the fact, that cars are not stopping on lower floors while moving downwards, because they are fully occupied. Passengers travelling from lower floors to the lobby have very high waiting times (see Figure 3.7). By installing CMCE elevator with cars of capacity 15, the average AWT of these passengers would significantly decrease. To achieve the average AWT under 18 seconds, 9 cars would suffice. Ideal number of cars with a capacity of 15 would be around 10, where the highest AWT is about 13 seconds.

By increasing the car capacity to 15 persons, the average AWT decreases, especially with lower numbers of cars, except for the interfloor traffic (see Figure 3.8). With 9 and more cars, the average AWT is lower in the up-peak traffic, where most passengers board on one floor, and in the down-peak traffic, where most passengers board cars in one shaft (see Figure 3.9). However, increasing the car capacity is not helpful in the lunch-peak and interfloor traffic.

In the up-peak traffic, the average AWT with 4 and more cars is lower with higher number of passengers (see Figure 3.10), where high waiting times caused by small capacity of cars are eliminated. In a more dense traffic, it is more likely, that a car is already moving towards the floor on which a new passenger arrived. Moreover, in a sparse traffic, cars spend more time waiting, until motionless cars, which are blocking their path, move away.

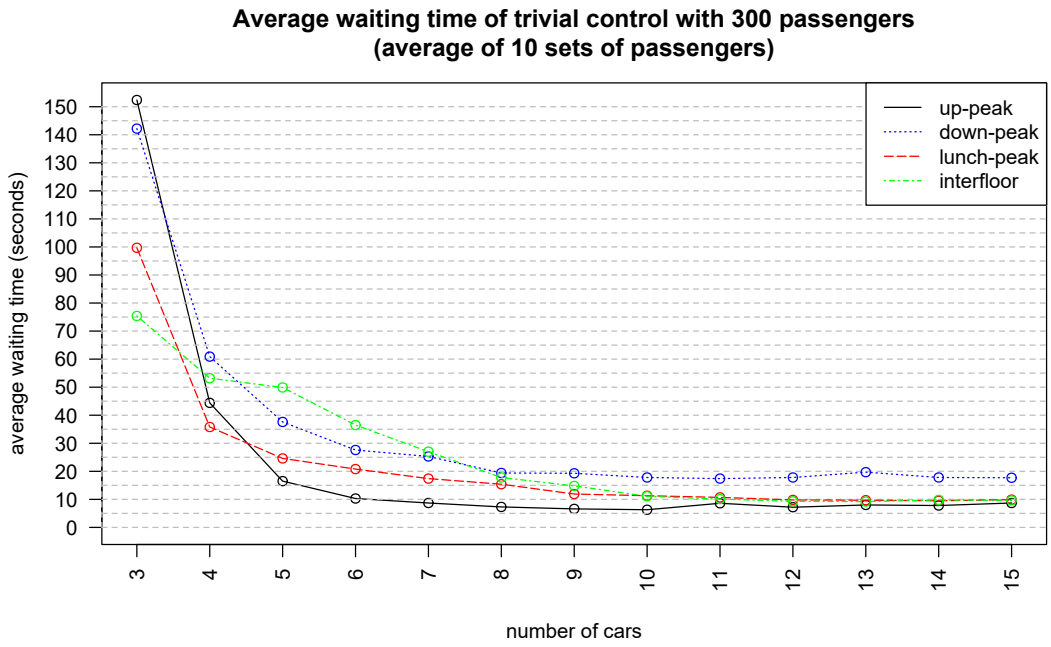

Figure 3.1: AWT in seconds in a ten-storey building with one CMCE with different number of cars. Values on the y-axis are average values of AWT of ten different sets of passengers. The CMCE was evaluated in different types of traffic with 300 passengers.

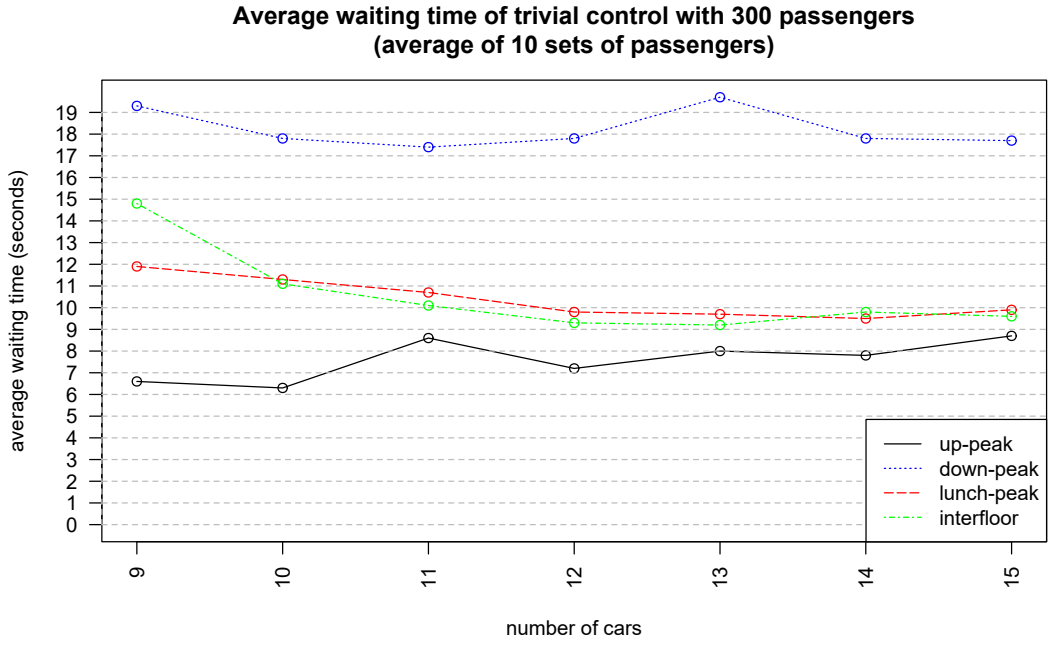

Figure 3.2: AWT in seconds in a ten-storey building with one CMCE with different number of cars. Values on the y-axis are average values of AWT of ten different sets of passengers. The CMCE was evaluated in different types of traffic with 300 passengers.

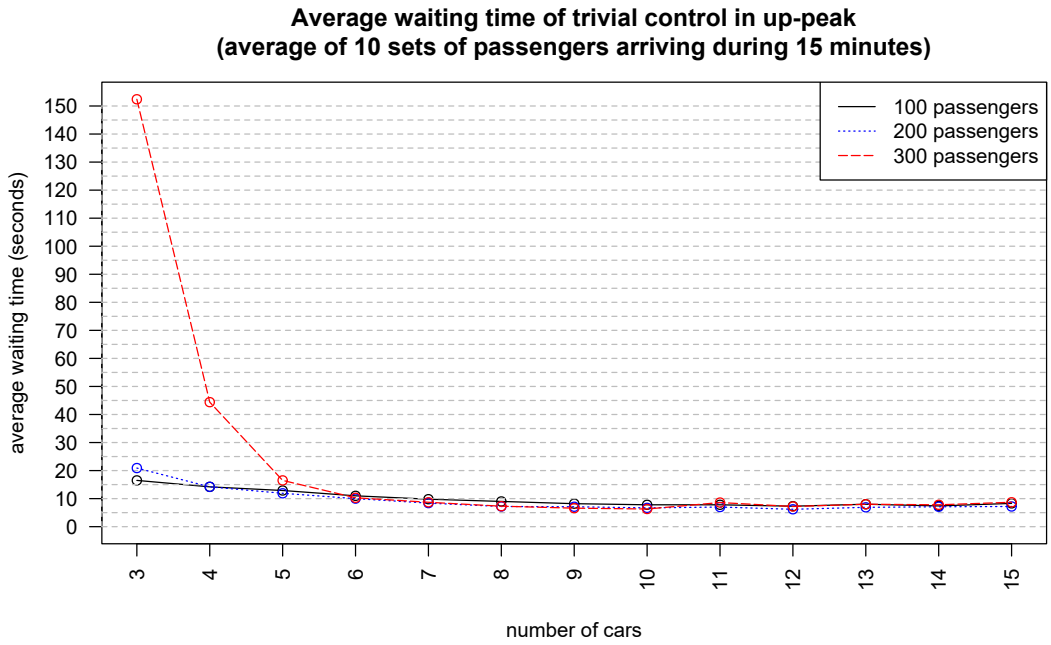

Figure 3.3: AWT in seconds in the up-peak in a ten-storey building with one CMCE with different number of cars and passengers. Values on the y-axis are average values of AWT of ten different sets of passengers.

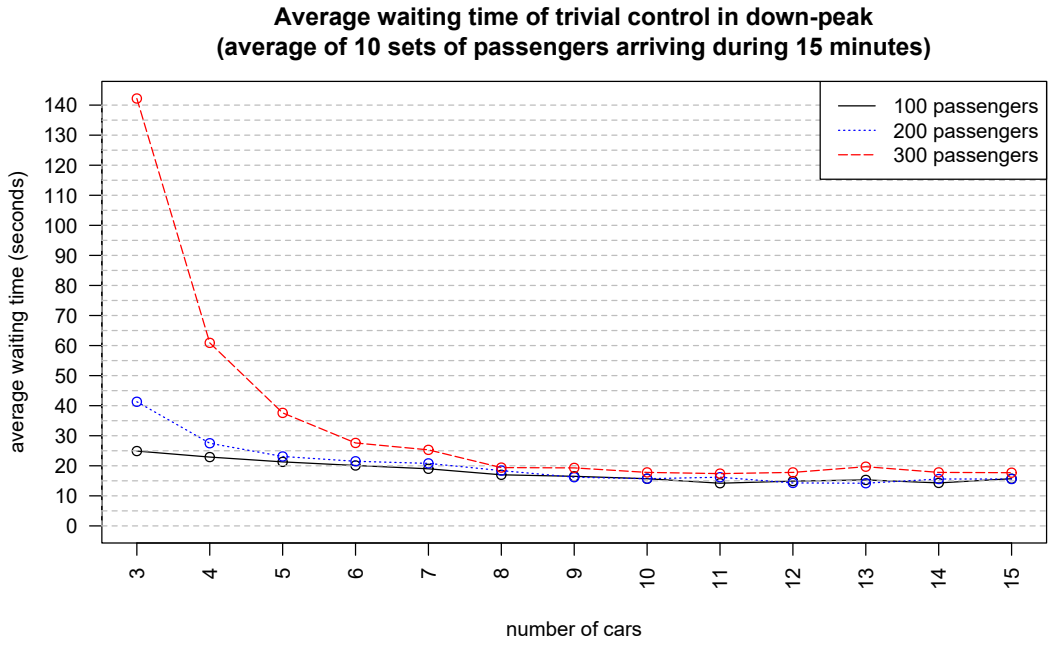

Figure 3.4: AWT in seconds in the down-peak in a ten-storey building with one CMCE with different number of cars and passengers. Values on the y-axis are average values of AWT of ten different sets of passengers.

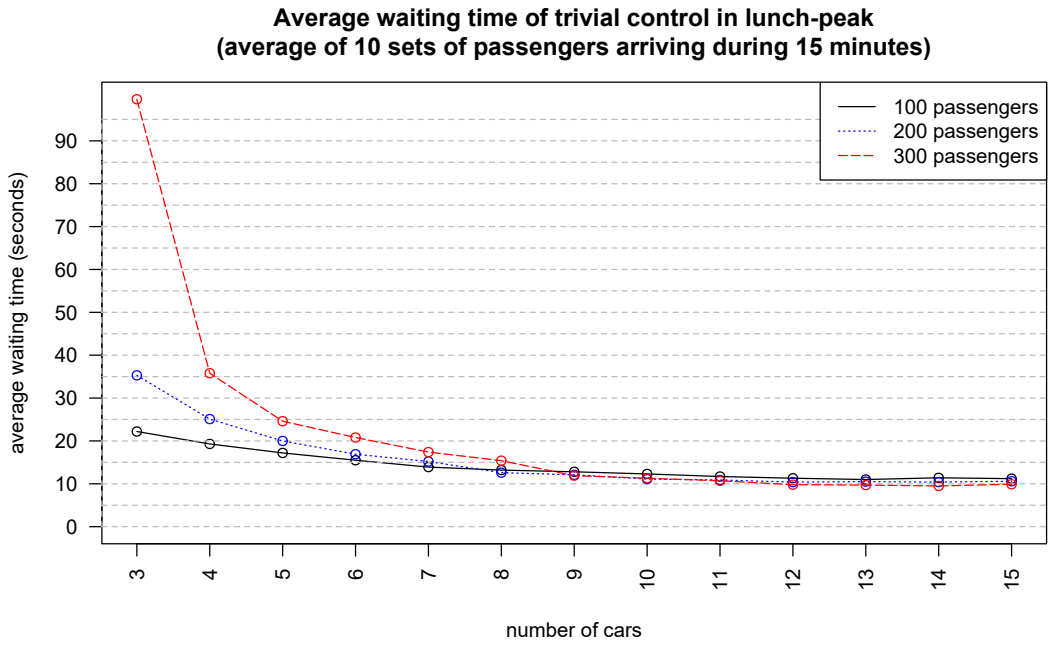

Figure 3.5: AWT in seconds in the lunch-peak in a ten-storey building with one CMCE with different number of cars and passengers. Values on the y-axis are average values of AWT of ten different sets of passengers.

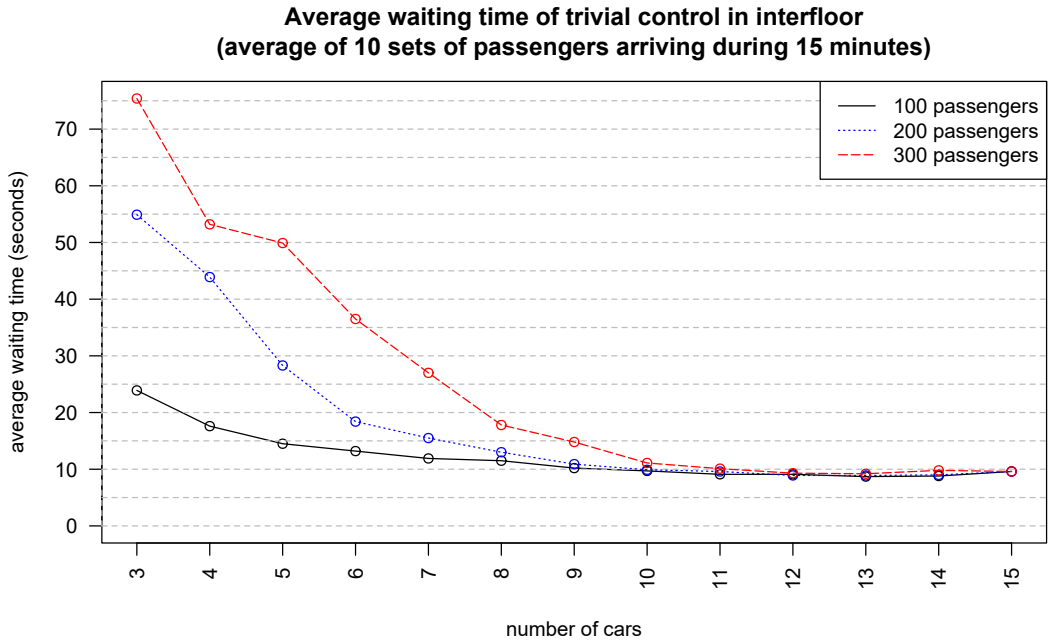

Figure 3.6: AWT in seconds in the interfloor in a ten-storey building with one CMCE with different number of cars and passengers. Values on the y-axis are average values of AWT of ten different sets of passengers.

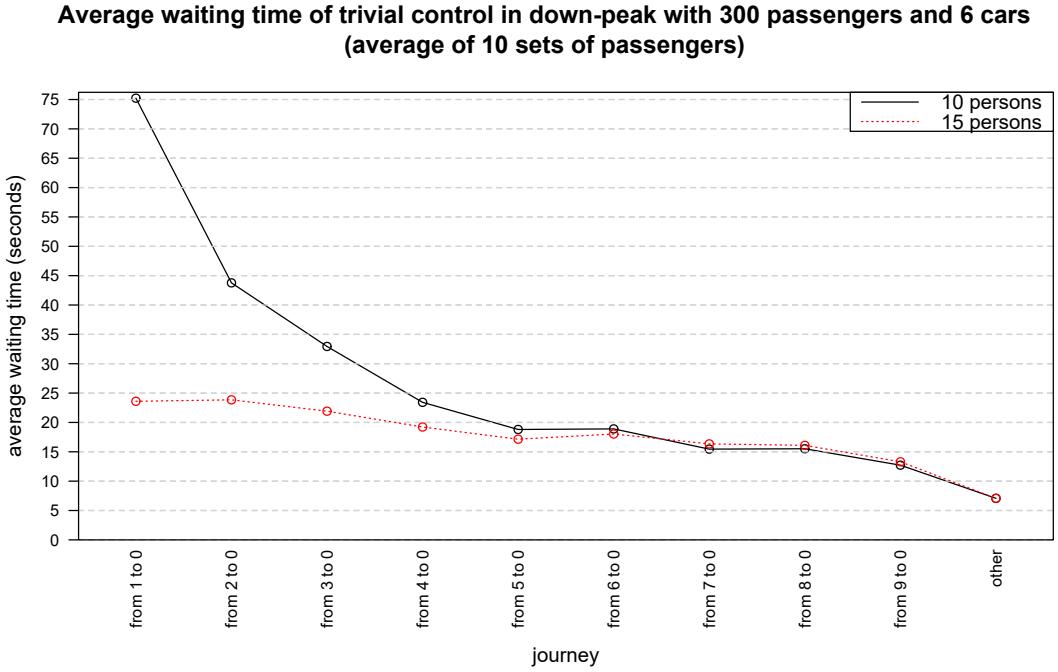

Figure 3.7: AWT in seconds depending on origin floors of passengers in the down-peak in a ten-storey building with one CMCE with 6 cars. Comparison of average AWT with cars with the capacity of 10 and 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

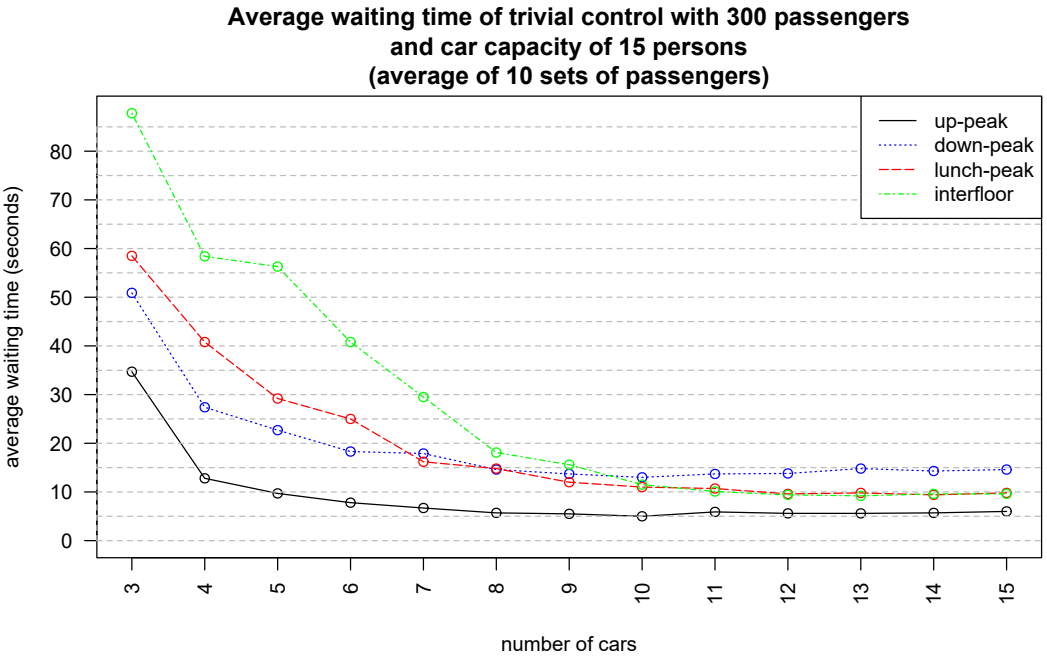

Figure 3.8: AWT in seconds in a ten-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers. The CMCE was evaluated in different types of traffic with 300 passengers.

In the down-peak traffic, the average AWT is lower with all numbers of cars and all numbers of passengers (see Figure 3.11). In the lunch-peak and interfloor traffic, the average AWT is similar with higher number of cars with all numbers of passengers to the average AWT with cars of capacity 10 (see Figure 3.12 and Figure 3.13). Similarly to the up-peak, in the down-peak and lunch-peak traffic, the average AWT is lower in more dense traffic with higher numbers of cars.

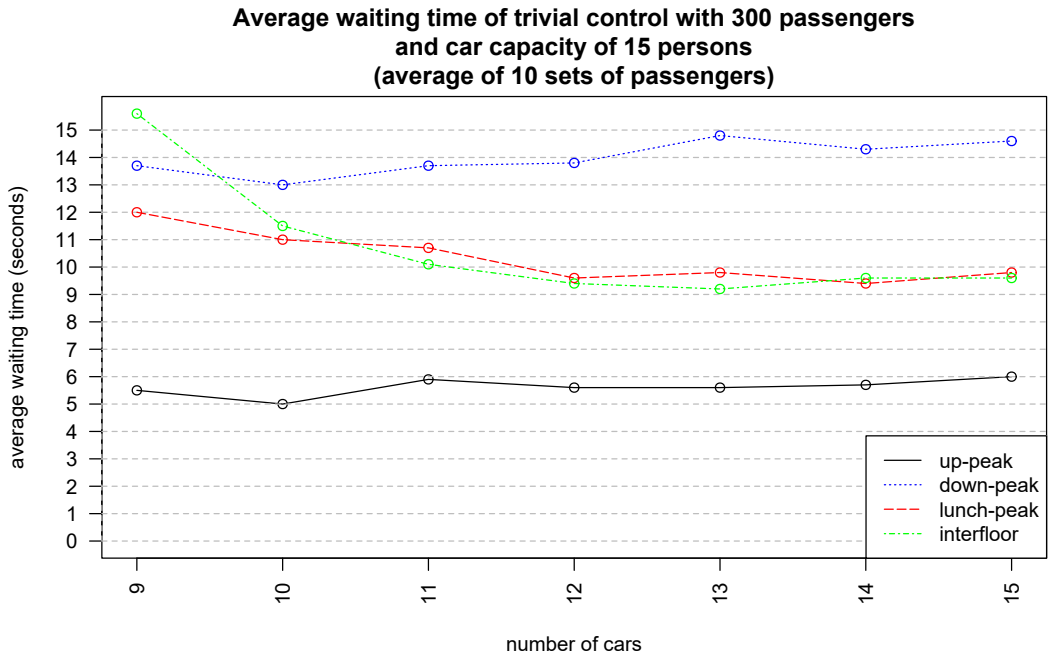

Figure 3.9: AWT in seconds in a ten-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers. The CMCE was evaluated in different types of traffic with 300 passengers.

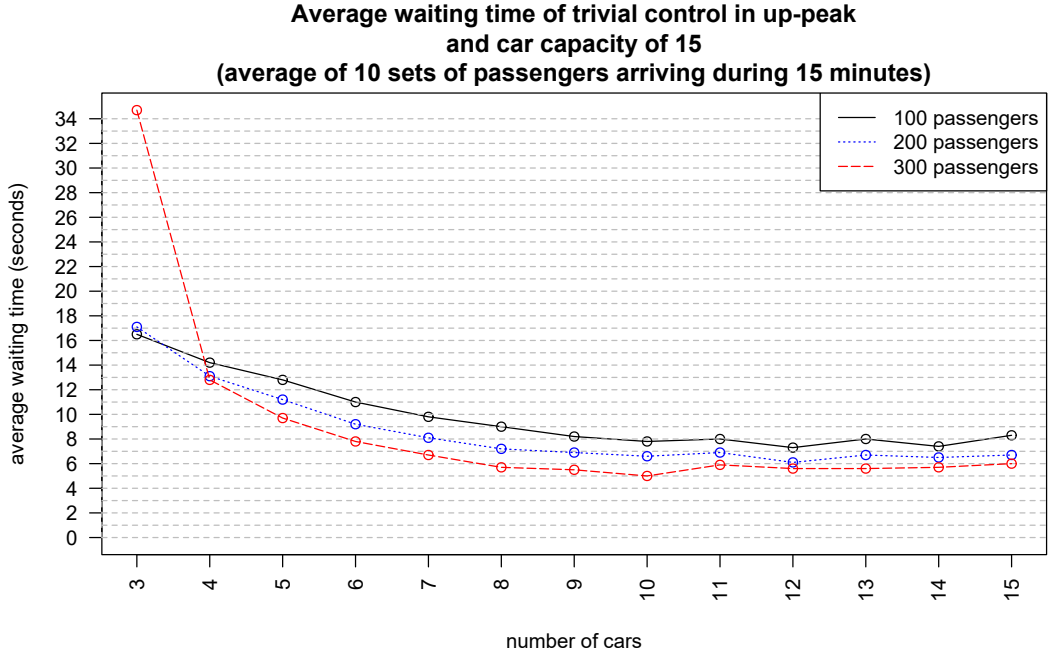

Figure 3.10: AWT in seconds in the up-peak in a ten-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

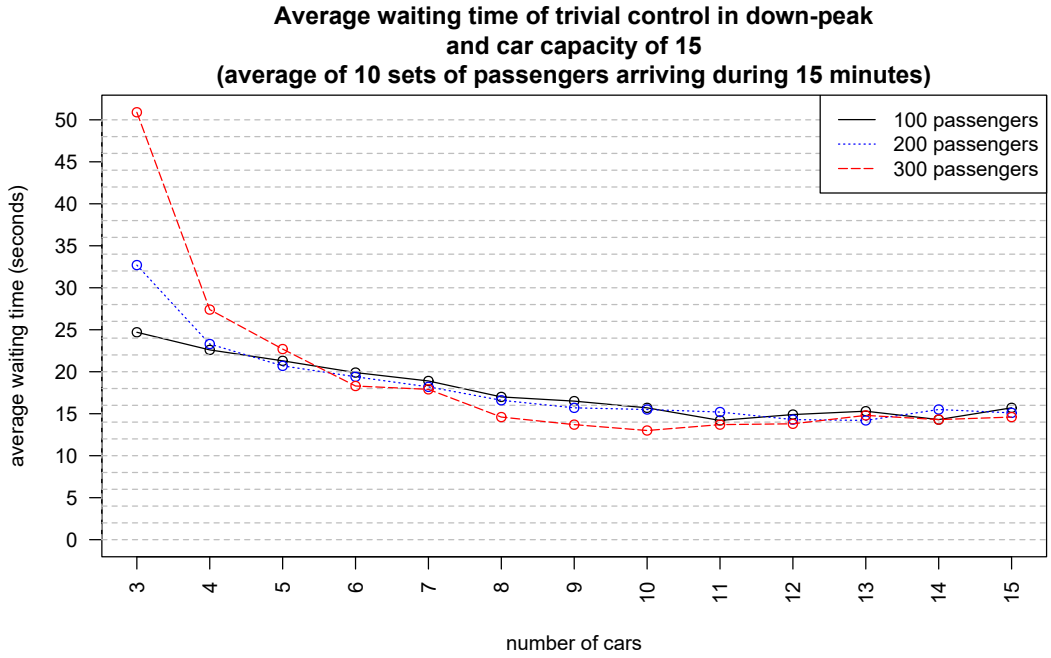

Figure 3.11: AWT in seconds in the down-peak in a ten-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

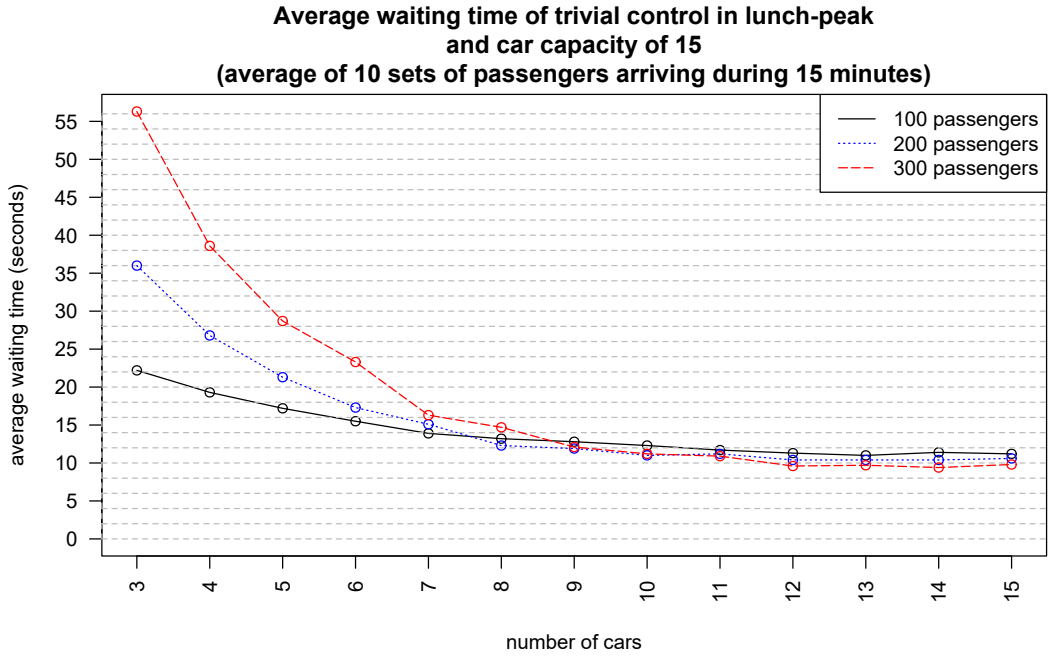

Figure 3.12: AWT in seconds in the lunch-peak in a ten-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

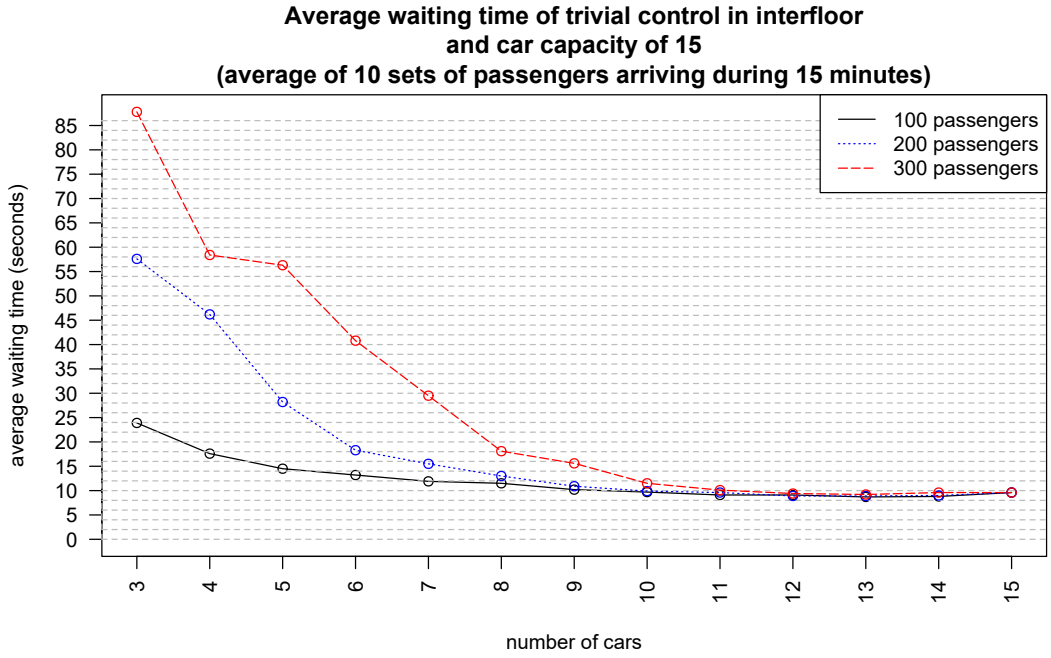

Figure 3.13: AWT in seconds in the interfloor in a ten-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

3.1.4 Evaluation of efficiency of the trivial control and comparison with results found by the genetic algorithm

This subsection contains comparison of AWT in the trivial control and AWT in the control found by the genetic algorithm for one set of passengers in the up-peak, down-peak, lunch-peak and interfloor traffic.

3.1.4.1 Up-peak with 4 cars

For one set of 300 passengers in the up-peak traffic with 4 cars, the genetic algorithm found after 5000 iterations an elevator control with AWT of 21 seconds, while AWT in the trivial control was 44 seconds (see Table 3.1). The simulation of the trivial control finished after almost 18 minutes, while the simulation of the optimized control finished after more than 16 minutes. In the trivial control, waiting times of passengers arriving after the 8^{th} minute are very high (see Figure 3.14). In the optimized control, waiting times of these passengers are significantly lower.

3.1.4.2 Down-peak with 5 cars

For a set of 300 passengers in the down-peak traffic with 5 cars, the genetic algorithm could find an elevator control with AWT of 26 seconds, while AWT in the trivial control was 43 seconds (see Table 3.2). The simulation of the trivial control finished after almost 17 minutes, while the simulation of the optimized control finished after about 16 minutes. The optimized solution lowered high AWT occurring in the trivial control for passengers arriving between the 9^{th} and the 14^{th} minute.

3.1.4.3 Down-peak with 6 cars

With 6 cars, the genetic algorithm found for the set of passengers described in 3.1.4.2 an elevator control with AWT of 23 seconds, while AWT in the trivial control was 32 seconds. Both simulations finished after more than 16 seconds. In the trivial control, passengers arriving between the 10^{th} and the 14^{th} minute (see Figure 3.16). In the optimized control, waiting times of these passengers were lower.

3.1.4.4 Lunch-peak with 4 cars

For a set of 300 passengers in the lunch-peak traffic with 4 cars, the genetic algorithm found after 7000 iterations a solution with AWT of 32 seconds, while AWT in the trivial control was 37 seconds (see Table 3.3). Both simulations

number of cars	4	5	6	7	8
average waiting time	44	21	9	9	6

Table 3.1: AWT in the trivial control in seconds for one set of passengers arriving during 15 minutes in the up-peak traffic in a ten-storey building depending on number of cars with a capacity of 10 persons in one CMCE elevator.

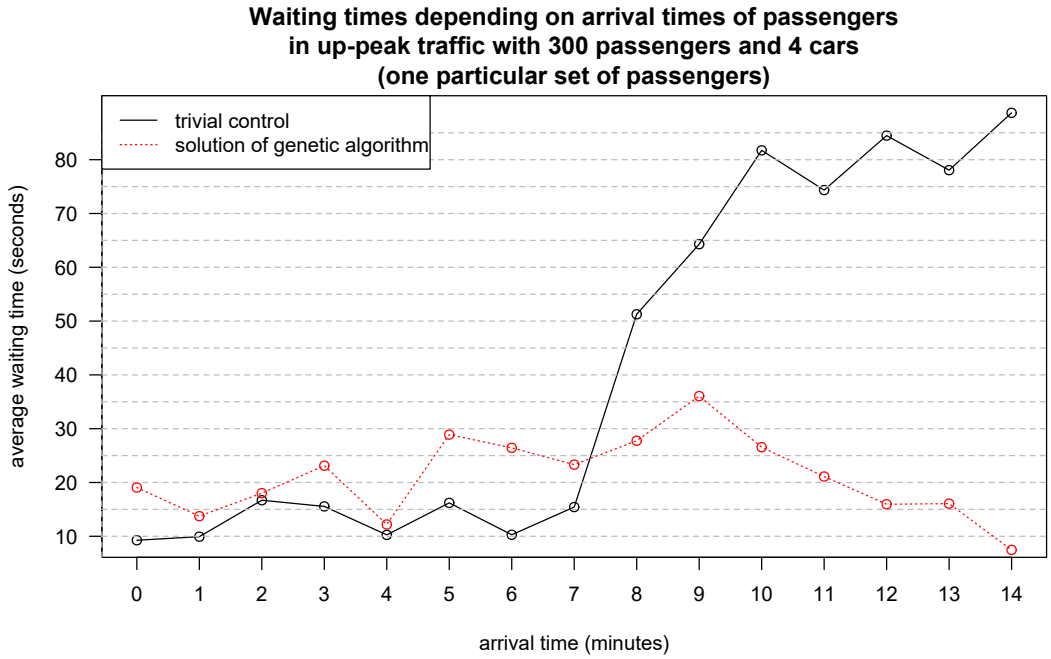

Figure 3.14: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the up-peak traffic in a ten-storey building with one CMCE with 4 cars with a capacity of 10 persons.

number of cars	4	5	6	7	8
average waiting time	56	43	32	29	24

Table 3.2: AWT in the trivial control in seconds for one set of passengers arriving during 15 minutes in the down-peak traffic in a ten-storey building depending on number of cars with a capacity of 10 persons in one CMCE elevator.

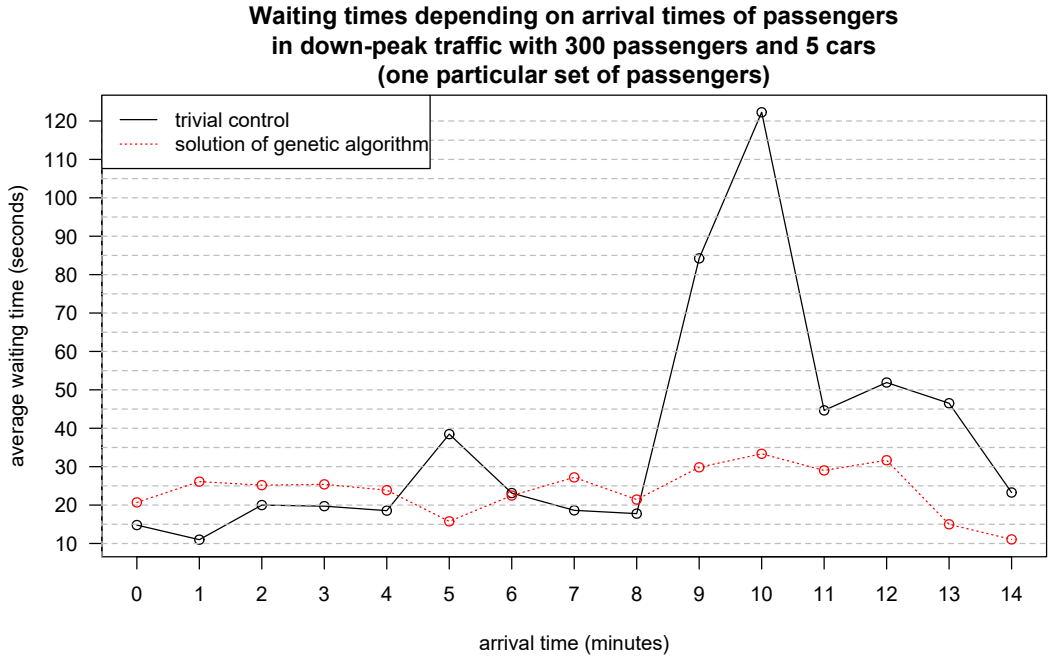

Figure 3.15: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the down-peak traffic in a ten-storey building with one CMCE with 5 cars with a capacity of 10 persons.

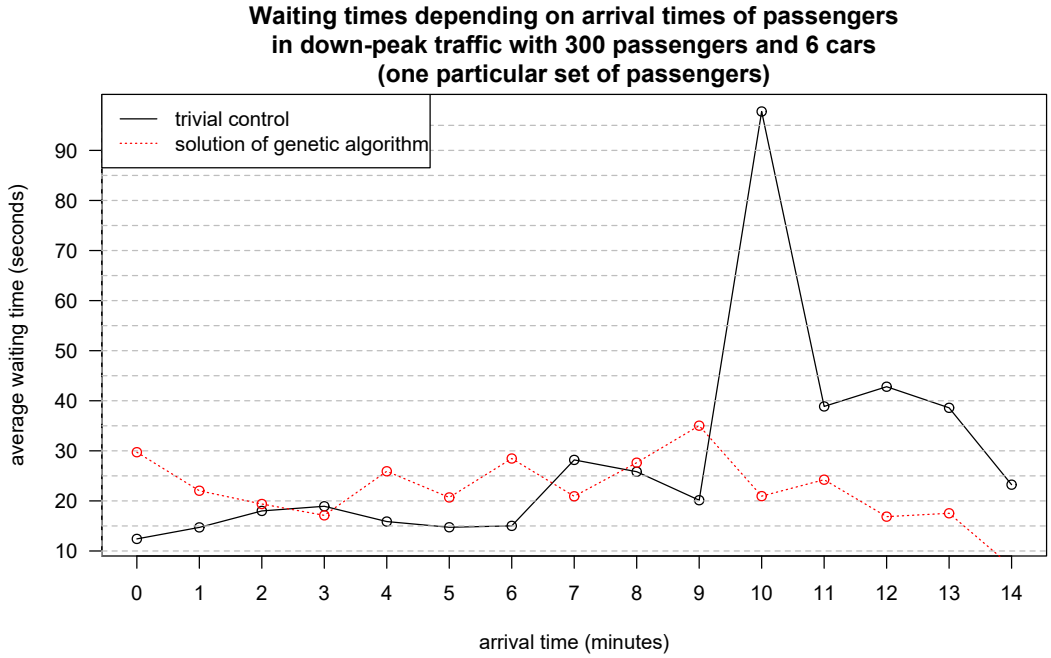

Figure 3.16: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the down-peak traffic in a ten-storey building with one CMCE with 6 cars with a capacity of 10 persons.

number of cars	4	5	6	7	8
average waiting time	37	31	24	22	16

Table 3.3: AWT in the trivial control in seconds for one set of passengers arriving during 15 minutes in the lunch-peak traffic in a ten-storey building depending on number of cars with a capacity of 10 persons in one CMCE elevator.

number of cars	4	5	6	7	8
average waiting time	55	51	35	26	17

Table 3.4: AWT in the trivial control in seconds for one set of passengers arriving during 15 minutes in the interfloor traffic in a ten-storey building depending on number of cars with a capacity of 10 persons in one CMCE elevator.

finished after about 16 minutes. In the trivial control, the highest waiting times occurred for passengers arriving between the 10^{th} and the 11^{th} minute (see Figure 3.17).

3.1.4.5 Lunch-peak with 5 cars

For the passengers described in 3.1.4.4 with 5 cars, the genetic algorithm found a solution with AWT of 26 seconds, while AWT in the trivial control was 31 seconds. Both simulations finished after about 16 seconds. In the trivial control, the highest waiting times occurred for passengers arriving between the 6^{th} and the 7^{th} minute (see Figure 3.18).

3.1.4.6 Interfloor with 5 cars

For a set of 300 passengers in the interfloor traffic, AWT in the trivial control with 5 cars (with a capacity of 10 persons) was 51 seconds (see Table 3.4). The genetic algorithm could found after 2000 iterations a different assignment of passengers to cars, whose AWT is 34 seconds. This shows, that the trivial control is not optimal, since this solution is by about 40 % better than the trivial control. In this optimized solution, the last passenger left the car after about 16 minutes from the beginning of the simulation. In the trivial control simulation, the last passenger got out of the car after almost 18 minutes. In the trivial control, a lot of the highest waiting times occur for passengers arriving in the second half of the time period (see Figure 3.19), where the elevator is not able to serve more incoming passengers. However, in first three minutes, AWT are significantly lower in the trivial control.

3.1.4.7 Interfloor with 6 cars

With 6 cars, the genetic algorithm could find for the set of passengers described in 3.1.4.6 an elevator control with AWT of 28 seconds, while AWT in the trivial control was 35 seconds. In the trivial control, AWT is very high especially for passengers arriving between the 7^{th} and the 11^{th} minute (see Figure 3.20). The optimized control eliminates these high AWT. The simulation of the trivial control finished after more than 17 minutes, while the simulation of the genetic algorithm solution finished after about 16 minutes.

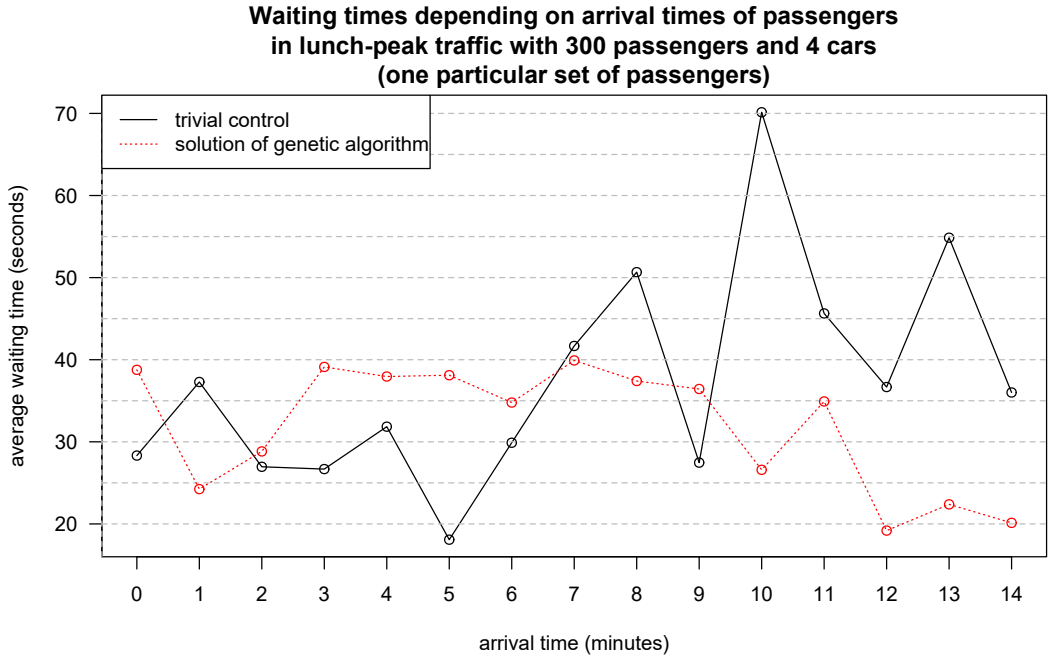

Figure 3.17: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the lunch-peak traffic in a ten-storey building with one CMCE with 4 cars with a capacity of 10 persons.

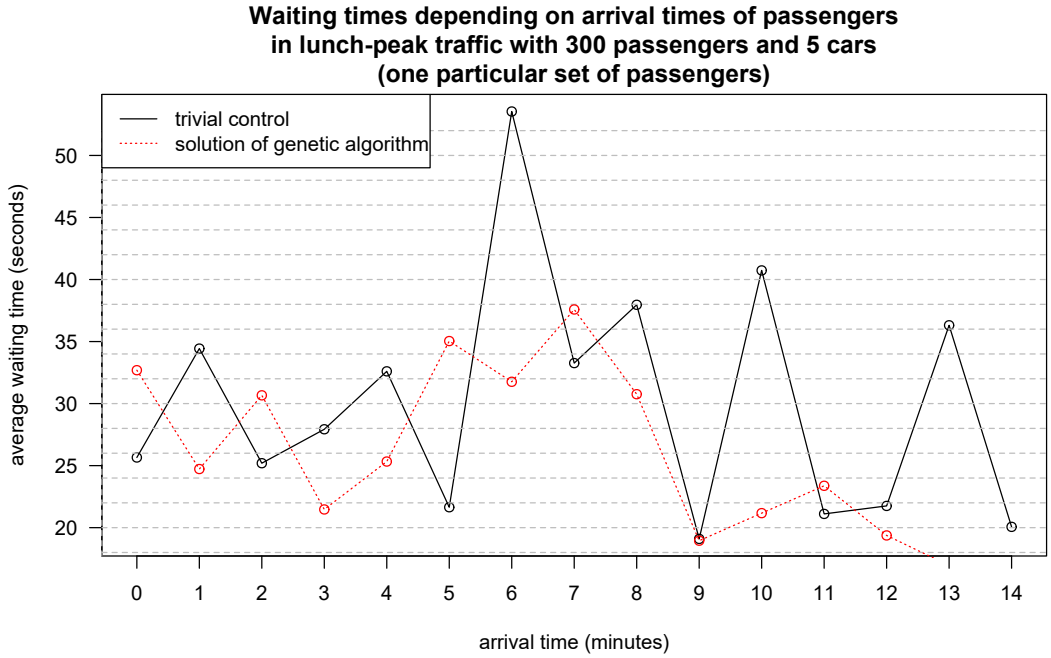

Figure 3.18: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the lunch-peak traffic in a ten-storey building with one CMCE with 5 cars with a capacity of 10 persons.

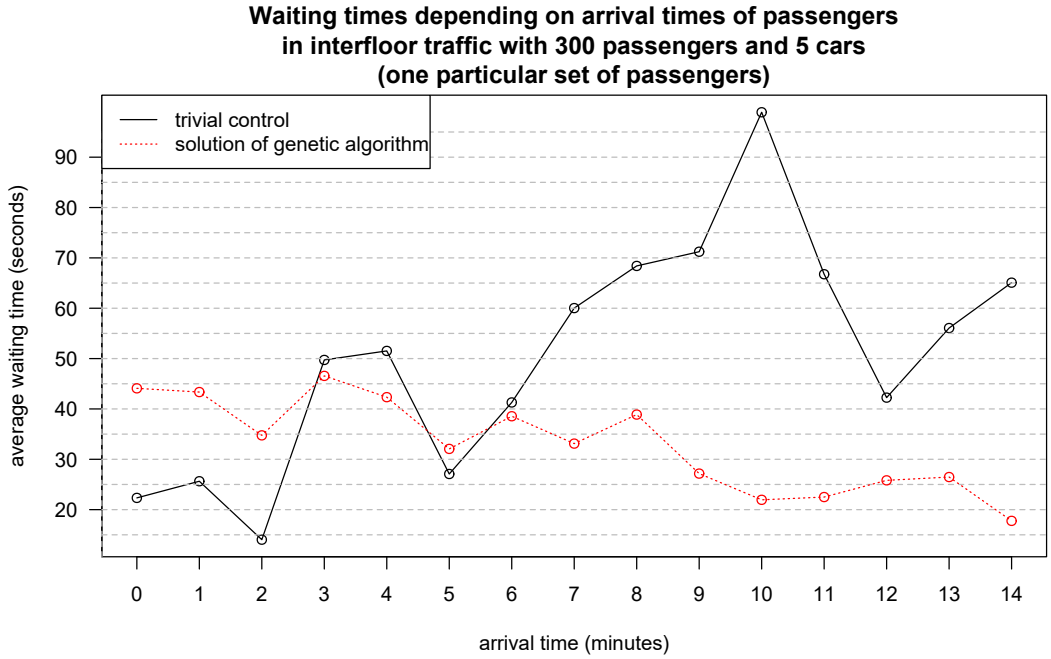

Figure 3.19: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the interfloor traffic in a ten-storey building with one CMCE with 5 cars with a capacity of 10 persons.

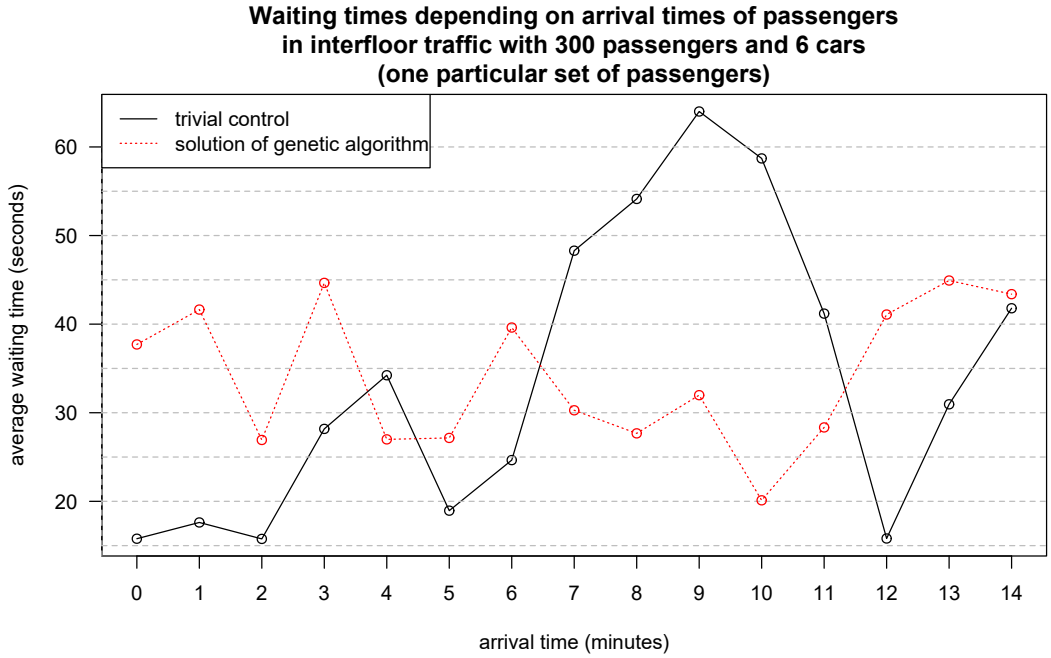

Figure 3.20: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the interfloor traffic in a ten-storey building with one CMCE with 6 cars with a capacity of 10 persons.

3.2 Simulations in a twenty-storey building

3.2.1 AWT in the trivial control with different number of cars and 300 passengers

Similarly to the previous building (see section 3.1), the average AWT with 300 passengers is generally the highest in the down-peak traffic (see Figure 3.21). The ideal number of cars is around 26 (see Figure 3.22), where the average AWT are about 10 seconds in the up-peak and interfloor, 12 in the lunch-peak and 22 in the down-peak. The average AWT generally decreases with increasing number of cars and slightly increases again with very high number of cars.

3.2.2 AWT for different number of passengers

Similarly to the previous building, the average AWT is higher for higher number of passengers with less cars and similar for different numbers of passengers with more cars (see Figure 3.23, Figure 3.24, Figure 3.25 and Figure 3.26).

3.2.3 Improving AWT by increasing car capacity

Similarly to the previous building, high waiting times can be lowered by increasing the car capacity. For instance in the down-peak with 300 passengers and 16 cars, where we can see that high AWT occur for passengers with low origin floors (see Figure 3.27), increasing the car capacity to 15 persons significantly improves waiting times. And also improvement of AWT is noticeable mainly with low number of cars (see Figure 3.28 and Figure 3.29). Increasing the car capacity is useful especially in the up-peak and down-peak. To decrease the average AWT under 22 seconds, 16 cars with the capacity of 15 persons would be enough.

In the up-peak and lunch-peak and sometimes in the down-peak, waiting times are higher with lower number of passengers (see Figure 3.30, Figure 3.32 and Figure 3.31). In contrast with other types of traffic, in the interfloor the average AWT is higher with a higher car capacity with low number of cars and at least 200 passengers (see Figure 3.33). It seems that in the interfloor traffic with small number of cars, it was more convenient not to load many passengers to one car. Cars then arrive later to distant passengers, since they must stop more often to load and unload passengers and therefore they are moving slower around the elevator.

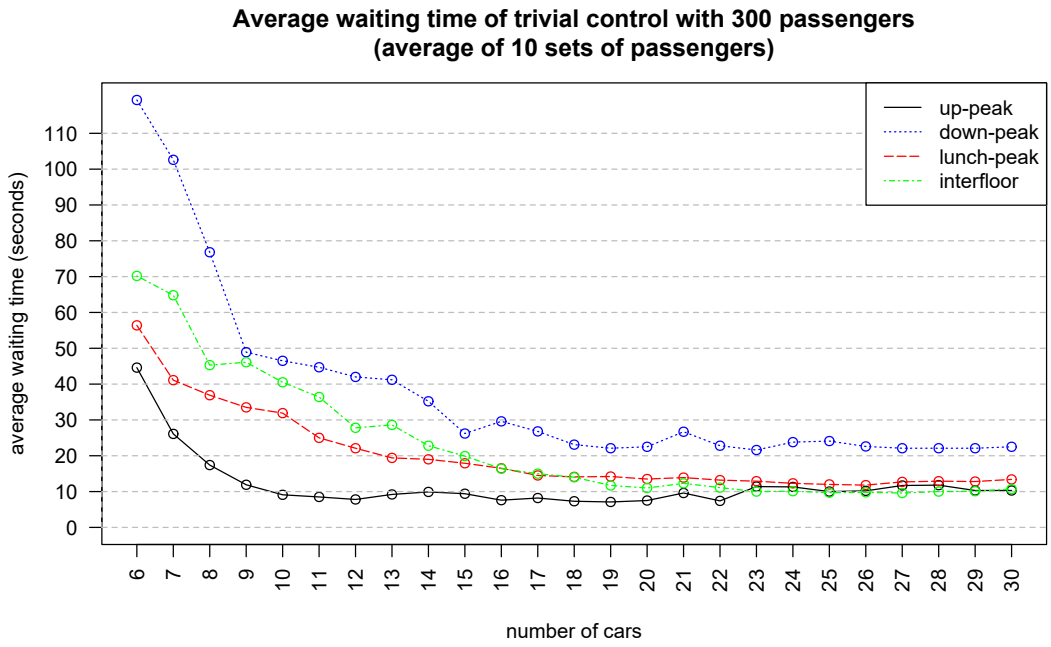

Figure 3.21: AWT in seconds in a twenty-storey building with one CMCE with different number of cars. Values on the y-axis are average values of AWT of ten different sets of passengers. The CMCE was evaluated in different types of traffic with 300 passengers.

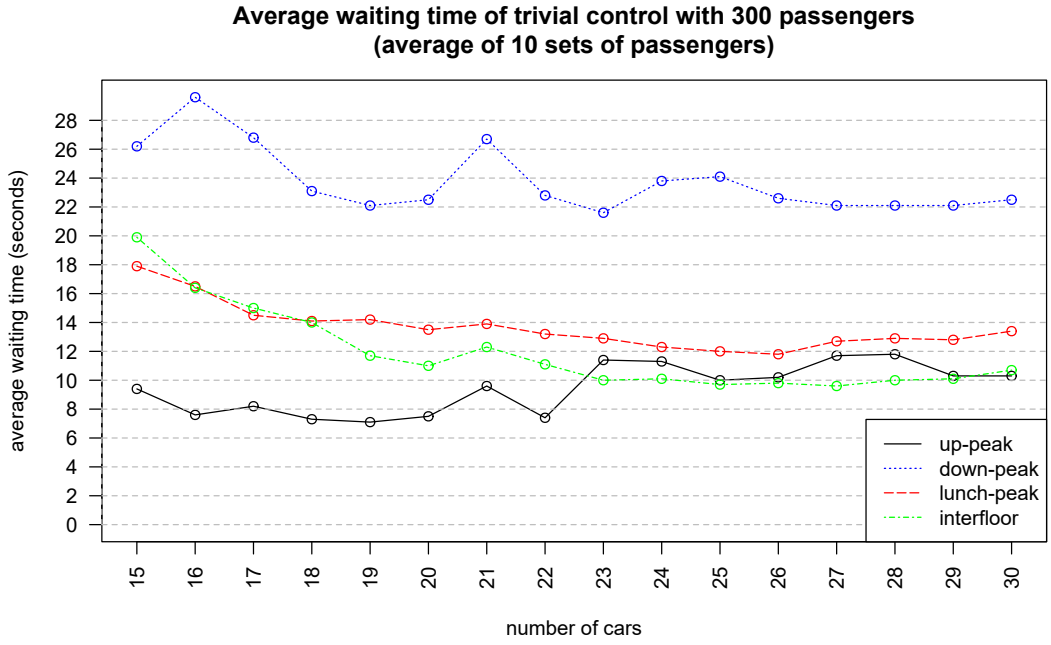

Figure 3.22: AWT in seconds in a twenty-storey building with one CMCE with different number of cars. Values on the y-axis are average values of AWT of ten different sets of passengers. The CMCE was evaluated in different types of traffic with 300 passengers.

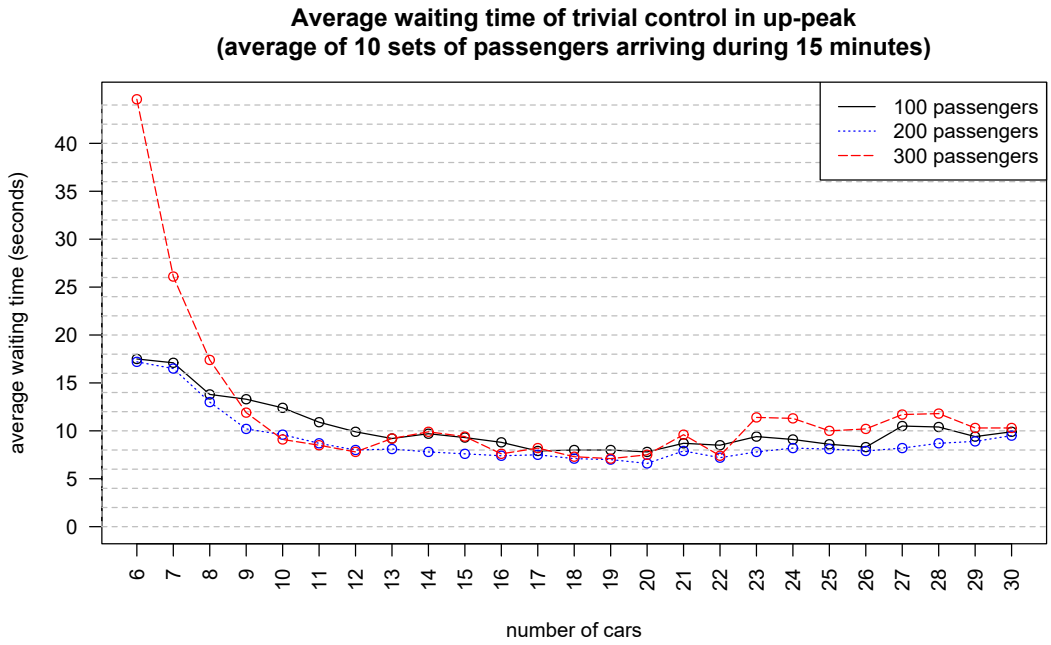

Figure 3.23: AWT in seconds in the up-peak in a twenty-storey building with one CMCE with different number of cars and passengers. Values on the y-axis are average values of AWT of ten different sets of passengers.

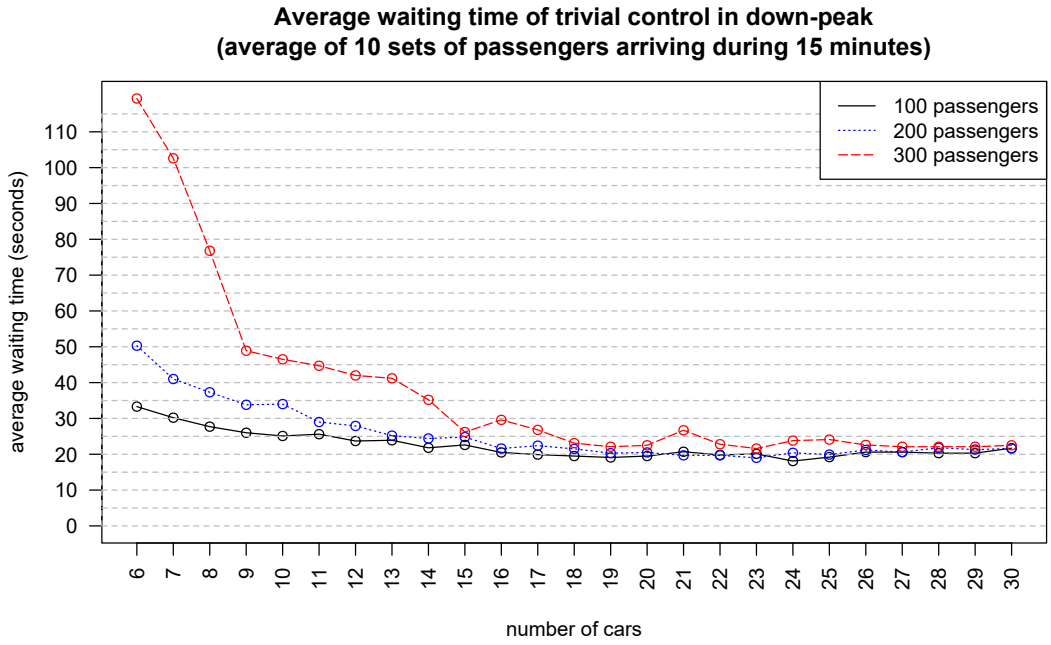

Figure 3.24: AWT in seconds in the down-peak in a twenty-storey building with one CMCE with different number of cars and passengers. Values on the y-axis are average values of AWT of ten different sets of passengers.

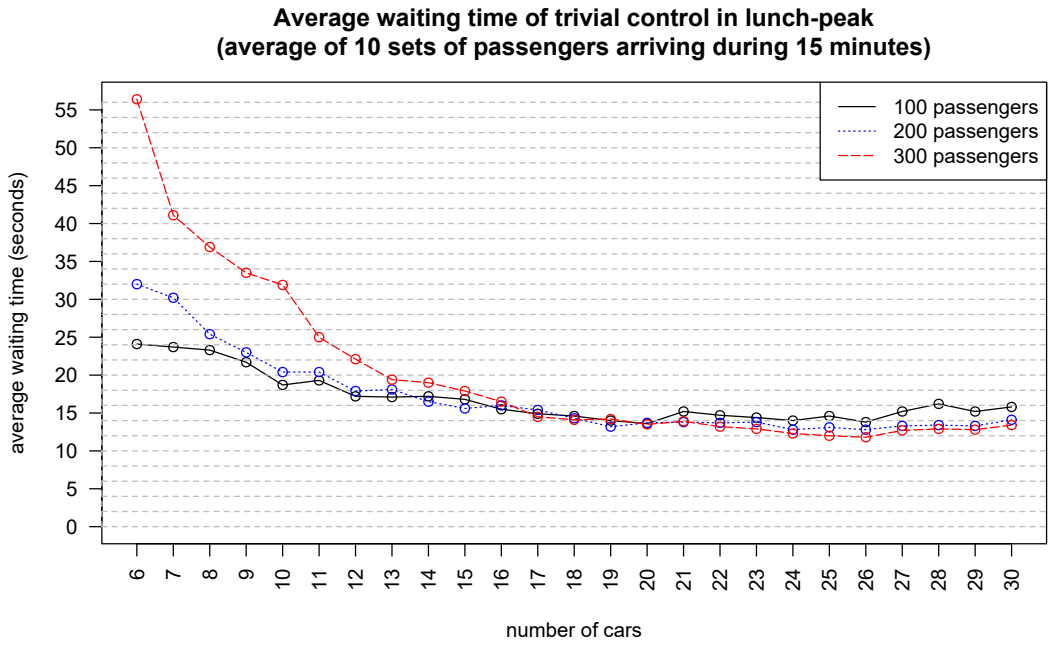

Figure 3.25: AWT in seconds in the lunch-peak in a twenty-storey building with one CMCE with different number of cars and passengers. Values on the y-axis are average values of AWT of ten different sets of passengers.

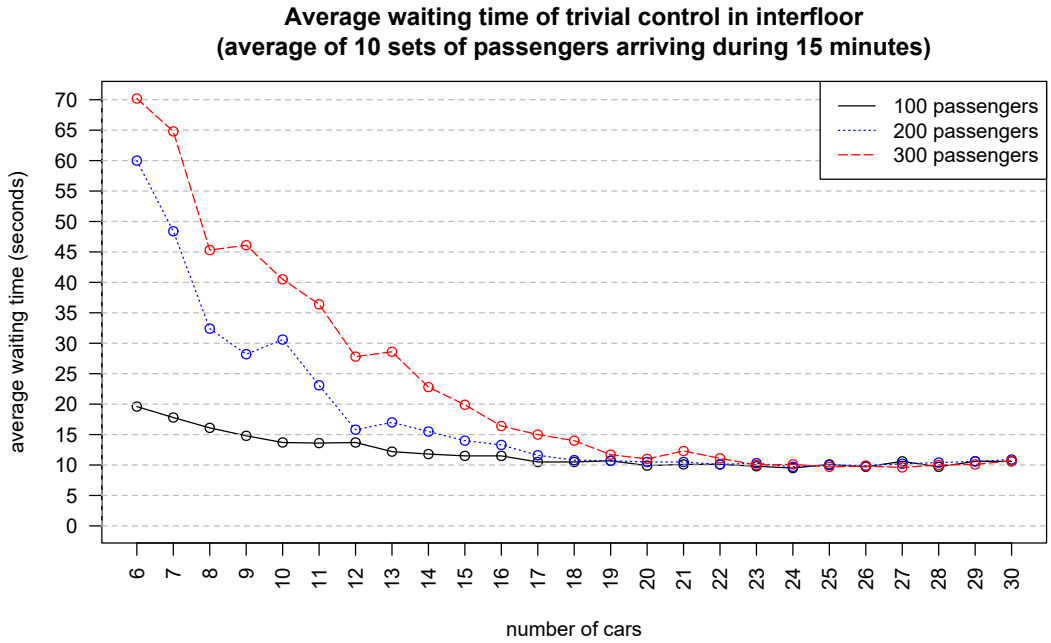

Figure 3.26: AWT in seconds in the interfloor in a twenty-storey building with one CMCE with different number of cars and passengers. Values on the y-axis are average values of AWT of ten different sets of passengers.

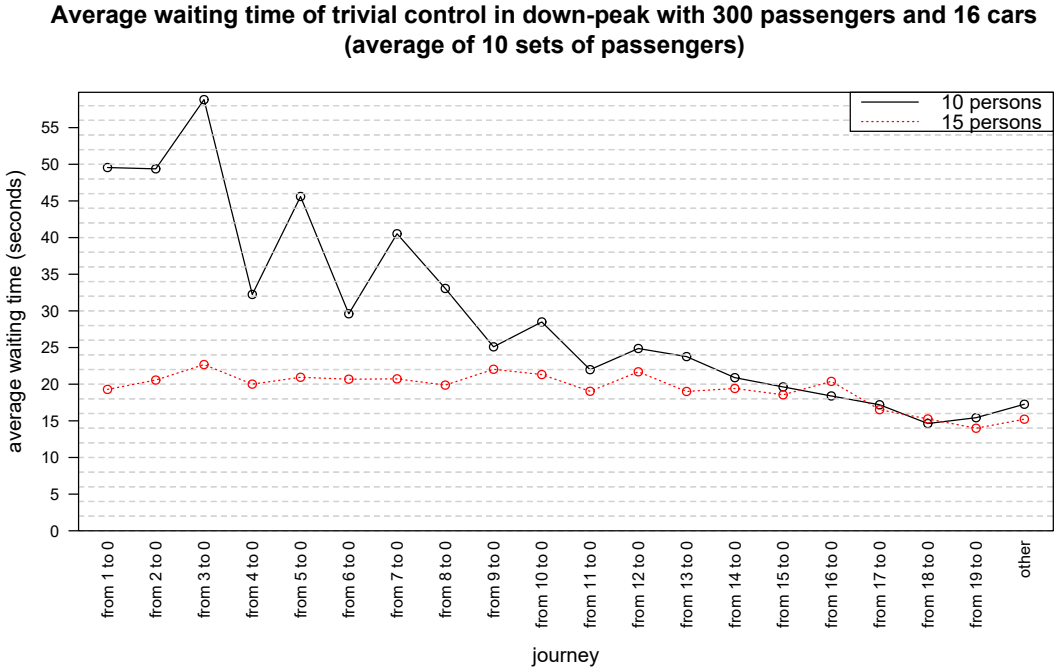

Figure 3.27: AWT in seconds depending on origin floors of passengers in the down-peak in a twenty-storey building with one CMCE with 16 cars. Comparison of average AWT with cars with the capacity of 10 and 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

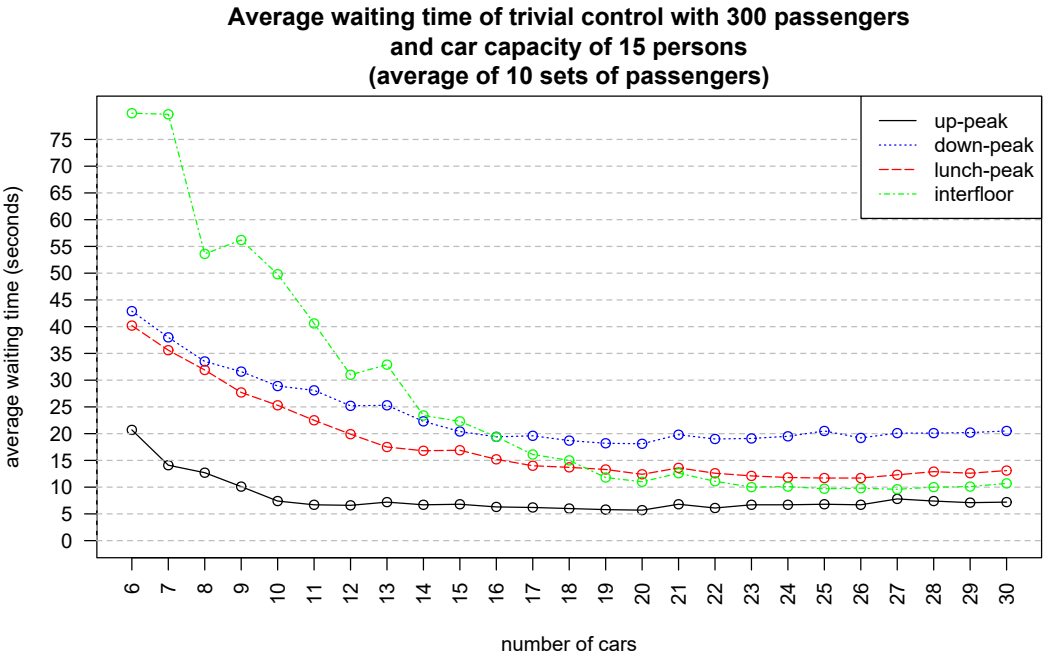

Figure 3.28: AWT in seconds in a twenty-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers. The CMCE was evaluated in different types of traffic with 300 passengers.

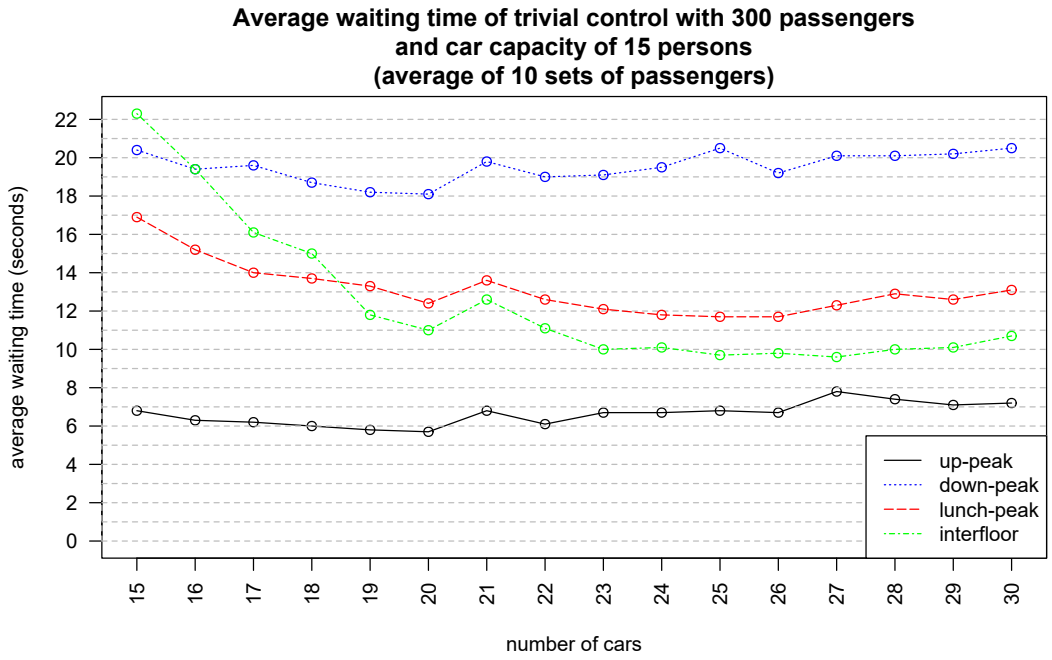

Figure 3.29: AWT in seconds in a twenty-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers. The CMCE was evaluated in different types of traffic with 300 passengers.

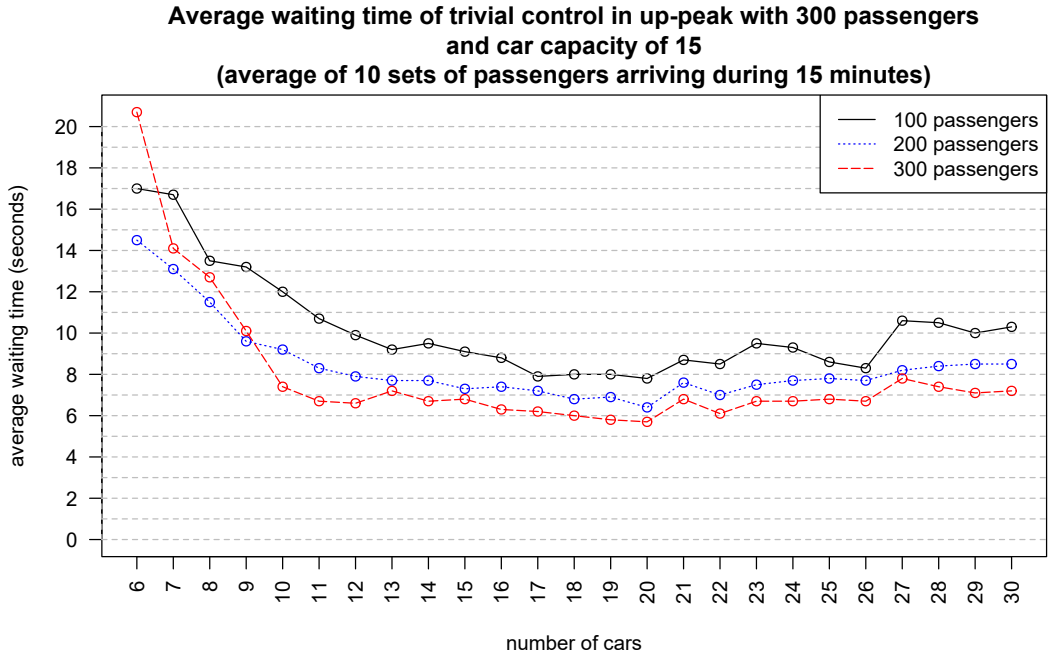

Figure 3.30: AWT in seconds in the up-peak in a twenty-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

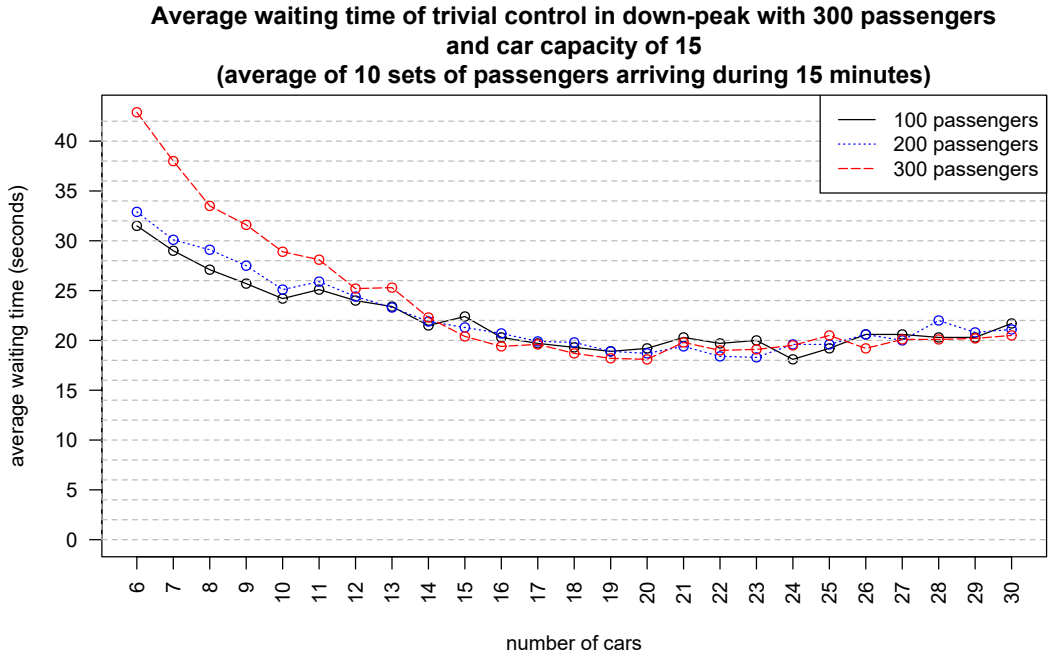

Figure 3.31: AWT in seconds in the down-peak in a twenty-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

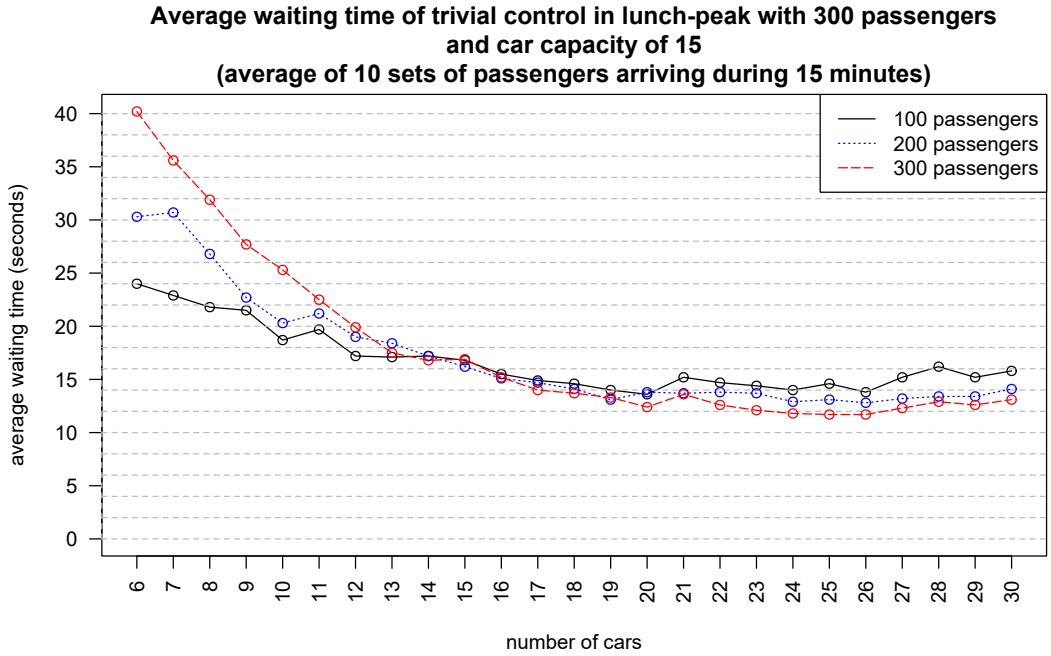

Figure 3.32: AWT in seconds in the lunch-peak in a twenty-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

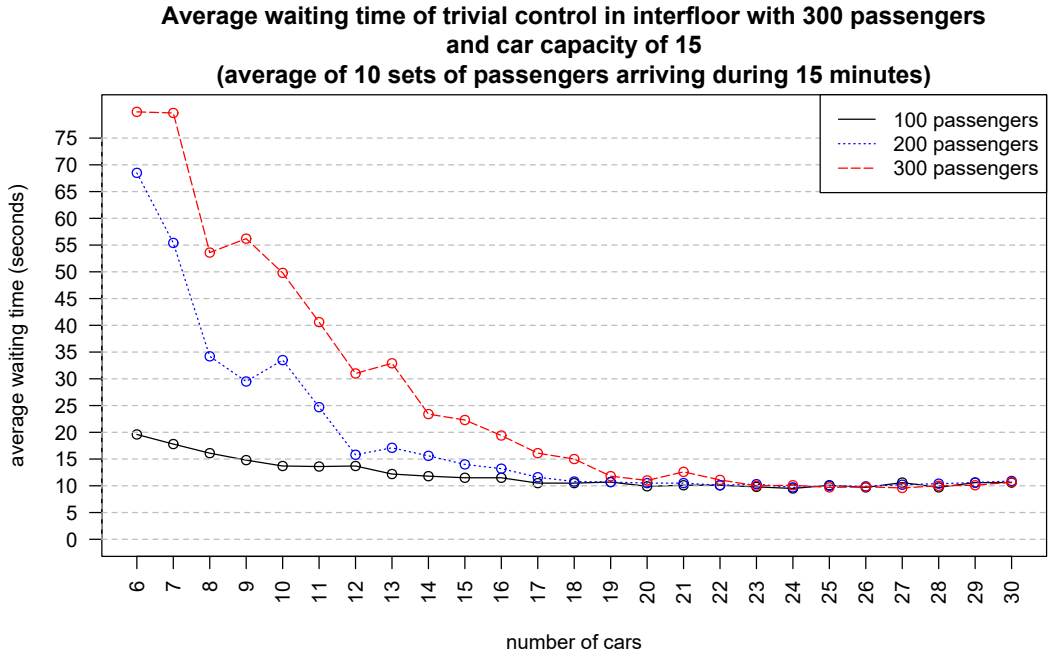

Figure 3.33: AWT in seconds in the interfloor in a twenty-storey building with one CMCE with different number of cars with capacity of 15. Values on the y-axis are average values of AWT of ten different sets of passengers.

3.2.4 Evaluation of efficiency of the trivial control and comparison with results found by the genetic algorithm

This subsection contains comparison of AWT in the trivial control and AWT in the control found by the genetic algorithm for one set of passengers in the up-peak, down-peak, lunch-peak and interfloor traffic.

3.2.4.1 Up-peak with 6 cars

For a set of 300 passengers in the up-peak traffic with 6 cars in one elevator, the genetic algorithm found a solution with AWT of 23 seconds, while AWT of the trivial control was 49 seconds (see Table 3.5), which is more than double. It means that if the elevator control was more efficient, six cars would be sufficient to serve this set of passengers. Efficient trivial control would require more cars. The simulation of the trivial control finished after almost 19 minutes, while the simulation of the solution found by the genetic algorithm finished after about 17 minutes. In the trivial control, AWT are very high for passengers who arrived later than in the first 4 minutes (see Figure 3.34). In the solution of the genetic algorithm, AWT does not seem to depend on arrival times.

3.2.4.2 Down-peak with 6 cars

For a set of 300 passengers in the down-peak traffic with 6 cars, the genetic algorithm found a solutions with AWT of 40 seconds, while AWT in the trivial control was 83 seconds (see Table 3.6). In the trivial control, AWT are very high for passengers arriving after 5 or more minutes (see Figure 3.35). In the optimized control, AWT are significantly lower. AWT of the solution of the genetic algorithm is by about 52 % lower than AWT in the trivial control. The simulation of the trivial control finished after more than 18 minutes, while the simulation of the control found by the genetic algorithm finished after more than 16 minutes.

3.2.4.3 Down-peak with 7 cars

With 7 cars in one elevator, the genetic algorithm found for the set of passengers described in3.2.4.2 a solution with AWT of 34 seconds, while AWT in the trivial control was 61 seconds. Both simulations finished after about 17 minutes. The optimized solution eliminated AWT exceeding a minute, which occurred in the trivial control for passengers arriving between the 4^{th} and the 9^{th} minute (see Figure 3.36).

number of cars	6	7	8	9	10	11	12
average waiting time	49	28	15	15	12	10	9

Table 3.5: AWT in the trivial control in seconds for one set of passengers arriving during 15 minutes in the up-peak traffic in a twenty-storey building depending on number of cars with a capacity of 10 persons in one CMCE elevator.

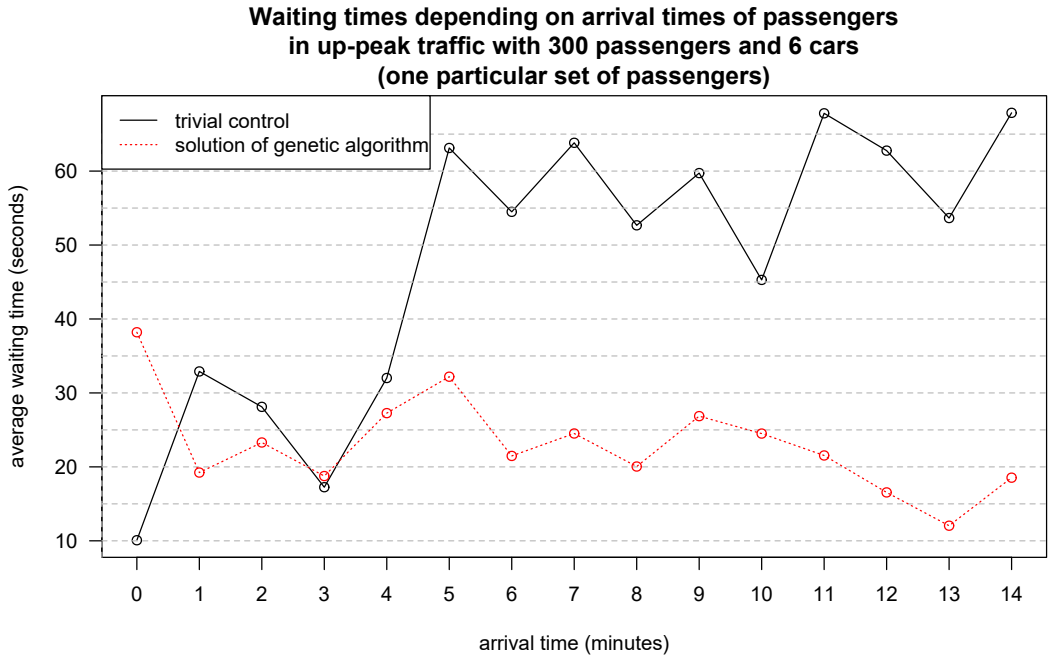

Figure 3.34: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the up-peak traffic in a twenty-storey building with one CMCE with 6 cars with a capacity of 10 persons.

number of cars	6	7	8	9	10	11	12
average waiting time	83	61	50	59	37	35	26

Table 3.6: AWT in the trivial control in seconds for one set of passengers arriving during 15 minutes in the down-peak traffic in a twenty-storey building depending on number of cars with a capacity of 10 persons in one CMCE elevator.

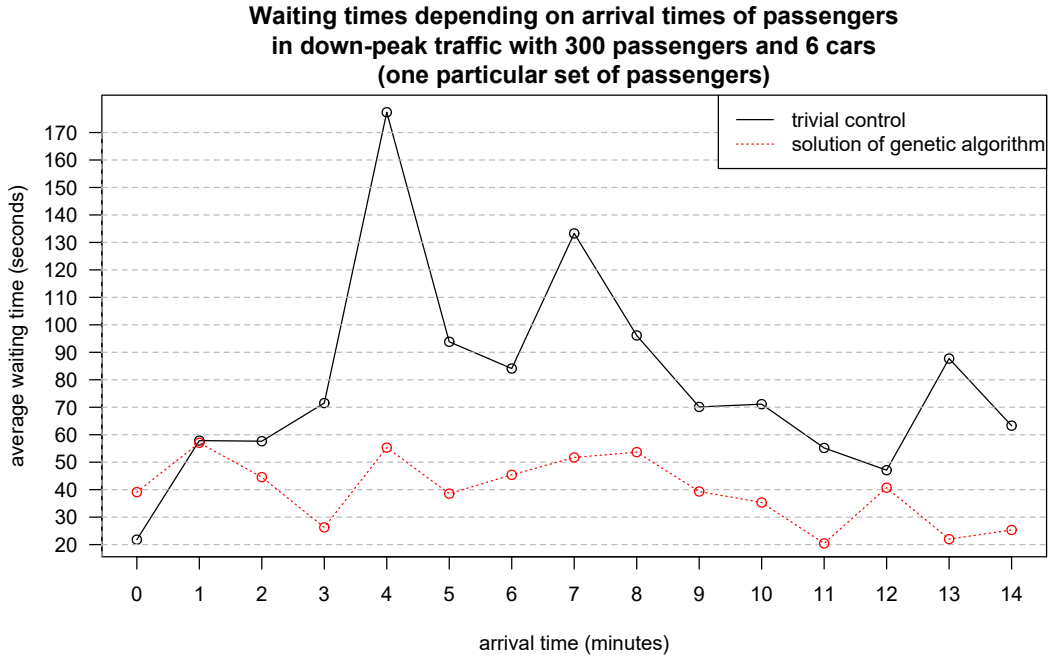

Figure 3.35: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the down-peak traffic in a twenty-storey building with one CMCE with 6 cars with a capacity of 10 persons.

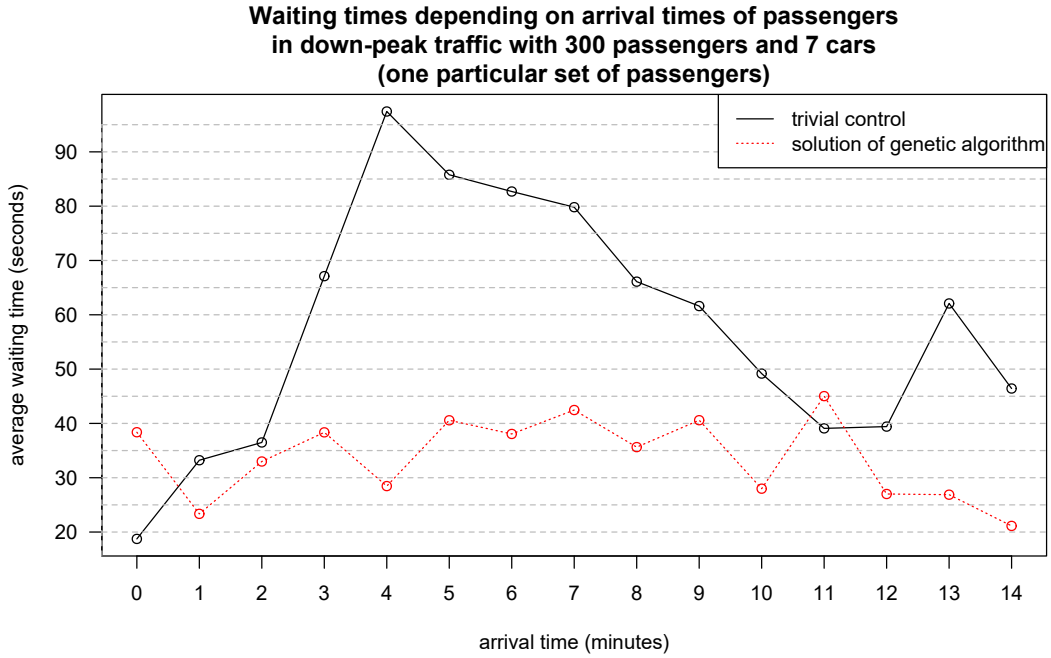

Figure 3.36: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the down-peak traffic in a twenty-storey building with one CMCE with 7 cars with a capacity of 10 persons.

number of cars	6	7	8	9	10	11	12
average waiting time	87	32	42	32	30	25	20

Table 3.7: AWT in the trivial control in seconds for one set of passengers arriving during 15 minutes in the lunch-peak traffic in a twenty-storey building depending on number of cars with a capacity of 10 persons in one CMCE elevator.

3.2.4.4 Down-peak with 8 cars

With 8 cars, the genetic algorithm found for the set of passengers described in 3.2.4.2 a solution with AWT of 34 seconds, while AWT in the trivial control was 51 seconds. Both simulations finished after more than 17 minutes. The optimized solution does not include AWT higher than a minute, which occurred in the trivial control for passengers arriving between the 6^{th} and the 9^{th} minute (see Figure 3.37).

3.2.4.5 Down-peak with 9 cars

With 9 cars, the genetic algorithm also found for the set of passengers described in 3.2.4.2 a solution with AWT of 33 seconds. AWT in the trivial control was 59 seconds. The simulation of the optimized control finished more than 17 minutes, about a minute earlier than the simulation of the trivial control, which finished after more than 18 minutes. The optimized solution does not include AWT of about a minute and higher, which occurred in the trivial control for passengers arriving after more than 5 minutes (see Figure 3.38).

3.2.4.6 Lunch-peak with 8 cars

For a set of 300 passengers in the lunch-peak traffic with 8 cars, the genetic algorithm found a solution with AWT of 31 seconds, while AWT of the trivial control solution was 42 seconds (see Table 3.7). Simulations of both solutions finished after about 16 minutes. The optimized control decreased high AWT of passengers arriving between the 8^{th} and the 12^{th} minute (see Figure 3.39), where AWT in the trivial control were over a minute.

3.2.4.7 Interfloor with 8 cars

For a set of 300 passengers in the interfloor traffic with 8 cars, the genetic algorithm found a solution with AWT of 41 seconds, while AWT of the trivial control was 44 seconds (see Table 3.8). The simulation of the trivial control finished after almost 18 minutes, while the simulation of the optimized control finished after almost 17 minutes. Unlike the trivial control, AWT in the optimized control do not exceed a minute for any minute in the period of arrival times (see Figure 3.40).

3.2.4.8 Interfloor with 9 cars

With 9 cars, the genetic algorithm found after 7000 iterations for the set of passengers described in 3.2.4.7 a solution with AWT of 34 seconds, while AWT of the trivial control was 49 seconds. The simulation of the optimized control

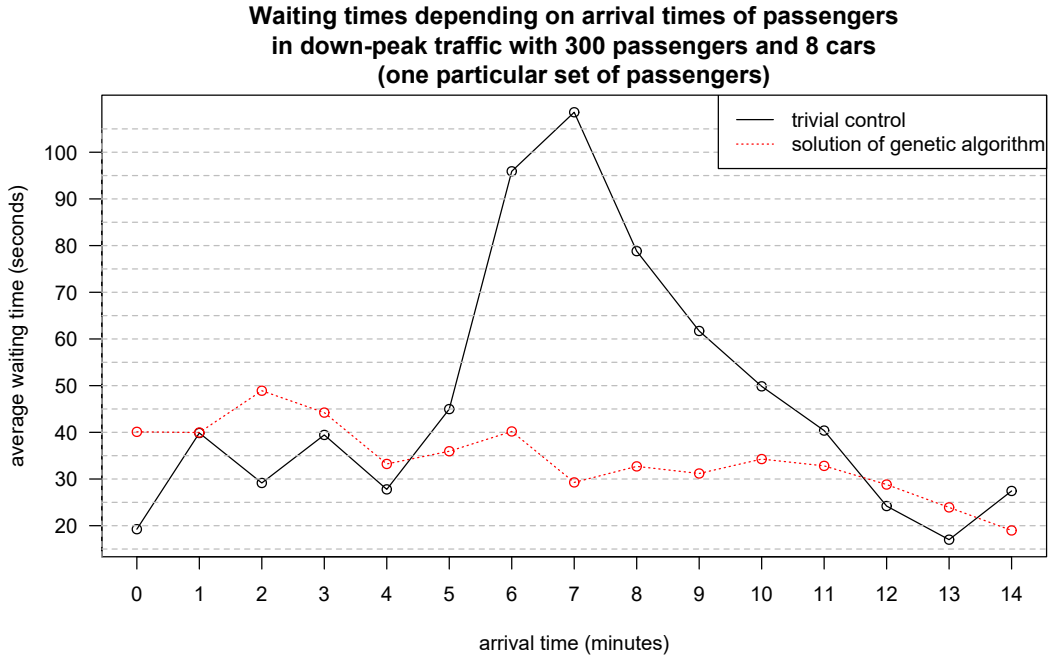

Figure 3.37: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the down-peak traffic in a twenty-storey building with one CMCE with 8 cars with a capacity of 10 persons.

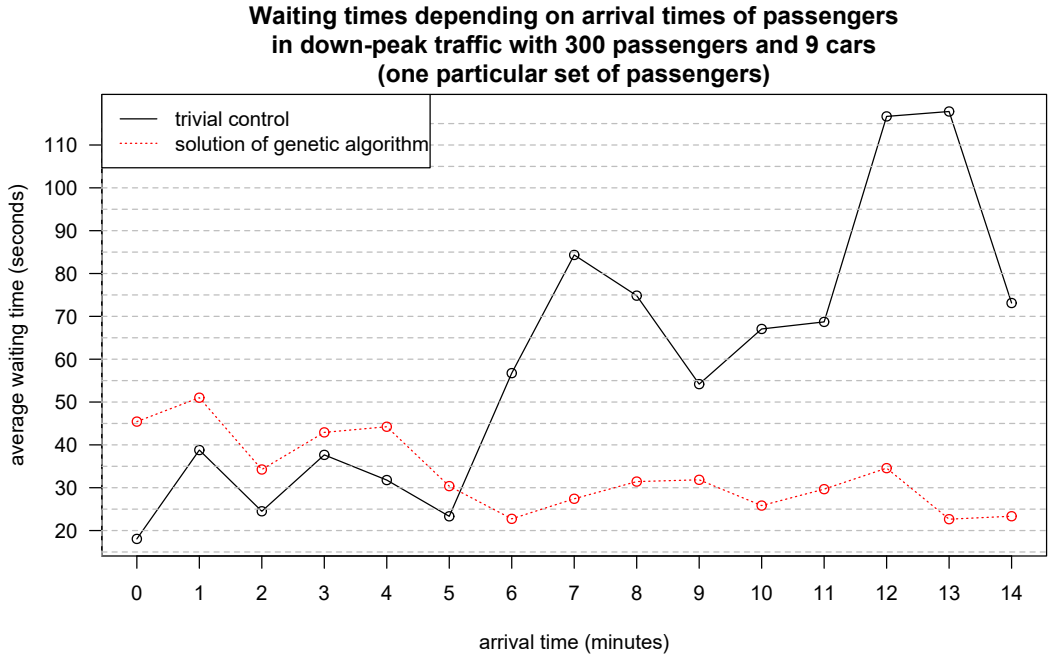

Figure 3.38: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the down-peak traffic in a twenty-storey building with one CMCE with 9 cars with a capacity of 10 persons.

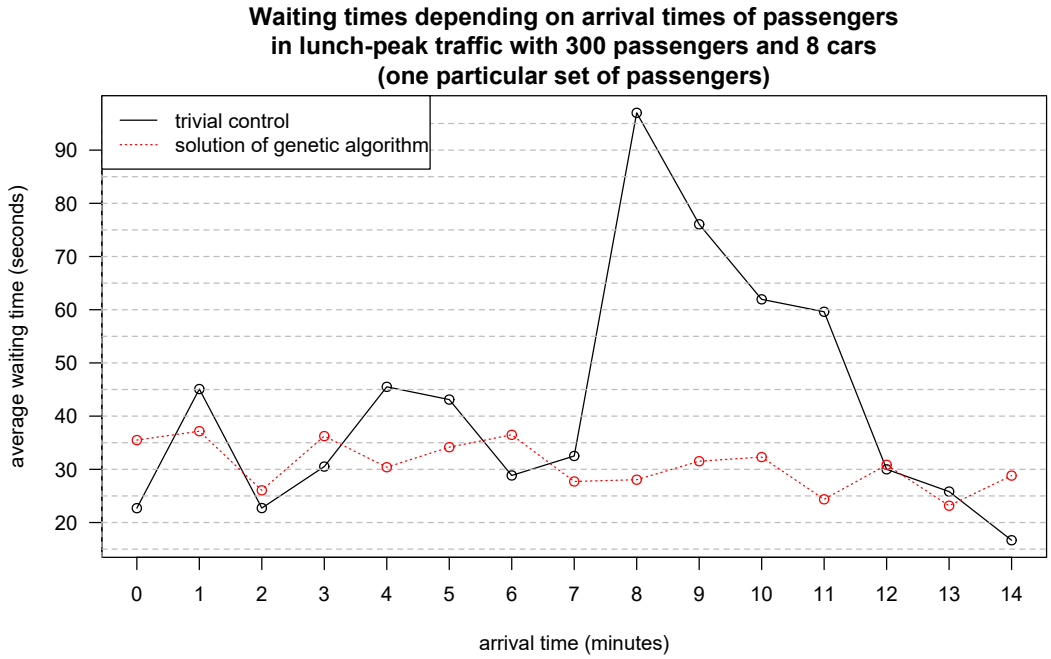

Figure 3.39: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the down-peak traffic in a twenty-storey building with one CMCE with 8 cars with a capacity of 10 persons.

number of cars	6	7	8	9	10	11	12
average waiting time	67	62	44	49	29	20	23

Table 3.8: AWT in the trivial control in seconds for one set of passengers arriving during 15 minutes in the interfloor traffic in a twenty-storey building depending on number of cars with a capacity of 10 persons in one CMCE elevator.

finished after about 16 minutes, while the simulation of the trivial control finished after about 17 minutes. The optimized solution eliminates high AWT occurring in the trivial control for passengers arriving after the 6^{th} minute (see Figure 3.41). However, AWT in the first five minutes are mostly lower in the trivial control.

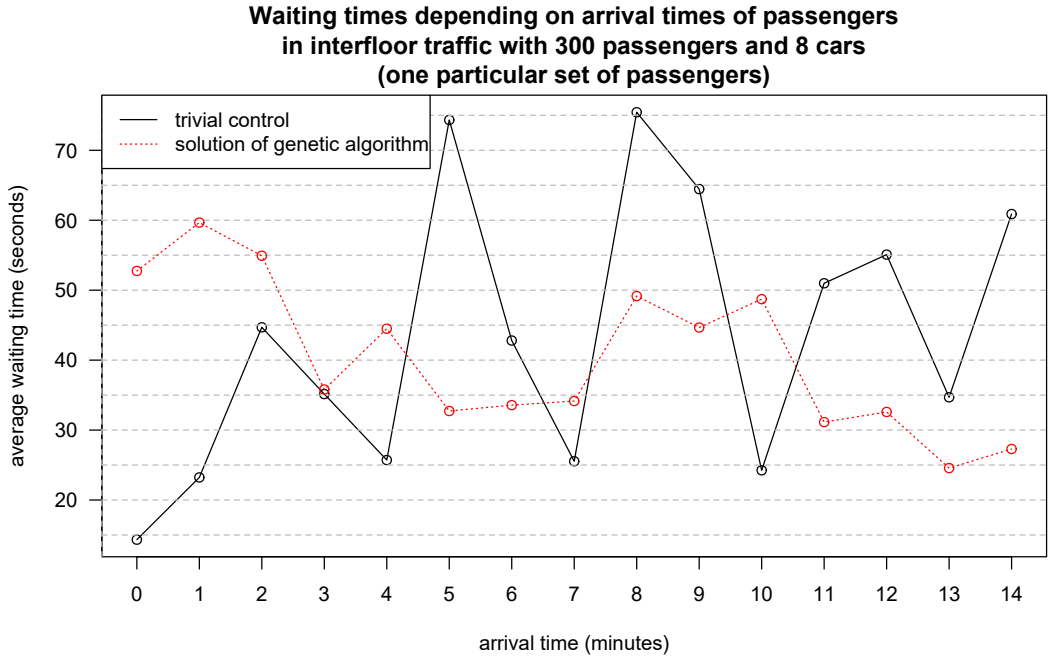

Figure 3.40: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the interfloor traffic in a twenty-storey building with one CMCE with 8 cars with a capacity of 10 persons.

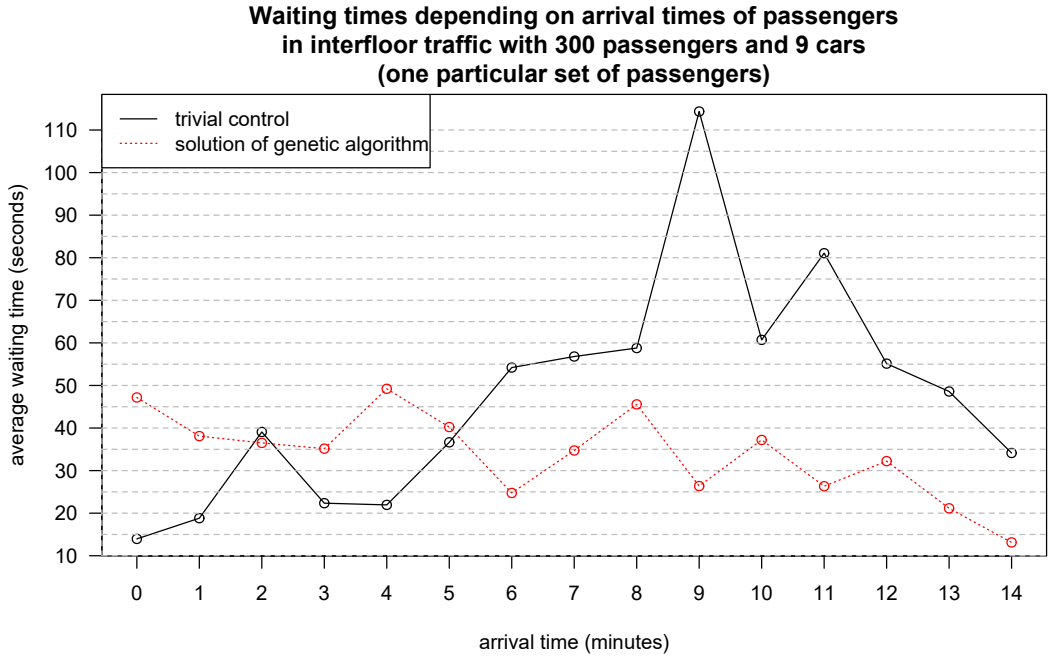

Figure 3.41: AWT (in seconds) depending of arrival time of passengers (in minutes) for one set of 300 passengers in the interfloor traffic in a twenty-storey building with one CMCE with 9 cars with a capacity of 10 persons.

3.3 Conclusion

Based on results listed in this chapter, AWT in the trivial control generally decreases with increasing number of cars and sometimes slightly increases with very high number of cars. With less cars, waiting times significantly increase with increasing number of passengers. With more cars, waiting times are sometimes slightly lower with higher number of passengers, since cars are circulating in the elevator and stationary cars are not blocking their movement. Increasing the car capacity generally improves waiting times, when there are few cars in CMCE. With more cars, higher car capacity is most useful in the up-peak traffic, where most passengers have the same origin floor, and in the down traffic, where most passengers have the same direction. Observations presented in this paragraph are valid for both buildings.

Judging by examples presented in subsection 3.1.4 and subsection 3.2.4, the trivial control is not optimal. In some cases, AWT could be reduced by at least 40 % by using a more efficient algorithm to control CMCE. In most cases, the highest AWT occurred in the trivial control for passengers, who arrived in the second half of the time period. However, AWT did not directly depend on the arrival time.

Results presented in subsection 3.1.4 and subsection 3.2.4 were obtained by the genetic algorithm with these parameters:

- the mutation rate: 0.01
- the percentage of elites excluded from mutation: 10 %
- the percentage of chromosomes replaced by random chromosomes: 0 %
- the size of subsets in the tournament selection: 3 chromosomes
- the size of a population: 150 chromosomes
- total iterations: 2000 to 7000

In general, algorithms with higher mutation rates, very low mutation rates, lower percentage of elites or smaller populations found worse solutions after the same number of iterations. Replacing a few percent of the worst chromosomes did not seem to affect the performance. Algorithms with bigger populations were slower. In algorithms with the populations size of 150, choosing parents in the tournament selection from more than three random chromosomes caused the algorithm to converge to worse solutions. On the other hand, choosing parents from less than three parents led to much slower convergence. The genetic algorithm with 7000 iterations took more than 6 hours to finish on a personal computer.

4. State of art

Efficient elevator control is a subject of interest of many researchers. This chapter mentions a few papers related to elevator control.

Y. Kuroda, M. Nakata and S. Yamaguchi [13] focused on optimization of multi-car elevator group control. They proposed an algorithm for minimizing the average service completion time. The algorithm is based on car collision avoidance realized by zone and direction restriction. They avoid car collisions by synchronizing directions of cars in a shaft and stopping cars until the collision is improbable. They compare the average service completion time in multi-car and single-car elevator groups in the up-peak, down-peak and interfloor traffic and in the traffic with more journeys to one special floor. Their algorithm is efficient especially in the up-peak and interfloor traffic. Their simulations show, that multi-car elevator systems controlled by their algorithm are more efficient than single-car elevator systems in high hall call rates and increasing the number of cars improves performance.

A. Valdivielso, T. Miyamoto and S. Kumagai [14] proposed a method of multi-car elevator group control including schedule completion time optimization, optimal parking for different traffic types and interference risk prevention based on synchronized scheduled directions. They compare their algorithm with a zoning algorithm. Their algorithm outperforms the zoning algorithm in the down-peak and interfloor traffic and in the up-peak traffic with lower passenger arrival rates.

M. Brand and D. Nikovski [15] focused on optimal parking of empty cars in a single-car elevator system in the up-peak and down-peak traffic. In the down-peak traffic, they split a building to zones, each served by one car. Then they compared the benefit of actively parking the cars at the center of their zones for different passengers arrival rates. In the up-peak traffic, they used Markov chains with dynamic programming to find the parking minimizing average waiting times in a limited time horizon and they evaluated this approach for different arrival rates.

M. Brand and D. Nikovski [16] focused on reduction of long waiting times based on using higher moments of the average waiting time in the optimization function. They introduced a method for computing these moments based on dynamic programming and Markov chains.

S. Ramalingam, A. U. Raghunathan and D. Nikovski [17] proposed an approach to group elevator scheduling problem based on maximizing a submodular function under a matroid constraint. They proposed an approximation of the total waiting time and formulated the problem as a quadratic Boolean optimization problem. Then they reformulated the problem as a submodular maximization problem under a matroid constraint. The solution was then found by a greedy algorithm. Their method with further enhancements achieved lower average waiting times in comparison with conventional methods.

M. Ruokokoski, H. Ehtamo and P. M. Pardalos [18] focused on formulation of the elevator dispatching problem as a mixed-integer linear program. They presented and improved the formulation of the problem. They proposed a reformulation of the problem with decreased number of variables and they also provided rules for removing unnecessary arcs from the graph of the problem. They

introduced new valid inequalities for the problem and described the polytope of the elevator dispatching problem in the down-peak period.

P. Cortes, J. Larrañeta and L. Onieva [19] proposed a genetic algorithm for controllers of single-car elevator groups. The fitness function of an elevator is the approximate time the elevator needs to serve all its calls. The function is computed from distances between the current floor and highest and lowest floors in the upwards and downwards direction and estimated floor-to-floor time. The function does not involve destination floors of passengers. The algorithm includes mutation, uniform crossover and random selection of parents. Chromosomes in a population are replaced by new chromosomes with probability based on their ranking in the population. They compared their algorithm with an algorithm used traditionally in industry. The genetic algorithm reduced the system waiting time, consisting of the waiting time and the trip time, by almost 25 %.

B. Bolat et al. [20] focused on the elevator group control based on a genetic algorithm. They used a roulette wheel selection method and compared single-point, two-point and uniform crossover and more mutation probabilities. The algorithm was tested on more buildings in a lunch-peak period. The optimized control was by 20-25 % more efficient regarding the average waiting time and the average journey time. Their fitness function can be computed fast, therefore their algorithm can be used for real time elevator control.

M. Baygin et al. [21] compared a genetic algorithm based elevator control with classical control, assigning the call to the closest car, and elevator control with passengers' destination floors entered using a numeric keypad. The methods were tested on a prototype of an elevator group in the interfloor traffic. The system with a keypad was by 20 % more efficient than the optimized system without the knowledge of destination floors.

J. Debnath and G. Serpen [22] proposed and tested methods of optimization of elevator scheduling in fully-automated parking structures. Their hybrid method is based on genetic algorithms and nested partitioning. A genetic algorithm is used during iterations of nested partitioning for finding the best subregion and comparing it with its surrounding region. They also provided a method based on the queueing theory for computing the minimum number of elevators for given number of floors in the parking structure.

L. Yu et al. [23] proposed a hybrid real-time scheduling algorithm for double-deck elevator group based on genetic network programming and ant colony optimization with evaporation. In their hybrid algorithm, they modify the new generation created by crossover using the pheromone instead of mutation once in ten iterations. Thanks to ant colony optimization, their algorithm often converges faster than an approach based only on genetic programming. Genetic programming increases the chance of finding the global minimum. They optimized average waiting time, percentage of persons waiting for more than 60 seconds and energy consumption in regular, up-peak and down-peak traffic. They demonstrated the efficiency of their algorithm via simulations.

C. E. Imrak and M. Ozkırım [24] focused on optimization of elevator group control using neural networks. Their feed-forward neural network with back-propagation allows adapting to new traffic patterns, while the traditional duplex/triplex algorithm for controlling groups of two or three cars is not efficient in the up-peak traffic. In comparison with traditional algorithms, the performance

of their algorithm based on combining a neural network with the duplex/triplex algorithm is better.

L. Cui [25] focused on intelligent elevator group scheduling in different types of traffic based on fuzzy neural networks with back-propagation. He proposed a new Gaussian membership function for the fuzzy set.

B. Bolat and P. Cortés [26] proposed algorithms for optimizing the average journey time in elevator group control based on genetic algorithms and tabu search. The fitness function included both initial and destination floors of passengers and balance of service of cars in the elevator group. In their simulations, the probabilistic tabu search was the most efficient regarding average journey and waiting times. However, genetic algorithms were faster.

B. Bolat, O. Altun and P. Cortés [27] proposed an algorithm for elevator group control based on particle swarm optimization. They compared their algorithm with two genetic algorithms and a tabu search algorithm in buildings with 10 to 24 floors and 2 to 6 cars. Their algorithm with 30 particles and 30 generations was more efficient regarding average journey times and also computational times.

B. Bolat et al. [28] focused on different meta-heuristic algorithms for optimization of journey times in elevator group control. They also proposed a new version of simulated annealing called simulated annealing with random restarts. In their version, the local search restarts from a random position, if the local search does not improve the current solution after a given number of attempts. They compared the differential evolution, simulated annealing with random restarts, artificial bee colony, bat algorithm, bacterial foraging optimization algorithm, particle swarm optimization, genetic algorithm and tabu search. In buildings with 10 to 24 floors and 2 to 6 cars with at most 10000 function evaluations allowed to each algorithm, artificial bee colony and tabu search were the most efficient algorithms for reducing the average passenger journey time. Their algorithm simulated annealing with random restarts was the third best algorithm.

5. User interface

This chapter describes the graphical user interface of the program created within this thesis. A user can interact with the program via forms. Some parameters can be loaded from files and the program writes the results to files (see section 6.4).

5.1 Input of parameters

The user interface is opened by running the program CirculatingMultiCarElevatorGUI.exe. There must be the files CirculatingMltiCarElevator.dll and CirculatingMultiCarElevatorLogLines.dll in the same directory.

After starting the application, a form is shown (see Figure 5.1). In this form, the user can set model properties. The user enters parameters of the building, elevators and traffic in required units. The settings can be loaded from a file or saved to a file. The user also selects passengers for simulations. If the settings are not valid, the application displays a message. By closing this form, the application is closed and all of the computations are cancelled.

Properties of elevators include:

- the number of elevators in the elevator system
- the number of cars in each elevator
- the capacity of cars
- the time required to open the door of a car
- the time required to close the door of a car
- the maximum velocity of a car
- the acceleration of a car
- the deceleration of a car
- the time required to move a car from one shaft to the other shaft
- the height of a car
- the minimum distance between two cars

Minimum distance is the distance in a shaft, which will be always preserved between two cars during a simulation. If the elevator system contains more elevators, it is assumed, that the properties of all elevators are the same. The sum of the minimum distance and the height of a car must not be higher than height of a floor. The height of a floor can be set in the section with building parameters. Maximum number of cars is set to 1.5 * number of floors in order to provide enough space for movement of cars.

Properties of a building include:

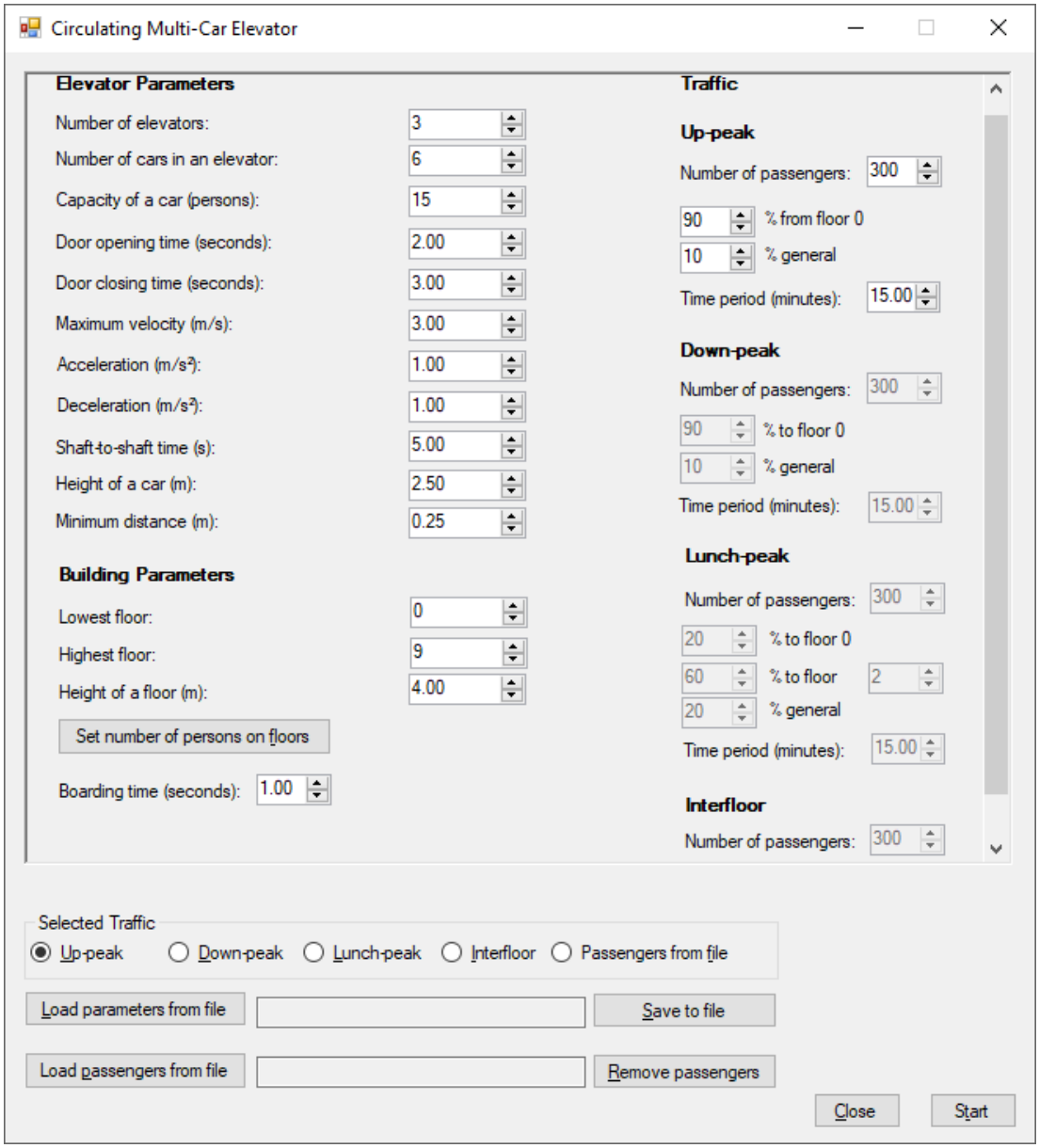

Figure 5.1: Model parameters. The user selects parameters for the building, elevators and traffic. Parameters can be also loaded from a file or saved to a file. On the bottom, the user selects the traffic type or loads passengers from a file.

- the number of the lowest floor
- the number of the highest floor
- the height of a floor
- the time required for boarding and alighting of a passenger

By clicking the button labelled Set number of persons on floors, the user can display another form (see Figure 5.2). In this form, the user can set for each floor the number of persons who have their offices on that floor. New settings can be saved by clicking the button Save or cancelled by clicking the button Cancel.

On the right of the form with model parameters settings (see Figure 5.1), the user can set parameters of the up-peak, down-peak, lunch-peak and interfloor traffic. Controls are enabled based on the traffic type selected on the bottom of the form. For each type of traffic, the user must enter specific parameters. On the basis of the selected traffic parameters, passengers for simulations will be generated.

In the up-peak traffic, most persons travel from the lobby to their offices. In the down-peak traffic, most persons travel from their offices to the lobby. In the lunch-peak traffic, some persons travel from their offices to the canteen and some persons travel from their offices to the lobby. In the interfloor traffic, persons travel either from their offices to various floors or from various floors to their offices. For each type of traffic, the user sets the number of passengers in a simulation. Then the program will generate a random subset of this size of all possible passengers for simulations. The user sets percentages of journeys to the lobby, from the lobby and to the canteen and the percentage of general traffic, which consists of journeys between various floors.

For arrival times of passengers, the program will generate random numbers falling in the period of the traffic chosen by the user. A simulation begins at the time zero and ends after the last passenger leaves the car on their destination floor. The application is not suitable for simulating extremely heavy traffic, because the simulation is always stopped when the time is higher than twice the time period of arrival times of passengers and the maximum value of the integer data type is assigned to the average waiting time. This action eliminates extremely bad solutions during the genetic algorithm, especially when the time period of arrival times is short.

If the option Passengers from file is selected, the user must enter the file containing the passengers. By clicking the button labelled Remove passengers, the passengers previously loaded from a file are removed. By clicking the button labelled Save to file, the settings selected in the form are saved to a file in the valid format and can be loaded later.

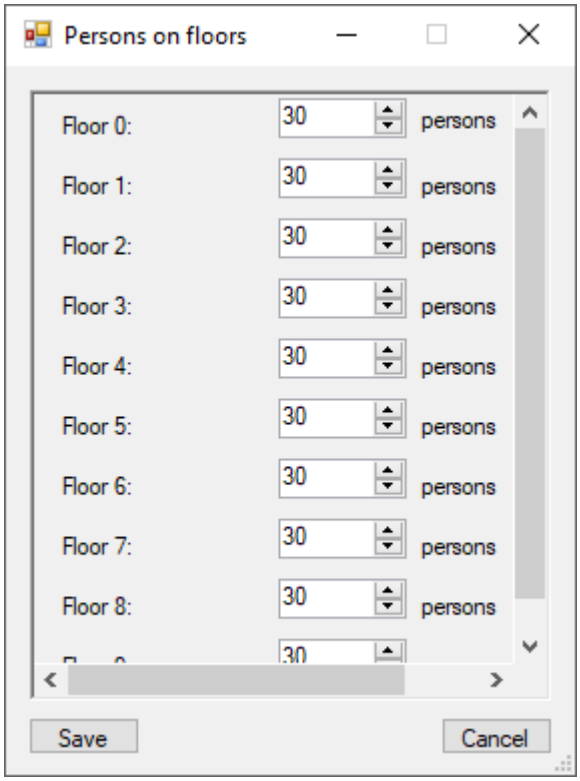

Figure 5.2: Settings of the number of persons, who have their office on particular floors.

5.2 Running the algorithm

After clicking the button Start, the application displays a form with a summary of the model's properties (see Figure 5.3). The user can run a simulation with assignment of passengers to cars saved in a file, run a simulation of the trivial control or run a genetic algorithm. The user must select the file, to which the result will be written. The user can also choose a file for saving generated passengers. If the user selected the option Genetic algorithm, they can select parameters for a genetic algorithm:

- the number of iterations
- the size of a population
- the mutation rate
- the value, by which the mutation rate decreases between generations
- the minimum mutation rate
- the number of rivals in tournament selection
- the percentage of elites in a population
- the percentage of chromosomes, which will be replaced by new chromosomes

If the option Simulation with given assignment is chosen, the user must choose a file with assignment of cars to passengers in the required format (see section 6.4).

Mutation rate indicates the ratio of the number of mutated genes and the total number of genes in a population. At the end of each iteration, the mutation rate is decreased by the given mutation rate decrease. Mutation is not performed on elites. At the end of each iteration, the worst chromosomes are replaced by new random chromosomes.

After clicking the button Start, the calculation starts. If the user selected the genetic algorithm, the form shows another form displaying the progress of the algorithm (see Figure 5.4). If the user selected the simulation of the trivial control or simulation with given assignment, the form with results is shown (see Figure 5.5). If the application could not generate passengers on the basis of the traffic selected in the previous form, the form displays a message.

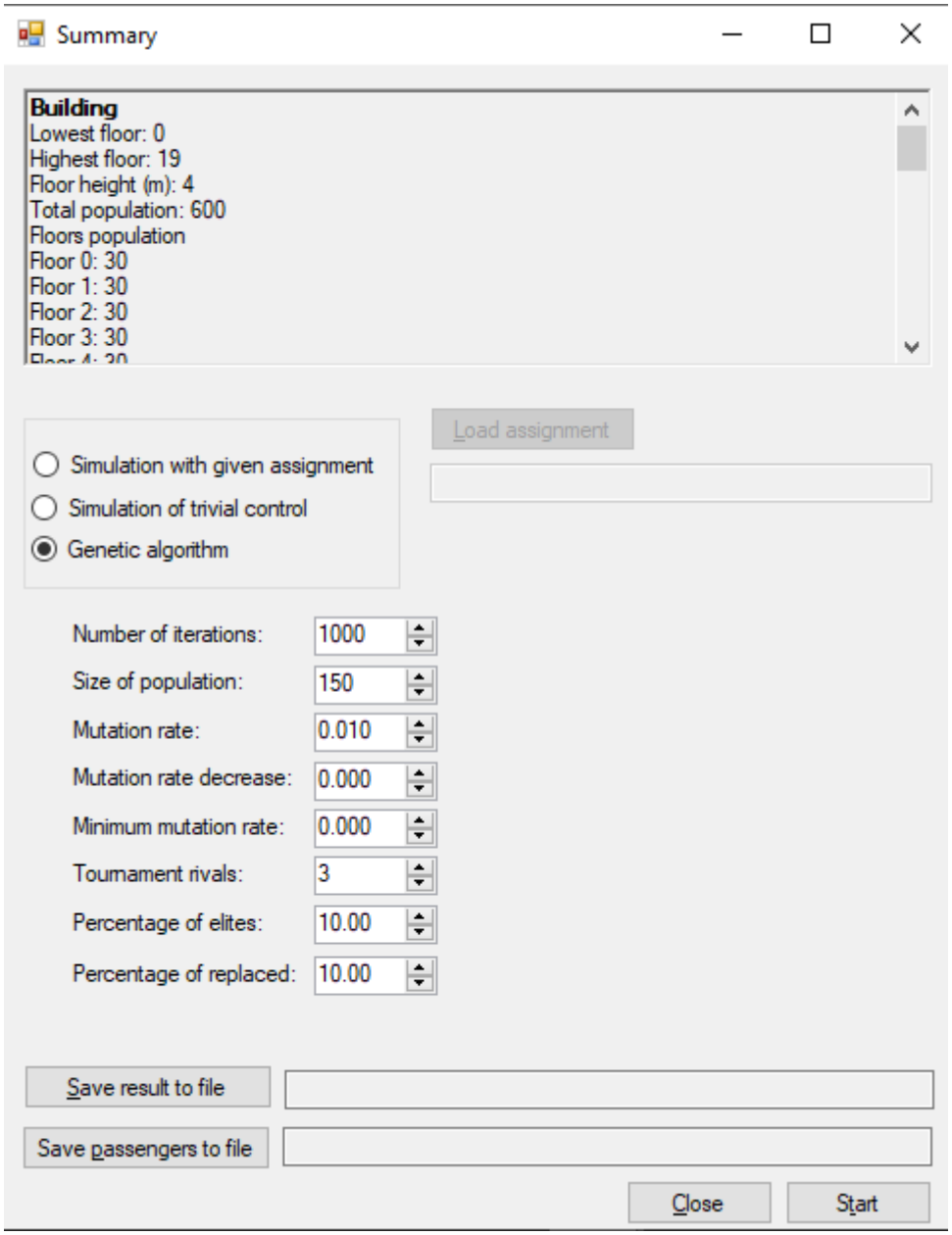

Figure 5.3: In this form, the user can run a simulation with an assignment of passengers to cars loaded from a file, a simulation of the trivial control or a genetic algorithm with configurable parameters.

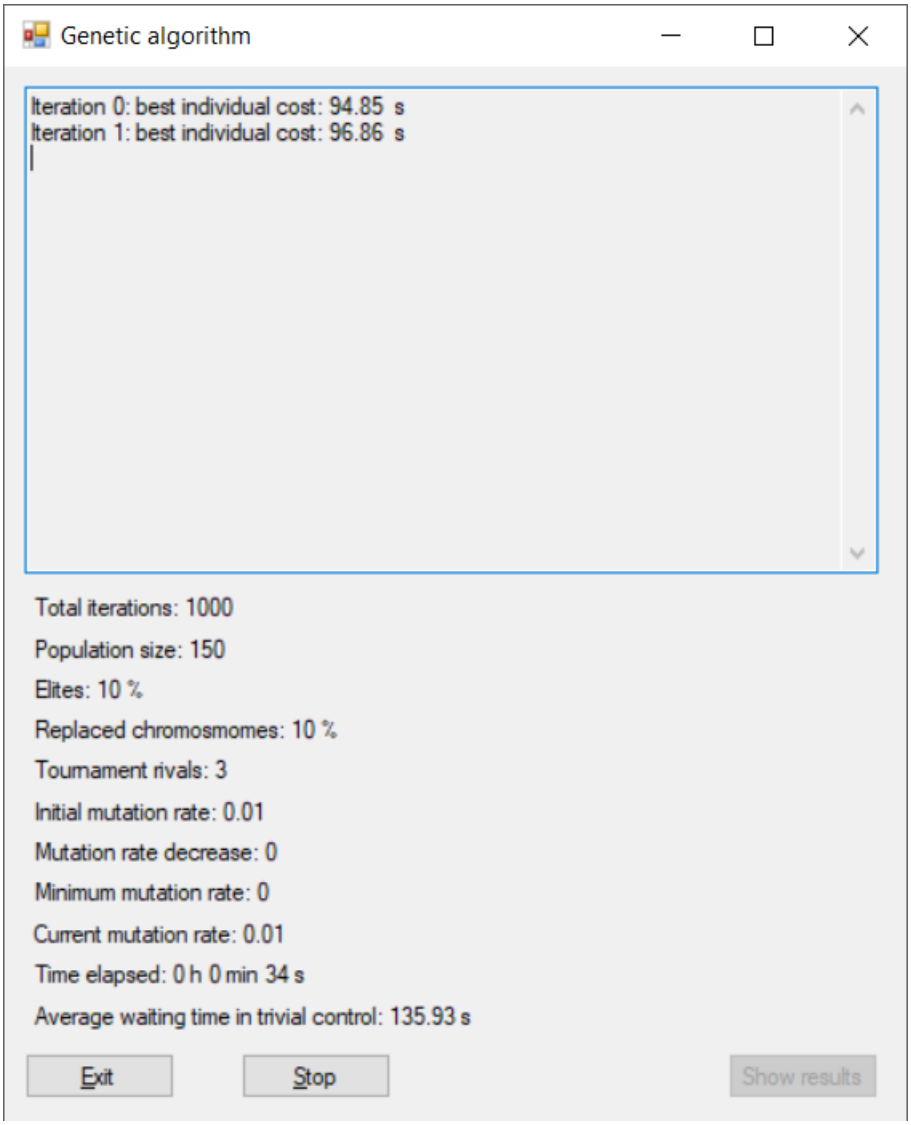

Figure 5.4: Genetic algorithm progress. This form displays the lowest cost in each generation, elapsed time and algorithm parameters and allows cancelling the computation or stopping the evolution after the current iteration.

5.3 Genetic algorithm progress

On the top of the form showing the genetic algorithm progress (see Figure 5.4), there is a box with the evolution log. For each iteration, the box contains a line with the cost of the best individual in the generation. The form is updated once in a second. Under the box, there are parameters of the algorithm. The form also displays the time elapsed from the beginning of the computation and the average waiting time in the trivial control for comparison.

When the algorithm finishes, the form displays another form with the results (see Figure 5.5). The form containing results can be displayed again by clicking the button Show results. The user can stop the evolution by clicking the button Stop. After clicking this button, the evolution will be stopped, after the current iteration finishes, and the results will be displayed. By clicking the button Exit, the calculation is cancelled and the form is closed.

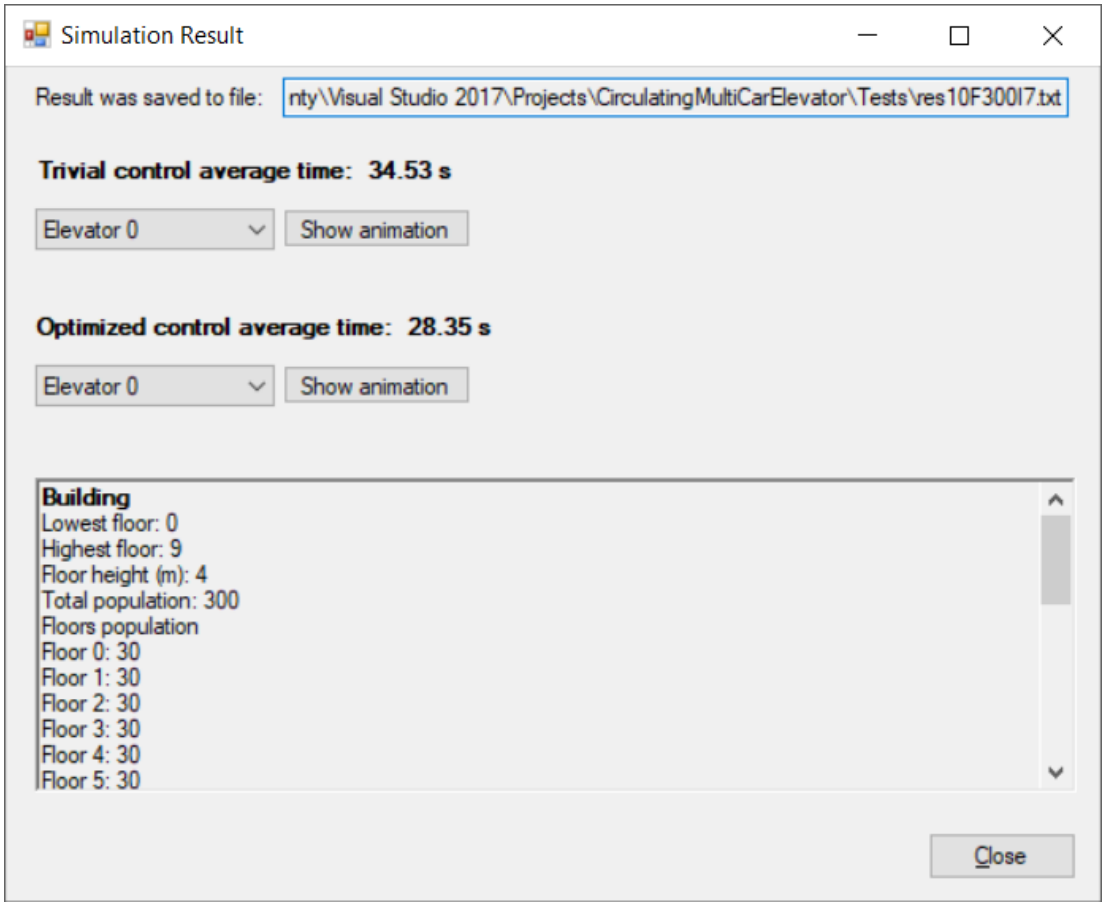

Figure 5.5: Simulation results. The user can view an animation of the simulation for the selected elevator in the selected elevator control.

5.4 Results

The form with simulation results (see Figure 5.5) shows average waiting times in the trivial control simulation and in the solution found by the genetic algorithm. The log of the simulations is written to the selected file (see subsection 6.4.4).

5.5 Animation

In the form with simulation results (see Figure 5.5), the user can select one elevator and view a simple animation of a simulation in the trivial or optimized control. The animation is shown in a new form (see Figure 5.6). In the picture on this form, there is a shaft with cars moving upwards on the left, the shaft with cars moving downwards on the right and left and right sides of hallways in the middle. Passengers on the left side of a hallway are waiting for a car moving upwards and passengers on the right side of a hallway are waiting for a car moving downwards. In the middle of a hallway, numbers of passengers waiting on both sides of the hallway and the number of the floor are shown. Next to the bottom right corner of the picture, there are four buttons labelled by the plus or minus sign. These buttons allow zooming the picture in or out in the vertical or horizontal direction. The simulation can be progressed manually, by clicking the button labelled Next step, or by a timer with the selected interval. The time period between steps can be set in the bottom left corner of the form. Buttons labelled Start and Stop allow starting or stopping the timer. By clicking the button labelled Finish, the simulation is finished. The user can view the animation again from the beginning by clicking the button Repeat, which appears after finishing of the animation.

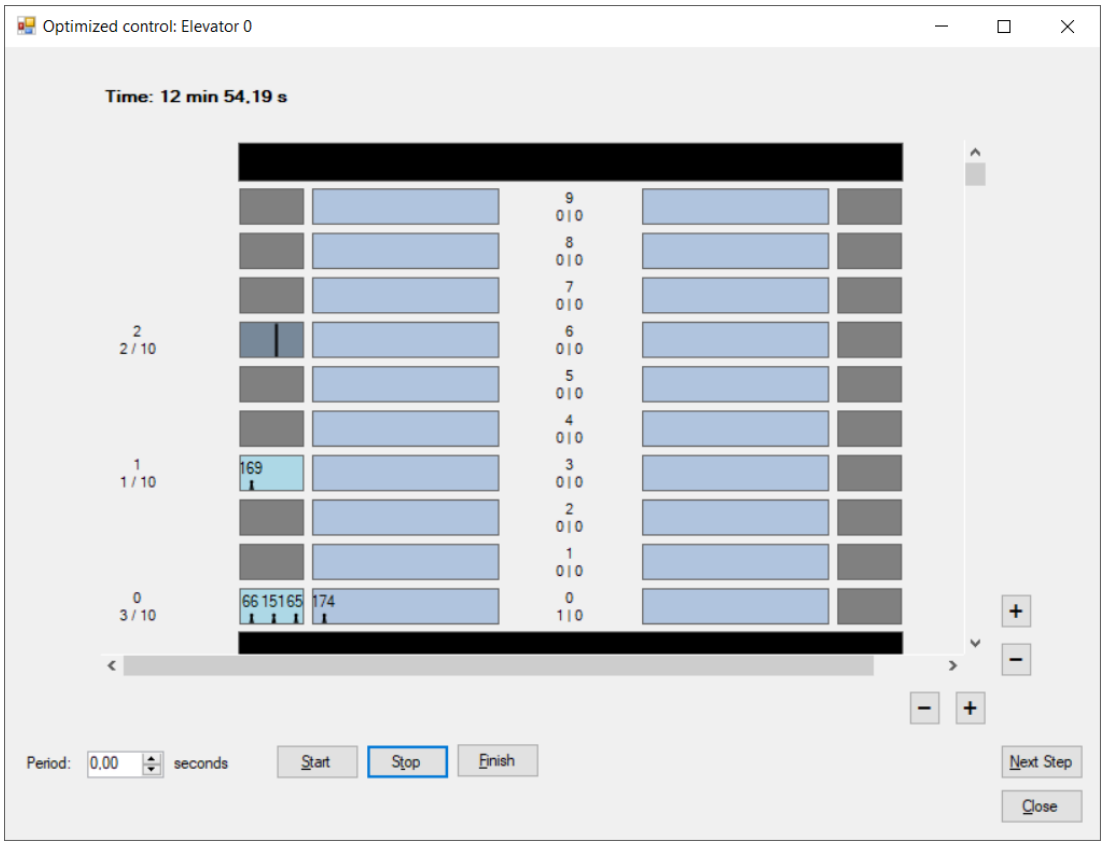

Figure 5.6: The animation of a simulation. In the picture, there is a building with 10 floors. On the floor number zero, there is the car number 0. There are passengers number 66, 151 and 65 inside this car. The passenger number 174 is still waiting in the hall. On the floor number three, there is the car number 1 with the passenger number 169 inside. On the floor number 6, there is the car number 2 with two passengers inside. The door of this car is closed. These three cars are moving upwards. After reaching the top of the left shaft, they will move to the right shaft and continue to move downwards in the right shaft. The animation can be progressed manually or by a timer using the buttons under the picture. The picture can be zoomed in or out horizontally or vertically.

6. Software documentation

This chapter contains an overview of classes and their relations in the program created within this thesis. The program (see section A.2 and section A.3) is created in C# 7.0 and developed with .NET Framework 4.6. The solution consists of three projects: CirculatingMultiCarElevator, CirculatingMultiCarElevatorGUI and CirculatingMultiCarElevatorLogLines.

6.1 Communication with GUI

The library CirculatingMultiCarElevatorLogLines contains an interface representing the object displaying the animation of one CMCE elevator and the hierarchy of descendants of the abstract class Line, representing logs of simulations.

6.1.1 Animation

The object used for the animation must implement the interface IElevatorAnimation. In the project CirculatingMultiCarElevatorGUI, the animation is performed by the form AnimationForm.

6.1.2 Lines of a log

The abstract class Line is used to create a log from a simulation and display the events in graphical form. Descendants of this class include information about a given event. The class contains the abstract method Process called by an instance of a class implementing the interface IElevatorAnimation to change the state of the object on the basis of the event.

Descendants of the class Line include the abstract class CarLine, representing an event of a car, ModelLine, representing an event of a model and PassengerLine, representing an event of a passenger.

Descendants of the class CarLine, representing different events of cars, include classes CarPlacedLine, CarCalledLine, CarArrivalLine, DoorOpenedLine, DoorClosedLine and CarLeavingLine. The descendant of the class ModelLine are the classes StartLine and EndLine, representing the beginning and the end of a simulation. The descendants of the class PassengerLine are the classes PassengerArrivalLine and the abstract class PassengerInCarLine with descendants GotInLine and GotOutLine.

6.2 CMCE simulation and optimization

Classes for the CMCE simulation and optimization using a genetic algorithm are located in the library CirculatingMultiCarElevator.

6.2.1 Model representation and simulation

This section contains the documentation of classes used for a discrete event simulation.

6.2.1.1 Model

The class Model represents a model of a discrete event simulation. During the simulation, events are planned to a calendar. Events are then processed by their subjects in the order of their time. Model provides the method Simulate, which allows running a simulation and returns an instance of the class SimulationResult.

Model has a reference to one instance of the class Building, ElevatorSystem, Controller and MechanicsComputations. Model contains a list of instances of the class Passenger and at most one instance of Traffic, which is used in creating the model description. Model provides access to its instances of classes Building, ElevatorSystem, Traffic and MechanicsComputations and current time via public properties.

Model provides methods for planning and removing events and appending lines to a log (PlanEvent, UnplanEvent and Log), called by descendants of the class Process during a simulation. Public methods of the class Model also include the method CallCar called by passengers and the method CarCapacityFull called by a car to inform the model that the limit of its capacity has been reached. During planning their movement, cars call the method CanChangeShaft to find out whether they have enough space to move to the other shaft and the method GetAvailablePosition to get the furthest available position on their path to their next stop.

An instance can be created by the public static method CreateModel. The class model allows creating a deep copy by calling the method DeepCopy. This method creates deep copies of objects in a model, including an elevator system, calendar with events and passengers. Only the instance of the class Building is not copied, because it only contains properties which do not change during a simulation.

The class SimulationResult contains public properties representing the average waiting time computed in the simulation, assignment used in the simulation, the list of passengers and the list of lines logged by the model during the simulation. An instance can be created by the public static method CreateSimulationResult.

The class Controller contains the mode of elevator control represented by the enumeration class ControllerMode, either the trivial control (ControllerMode.Trivial) or a control with passengers assigned to cars in advance (ControllerMode.Assigned). In the second case, the model contains an instance of the class Assignment, which represents such an assignment. Controller contains methods for assigning calls to cars, which are called by a model and calls appropriate methods of a model.

The class Assignment contains a list representing cars assigned to passengers in the order given by passengers' identification numbers. Assigned cars are represented by their total order in the whole elevator system. The class contains the public static methods for creating a random assignment with given number of passengers and total number of cars.

Computations connected with movements of cars (distance, velocity or time) are realized by an instance of the class MechanicsComputations, owned by a model. Current state of the model is represented by an instance of the class State. This class manages model's current time and waiting passengers. It in-

cludes public method for placing passengers to their starting floors, getting waiting passengers from floors and removing passengers, and the public constructors.

6.2.1.2 Events

Events in a discrete event simulation are represented by instances of the class Event. An instance contains time of the event, the type of the event, represented by an instance of the enumeration class EventType, and an instance of a descendant of the abstract class Process representing the subject of the event.

Descendants of the class Process are classes Model, Passenger, Doorman and Car. These classes contain the method ProcessEvent, which is called by a model during a simulation.

6.2.1.3 Types of events

The types of events connected with a model are represented by instances Start and End, which represent the beginning and the end of a simulation.

The instances corresponding with cars are ArrivingToFloor, DoorOpened, StartClosingDoor, DoorClosed, ArrivalPlanning, ConstantVelocity, Decelerating, ArrivingToFloor, ArrivingToPosition, GettingToOtherShaft and BlockingCarNotification.

The event of the type ArrivalPlanning starts planning future movement of a car. The event StartClosingDoor represents the beginning of closing a door of a car. The closing of the door can be cancelled, if there are passengers on the floor wishing to enter the car, who were not present during boarding. The event type ArrivingToPosition indicates stopping at a position, which is located between floors. This type of stop is caused by moving away from the path of another car or waiting for another car to move away. The event of the type BlockingCarNotification is planned by a car blocking movement of another car to make that car re-evaluate its path.

Types of events of a passenger are PassengerArriving, GettingIn and GettingOut, representing entering and leaving a car.

The event planned by an instance of the class Doorman to plan arrivals of next passengers is represented by the instance DoormanAction.

6.2.1.4 Passengers

Instances of the class Passenger contain information about the passenger's origin and destination floor, the car and elevator used by passengers and their arrival time and waiting time.

Public methods include the method AssignCar called by a model and the method StopWaitingAt called by a car, when the final waiting time of a passenger is known. An instance can be created by the public static method CreatePassenger, a copy can be created by the method DeepCopy.

Arrivals of passengers are planned by an instance of the class Doorman. A model creates an instance of the class Doorman at the beginning of a simulation. The doorman then plans an event of the type DoormanAction to the time, when the first passenger arrives. During this event, the doorman plans arrivals

of passengers arriving at that time and then plans its event to the next arrival time.

6.2.1.5 Cars

The class Car represents a car in an elevator. Cars are ordered by a model to move towards incoming passengers. During their movement, cars request unoccupied positions, to which they can move, from a model.

An instance of this class contains a list of instances of the class Passenger, representing the passengers inside the car. Cars use enumeration classes CarMovement and CarAction for representing their current state. The movement of a car is represented by an instance of the class CarMovement, containing instances Accelerating, MovingWithConstantVelocity, Decelerating, NotMoving and ChangingShafts. The current action of a car is represented by an instance of the class CarAction, including instances MovingToStop, MovingAway, Boarding, DoorOpening, DoorClosing and None.

While planning its next movement, the car requests the next floor to stop from its instance of the class StopsManager, which manages its future stops and returns them in the right order.

The car provides the method Call, called by an instance of the class Elevator to call the car to the required floor. The car also contains the method UnassignStop called in trivial control by a model while assigning a passenger to another car with free capacity. Other public methods include the method Placed called by an elevator at the beginning of a simulation, method GetCurrentPosition called by a car or a model during planning paths of cars, method Distance called by an elevator in the trivial control to find the closest car for a passenger and the public constructor. The method DeepCopy creates a deep copy of an instance.

Classes Car and StopsManager use instances of classes Floor and PositionToMove, both descendants of the abstract class Position, for representing positions in a shaft.

6.2.1.6 Building

An instance of the class Building contains public properties describing a building. These properties include numbers of the highest and the lowest floor, total number of floors, height of a floor and a shaft, total population and a list of numbers of persons on each floor.

Public methods include methods for conversions between numbers of floors and indices of floors, where floors are indexed from zero, methods for conversions betweens floor numbers and positions in shafts and the method PositionBetween, which receives three floors and decides, whether the second floor lies between the other floors. An instance can be created by the public static method CreateBuilding.

6.2.1.7 Elevator system

An instance of the class ElevatorSystem contains a list of elevators in the elevator system. Since the properties of all elevators in the elevator system are

the same, an instance of this class contains also properties describing elevators. These properties include the number of elevators, number of cars in an elevator, door opening and closing time, maximum velocity, acceleration and deceleration, height of a car, minimum distance between cars, number of floors, time required to move a car to another shaft and capacity of cars.

Public methods of this class include methods for assigning a car to a passenger, called by an instance of the class Controller, methods returning a required car or elevator called by cars and model, method for placing all cars in all elevators in the system and resetting cars' positions, both called by a model.

An instance can be created either by the public static method CreateElevatorSystem or by the public constructor by copying properties of existing system. By calling the method DeepCopy, a deep copy of an instance is created.

6.2.1.8 Elevators

An instance of the class Elevator manages a list of cars. Public methods include methods for assigning passengers to cars, getting cars and placing cars, all called by the corresponding elevator system. An instance can be created by the public static method CreateElevator. This class contains the method DeepCopy.

6.2.1.9 Traffic

The abstract class Traffic represents a type of traffic in a building. An instance of a descendant of this class contains public properties representing total number of passengers, ratio of passengers in general traffic and time period of passengers' arrival times. The class includes the public abstract method CreatePassengers, which creates a list of passengers on the basis of the traffic.

The descendants of this class are classes UpPeak, DownPeak, LunchPeak and Interfloor. The class UpPeak extends the properties of the class Traffic by ratio of passengers travelling from the ground floor, DownPeak adds ratio of passengers travelling to the ground floor, LunchPeak adds ratio of passengers travelling to the ground floor, the number of the floor with a restaurant or a canteen and ratio of passengers travelling to this floor.

6.2.2 Genetic algorithm

This section contains the documentation of classes used in the implementation of a genetic algorithm. During the genetic algorithm, some operations are run asynchronously using System.Threading.Tasks.Task, System.Threading.Tasks.Task<TResult> and the async/await concept [29].

6.2.2.1 Evolution

The class Evolution represents a genetic algorithm. At the beginning of the evolution, an initial population of random chromosomes is created. In each generation, a new population is created by crossover. Then mutation is performed on chromosomes in the new population, except for elite chromosomes. The worst

chromosomes are removed from the new population and replaced by new random chromosomes. The best chromosome from all generations is returned from the evolution.

The class Evolution contains the public method RunAsync, which allows running the evolution. When the initial population is created, costs of chromosomes are computed asynchronously (in the method InitialPopulationAsync). During crossover, new chromosomes are created and their costs are computed asynchronously (in the method NextGenerationTournamentSelectionAsync). An instance can be created by the public static method CreateEvolution. This method accepts an instance of the class ChromosomeEvaluator for computing costs of chromosomes and parameters for evolution, including population size, mutation rate, mutation decrease between generations, minimum mutation rate, number of rivals in the tournament selection, maximum number of iterations, percentage of elites and percentage of chromosomes, which will be replaced by random chromosomes.

6.2.2.2 Chromosomes

The class Chromosome represents a chromosome used by the class Evolution. A chromosome contains an instants of the class Assignment, representing its assignment of passengers to cars. The class includes the static method Tournament, allowing running the tournament between a list of chromosomes, called by the class Evolution.

An instance contains the public method Crossover returning a pair of new chromosomes created by crossover with another instance. The cost of a chromosome can be computed by calling the method ComputeCost. For computing the cost, the chromosome uses an instance of the class ChromosomeEvaluator passed to this method. After the cost is computed, the instance saves the cost to return it immediately, when the method ComputeCost is called again. The instance also contains a partial simulation, a model in the state after half of the passengers arrived. Partial simulation is assigned to the chromosome by ChromosomeEvalutator after computing the cost or creation of the chromosome by single-point crossover. An instance can be created by the public static method CreateRandom, which creates a new chromosome with a random assignment.

6.2.2.3 Computing of costs of chromosomes

An instance of the class ChromosomeEvaluator contains references to classes Building, ElevatorSystem, Traffic and a list of passengers. An instance can be created by the public static method CreateChromosomeEvaluator. The class contains the public method Evaluate, which evaluates the chromosome passed as an argument and returns its cost. For evaluating the chromosome, the instance of the class ChromosomeEvaluator creates an instance of the class Model, calls model's method Simulate and returns the computed average waiting time. After performing simulations, ChromosomeEvaluator saves the computed costs.

6.2.2.4 Population

Instances of the class Population represent a generation in the algorithm run by the class Evolution. An instance of this class contains a list of chromosomes in the generation, an instance of the class ChromosomeEvaluator used for evaluating the chromosomes, its size and length and total number of cars of assignments in this population.

The method MutationAsync performs mutation on all chromosomes in the population using the given mutation rate and number of elites excluded from mutation. The method ReplaceWorstAsync replaces the worst chromosomes by new random chromosomes. Costs of mutated chromosomes and new chromosomes are computed asynchronously. The method FindBest returns the chromosome with the lowest cost of the population. Another public methods include a method returning random chromosome from the population and a method for setting the values in the assignment of the given chromosome. Chromosomes are added to the population by the method Add. An instance of the class Population is created by the public static method CreatePopulation. Methods of this class are called by an instance of the class Evolution during a genetic algorithm.

6.3 GUI

The user interface is implemented in the project CirculatingMultiCarElevatorGUI. This project contains classes for reading from files, classes for writing to files and forms for the input of parameters, running a genetic algorithm and simulations, displaying results and animations of CMCE control.

6.3.1 Files

This section contains the documentation of classes used for reading from files and writing to files. Instances of these classes are used by forms.

6.3.1.1 Format of files

The class FileFormatInfo specifies the format of files generated by the application. The class provides keywords, delimiters and integers representing number of parameters. These values are used by writers and readers.

6.3.1.2 Writers

The class ParametersWriter provides writing asynchronously properties of the model to a file in the required format. The class PassengersWriter provides saving generated passengers to a file. For writing asynchronously to files, these classes use the method System.IO.StreamWriter.WriteLineAsync of the class System.IO.StreamWriter.

6.3.1.3 Readers

Classes ParametersReader and PassengersReader provide loading model properties and passengers asynchronously from files. The class ParametersReader re-

turns parameters from a file in the form of a list of integers from its method LoadParametersAsync. The class PassengersReader returns a list of instances of the class Passenger from its method ReadPassengersAsync.

For reading from the file, both classes use an instance of the class FileReader. This class includes the method ReadKeywordAsync for reading a given keyword, the method IsComment, which recognizes a commented line, and the method ReadLineAsync. The class implements the interface System.IDisposable. FileReader uses the method System.IO.StreamReader.ReadLineAsync of the class System.IO.StreamReader to read from files asynchronously.

6.3.2 Forms

This section contains the documentation of forms. Forms are descendants of the class System.Windows.Forms.Form. There are some extension methods in the static class Extension simplifying working with components.

6.3.2.1 Parameters of the model

ParametersForm accepts parameters of a model. For obtaining populations of floors, the form creates an instance of the form BuildingPopulationForm. ParametersForm creates instances of the classes Building and ElevatorSystem and either an instance of the class Traffic or a list of passengers and passes them to SummaryForm.

6.3.2.2 Simulations and optimization

An instance of the form SummaryForm displays the summary of a model and requests the file for writing the result. The form creates passengers, if it did not receive them from ParametersForm, and writes them to a file, if it is required. The form runs the simulation of trivial control. Then the form shows GeneticAlgorithmProgressionForm, which runs the evolution to obtain an optimized assignment and shows the progress of the evolution. When the evolution ends, this form runs the simulation of the found assignment. After the computation finishes, it passes the instances of the class SimulationResult to the form SimulationResultForm.

6.3.2.3 Results

The form SimulationResultForm accepts the results of the trivial and optimized assignment. The form writes the results to a file and displays a description of a model. For showing the model description and writing result to a file, the form uses static methods ShowModelDescription and WriteResultToFile of the static class SimulationDescriptor.

6.3.2.4 Animation

On request, SimulationResultForm creates an instance of the form AnimationForm, implementing the interface IElevatorAnimation. This form displays a simple animation of the simulation. The events in a simulation are represented by a list of instances of the class Line. The form uses the static class Painter for

drawing pictures and instances and the class CarState for representing current states of cars.

6.4 File formats

Parameters of the model and passengers in the simulation can be loaded from files. The result of simulations is written to a file. This section describes formats of these files (see section A.1).

Input files (files with parameters, passengers and assignments) can contain comments. A comment begins with the character # and ends at the end of the line. The last parameter in the file cannot be followed by any other characters. Spaces and tabulators are skipped.

6.4.1 Input file with parameters

The file with parameters must contain keywords elevator, building and traffic. The keyword traffic can be followed by one of the keywords up-peak, down-peak, lunch-peak or interfloor, followed by corresponding parameters. Each parameter must be on a separate line. The parameters must be integers and they must be expressed in units used in the simulation. Distances must be indicated in centimetres, time in hundredths of seconds, velocity in cm/s and acceleration and deceleration in cm/s^2. Number of persons on floors are listed from the lowest floor to the highest floor after the height of a floor, each floor on a separate line.

The order of the parameters follows their order in the first form (see Figure 5.1). The keyword elevator is followed by lines containing:

- the number of elevators
- the number of cars in an elevator
- the capacity of a car
- the time required to open the door of a car
- the time required to close the door of a car
- the maximum velocity
- the acceleration
- the deceleration
- the time required to move a car from one shaft to the other shaft
- the height of a car
- the minimum distance

The keyword building is followed by:

- the number of the lowest floor
- the number of the highest floor

- the height of a floor
- the number of persons on each floor from the lowest floor to the highest floor (each floor on separate line)
- the time required for boarding and alighting of a passenger

The keyword traffic can be the last word in the file. Alternatively, it can be followed by exactly one of the keywords up-peak, down-peak, lunch-peak or interfloor. This keyword must be followed by corresponding parameters. The keyword up-peak must be followed by:

- the number of passengers
- the percentage of passengers travelling from the ground floor
- the percentage of passengers in general traffic
- the time period

The keyword down-peak must be followed by:

- the number of passengers
- the percentage of passengers travelling to the ground floor
- the percentage of passengers in general traffic
- the time period

The keyword lunch-peak must be followed by:

- the number of passengers
- the percentage of passengers travelling to the ground floor
- the percentage of passengers travelling to the floor with a canteen
- the number of the floor, where the canteen is situated
- the percentage of passengers in general traffic
- the time period

The keyword interfloor must be followed by:

- number of passengers
- time period

6.4.2 Input file with passengers

In the file with passengers, each passenger must be on a separate line. Each line must contain four integers:

- the identification number of the passenger
- the time of arrival of the passenger in hundredths of seconds
- the origin floor of the passenger
- the target floor of the passenger

These numbers must be separated by spaces or tabulators.

6.4.3 Input file with assignment

The program allows running a simulation with requested assignment of passengers to cars. This assignment is loaded from a file. Each line of this file contains three integers:

- the ID of a passenger
- the ID of the elevator assigned to the passenger
- the ID of the car assigned to the passenger

These numbers must be separated by spaces or tabulators. Each passenger ID must be contained exactly once in the file.

6.4.4 Output file

Results of simulations are written to a file. This file contains:

- average waiting times in the trivial control and the solution found by the genetic algorithm
- the summary of model and evolution parameters
- cars assigned to passengers in the trivial control and the solution of the genetic algorithm
- the waiting times of passengers in the trivial control and the solution of the genetic algorithm
- logs from simulations of the trivial control and of the assignment of passengers to cars found by the genetic algorithm

Conclusion

Efficient elevator system control with short waiting times is an important part of the design of buildings. Apart from conventional elevator system control methods, new methods based on different approaches are researched. These methods include algorithms for elevator system control based for example on linear programming ([17], [18]), genetic algorithms ([19], [20], [21], [22], [26], [28]), genetic programming ([23]), neural networks ([24], [25]), tabu search ([26], [28]) or partical swarm optimization ([27], [28]) and methods of finding the optimal car parking ([14], [15]). Multi-car elevator control must be aware of the movement of each car in the elevator. Circulating multi-car elevator is one of the new types of elevators invented to provide better vertical transportation in high buildings ([6], [4]).

This thesis was focused on optimizing CMCE systems regarding the average waiting time. In the simulations of the trivial control of CMCE, which were presented in this thesis, waiting times decreased with increasing number of cars and sometimes slightly increased with very high number of cars. With small number of cars, waiting times increased with increasing number of passengers. With more cars, waiting times were sometimes slightly better with more passengers. By increasing the capacity of cars, waiting times could be improved, especially with few cars. With more cars, higher car capacity was helpful mainly in the up-peak and down-peak traffic. In most simulations, waiting times were the highest in the down-peak and the lowest in the up-peak.

According to the solutions found by the genetic algorithm, the trivial control, assigning a call to the nearest car, is not optimal for CMCE. In some cases, the genetic algorithm found a solution with the average waiting time lower by at least 40 %. Waiting times in the CMCE system could be reduced by implementing a more efficient algorithm of elevator control.

Within this thesis, we created a computer program including a simulator of CMCE and implementation of a genetic algorithm searching for the most efficient CMCE system control.

Future work could be focused on finding a computationally less expensive way of evaluating CMCE control and improving the algorithm of optimization. The optimization method could also include elimination of extremely long waiting times, which is not guaranteed by using solely the average waiting time as the optimization criterion [16]. We can observe this fact on our results presented in chapter 3, where waiting times of several passengers are usually far higher than the average waiting time (see section A.1). The animation could be improved to display more information about cars.

Bibliography

[1] G. C. Barney and Lutfi Al-Sharif. *Elevator traffic handbook: theory and practice*. Second edition. London: Routledge, Taylor & Francis Group, London, 2016.

[2] Wikipedia contributors. Escalator — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Escalator&oldid=840948517`, 2018. Online. Accessed 28 April 2018.

[3] Wikipedia contributors. Elevator — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Elevator&oldid=841200623`, 2018. Online. Accessed 16 May 2018.

[4] Wikipedia contributors. Paternoster — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Paternoster&oldid=842636857`, 2018. Online. Accessed 28 April 2018.

[5] Wikipedie. Výtah — wikipedie: Otevřená encyklopedie. `https://cs.wikipedia.org/w/index.php?title=V%C3%BDtah&oldid=15747451`, 2018. Online. Accessed 8 June 2018.

[6] Hitachi. `http://www.hitachi.co.jp/New/cnews/month/2006/03/0301.html`, March 2006. Online. Accessed 28 April 2018.

[7] Wikipedia contributors. Online algorithm — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Online_algorithm&oldid=827449285`, 2018. Online. Accessed 16 May 2018.

[8] Randy L. Haupt and Sue Ellen Haupt. *Practical Genetic Algorithms*. Second edition. John Wiley & Sons, Inc., Hoboken, New Jersey, 2004.

[9] CTBUH. CTBUH Tall Building Height Calculator. `http://www.ctbuh.org/LinkClick.aspx?fileticket=s4SVxV%2fruCM%3d&tabid=1007&language=en-GB`. Online. Accessed 5 June 2018.

[10] KONE. Elevator solutions. `https://www.kone.my/new-buildings/elevators-lifts`. Online. Accessed 5 June 2018.

[11] James W. Fortune. How Fast Should Tall Building Elevators Go? *CTBUH Journal*, (3), 2016.

[12] Nationwide Lifts. Elevator Dimensions. `http://www.home-elevator.net/info-elevator-dimensions.php`. Online. Accessed 5 June 2018.

[13] Yuki Kuroda, Mitsuru Nakata, and Shingo Yamaguchi. A Problem and Its Solution for Multi-Car Elevator Group Control. In *ITC-CSCC: International Technical Conference on Circuits/Systems, Computers and Communications*, pages 693–696, 2008.

[14] Alex Valdivielso, Toshiyuki Miyamoto, and Sadatoshi Kumagai. Multi-Car Elevator Group Control: Schedule Completion Time Optimization Algorithm with Synchronized Schedule Direction and Service Zone Coverage Oriented Parking Strategies. In *ITC-CSCC: International Technical Conference on Circuits/Systems, Computers and Communications*, pages 689–692, July 2008.

[15] Matthew Brand and Daniel Nikovski. Optimal parking in group elevator control. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 1002–1008, April 2004.

[16] Matthew Brand and Daniel Nikovski. Risk-Averse Group Elevator Scheduling. In *14th International Congress on Vertical Transportation Technologies*, pages 2066–2070, April 2004.

[17] Srikumar Ramalingam, Arvind U. Raghunathan, and Daniel Nikovski. Submodular Function Maximization for Group Elevator Scheduling. In *ICAPS International Conference on Automated Planning and Scheduling*, pages 233–241, 2017.

[18] Mirko Ruokokoski, Harri Ehtamo, and Panos M. Pardalos. Elevator dispatching problem: a mixed integer linear programming formulation and polyhedral results. *Journal of Combinatorial Optimization*, 29(4):750–780, May 2015.

[19] P. Cortes, J. Larrañeta, and L. Onieva. A genetic Algorithm for Controlling Elevator Group Systems. *Lecture Notes in Computer Science: Artificial Neural Nets. Problem Solving Methods*, 2687:313–320, June 2003.

[20] Berna Bolat, Pablo Cortés, Ersun Yalçin, and Mustafa Alişverişçi. Optimal Car Dispatching For Elevator Groups Using Genetic Algorithms. *Intelligent Automation & Soft Computing*, 16:89–99, 2010.

[21] Mehmet Baygin, Dilbirin Orhan, Orhan Yaman, and Mehmet Karakose. A New Real Time Control Approach for Time Efficiency in Group Elevator Control System. *International Journal of Intelligent Systems and Applications in Engineering*, 4(Special Issue-1):211–215, September 2016.

[22] Jayanta K. Debnath and Gursel Serpen. Real-time optimal scheduling of a group of elevators in a multi-story robotic fully-automated parking structure. *Procedia Computer Science*, 61:507–514, 2015.

[23] Lu Yu, Jin Zhou, Shingo Mabu, Kotaro Hirasawa, Jinglu Hu, and Sandor Markon. Double-deck elevator group supervisory control system using genetic network programming with ant colony optimization with evaporation. *JACIII: Journal of Advanced Computational Intelligence and Intelligent Informatics*, 11:1149–1158, 01 2007.

[24] C. Erdem Imrak and Mustafa Ozkırım. The modelling and simulation of elevator group control sytems for public service buildings. *IFAC Proceedings Volumes*, 36(7):145–150, 2003.

[25] Le-fu Cui. An elevator intelligent scheduling method using neural network control. *International Journal of Digital Content Technology and Its Applications*, 7(3):174, 2013.

[26] Berna Bolat and Pablo Cortés. Genetic and tabu search approaches for optimizing the hall call—car allocation problem in elevator group systems. *Applied Soft Computing*, 11(2):1792–1800, 2011.

[27] Berna Bolat, Oğuz Altun, and Pablo Cortés. A particle swarm optimization algorithm for optimal car-call allocation in elevator group control systems. *Applied Soft Computing*, 13(5):2633–2642, 2013.

[28] Berna Bolat, Oğuz Altun, Pablo Cortes, Yunus Emre Yildiz, and Ali Osman Topal. A Comparison of Metaheuristics for the Allocation of Elevators to Calls in Buildings. *Journal of Polytechnic*, 20(3):519–529, 2017.

[29] Microsoft. Microsoft Developer Network. `https://msdn.microsoft.com`, 2018. Online. Accessed 24 May 2018.

A. Attachments

A.1 Input and output files

Attachments contain examples of files with model parameters (Parameters.txt), passengers (Passengers.txt), assignment of passengers to cars (Assignment.txt) and results (Result.txt). Files with parameters and passengers are in the format valid for loading the data to the program. Examples of files are located in the folder Files.

In the subfolder Passengers of the folder Files, there are files with passengers used in simulations in chapter 3. In the subfolder Results, there are output files of examples presented in subsection 3.1.4 and subsection 3.2.4.

A.2 Source codes

Attachments contain source files of the program located in the folder Source codes.

A.3 Program documentation

Attachments contain the HTML documentation of the program built from the XML documentation. The introductory page can be displayed by opening the file index.html. The documentation is located in the folder Help.