Felicitas Fabricius, BEng

ASP-based Task Scheduling for Industrial Transport Robots

Master Thesis

to achieve the university degree of

Diplom-Ingenieurin

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Assoc. Prof. Dipl.-Ing. Dr.techn. Gerald Steinbauer

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl.-Ing. Dr. techn. Wolfgang Slany

Graz, September 2019

Affidavit

18.09.2019	Felicitas Fabricius
Date	Signature

Acknowledgement

This thesis could not have been realized without great mental and practical support.

First, I want to thank my two supervisors Maximilian Selmair and Michael Reip for their great support and guidance during the entire project.

I would like to thank my two academic supervisors: Prof. Gerald Steinbauer for his support in various academic topics over the last two years and his guidance and useful remarks for this research. Prof. Gebser for the enthusiastic encouragement and help in ASP-related issues.

In addition, a thank to my colleagues at BMW and incubed IT. They provided me a pleasant atmosphere to work on my project. My special gratitude belongs to the Fluffy-team at incubed IT: Their enthusiasm and immediate encouragement for any of my belongings and questions was a great support.

Finally, I must express my very profound gratitude to my family who supported me through my entire study. Thanks to the loving support and faith of my parents. Thank you, Josephine, for proving me a pleasant accommodation during my time in Regensburg. And finally thanks to Jonas for his continuous support and calming my nerves in stressful situations.

Abstract

The number of robots used in industrial logistics is increasing every year. Therefore, solving the assignment problem of different robots to given transport orders and charging stations considering a variety of constraints (for example higher or lower transport capacity) becomes more complex. The companies BMW and incubed IT currently use imperative methods to describe the related constraints and solve the assignment problem considering given optimality criteria. As the number of constraints is increasing, the imperative solutions become more complex and harder to maintain. Moreover, the solving of the assignment problem does not scale well due to the rising number of considered robots.
In this thesis the assignment problem at BMW and incubed IT is modeled and solved using the alternative declarative method Answer Set Programming (ASP).
In a first step parts of the overall assignment problem where the ASP-based solving could give a benefit were identified and the related assignment strategies have been encoded in ASP. To improve the performance of the initial ASP-based solving we need to adapt the problem encoding. It appeared that encodings where the optimization strategy is represented using constraints perform better than encodings with dedicated optimization criteria. For a further performance improvement different heuristics for the solving approach, provided by the ASP solver Clasp, were evaluated. In order to handle the task assignment problems of both companies different optimal solving approaches were obtained. While at BMW splitting-based multithreading with a branch-and-bound-based optimization strategy was the best performing approach, at incubed IT compete-based multithreading with a Vsids-Heuristics for solving showed the best performance. The runtime and quality of the results of the imperative and declarative implementations for both settings were evaluated in extensive simulations. It turned out that for task assignment problems with fewer constraints, like at

BMW, imperative programs had a better runtime. However, the ASP-based implementation at BMW gave a better quality of the results as a more complex solving algorithm was encoded. For task assignment problems with many constraints, like at incubed IT, the declarative implementation showed a better performance. For very highly scaled problem instances the ASP approach reached its performance limits finding the optimal assignment set. But when using anytime algorithms in ASP a not optimal but still satisfying assignment set was found within milliseconds.

Contents

List of Figures

List of Tables

List of Examples

List of Listings

1. Introduction

These days global markets are broadly established and lead to increased economic and more competitive markets. In order to stay competitive, production plants have to be improved to fulfill the demands for more efficiency, higher flexibility and an increased production throughput.
These improvements do not only affect the various production steps but also the field of intralogistics.
Currently many logistics tasks are performed by man-driven tugger trains and forklifts. Delivering tasks are predefined and handled in a fixed order. Some logistics tasks are undertaken by automated intralogistics robots.
As in the last years technologies for laser scanning and powerful processors improved, and parts became more favorable, autonomous robots are now more and more reliable. In the last few years a couple of different autonomous logistic solutions came up to replace the existing logistics systems. The centralized logistics management can be transformed in a partially decentralized one. While the assignment of orders and management of charging and parking strategies is managed centralized, the routing is taken over decentralized by every robot itself.

1.1. Research Objective

Incubed IT develops software to operate autonomous, self-navigating and co-operative mobile robots. The BMW Group increases the number of autonomous robots in its plants with self-developed robots and software. Both companies are working with a cloud-based Fleet Management System (FMS). The robots navigate and drive decentralized to perform given transport tasks. Orders that have to be delivered and assignments of robots are managed centralized at the FMS. By that the complete in-house delivering

process is taken over by the FMS and autonomous robots.
The intralogistics management at BMW Group and incubed IT is a very high scaled problem with critical constraints that have to be maintained. For example a requested good must be delivered to the production line at BMW before a hard time limit is reached.
At the beginning of the development process both companies worked on small-scaled scenarios. By scaling up the FMS with an increased number of open orders and assignable robots the performance of the assignment algorithm has to be considered more. Additionally, with an increased application field for the autonomous robots the number of rules and constraints for an assignment increase.
In recent research projects declarative programs were successfully used for the solving of assignment problems. The result of these research projects showed a promising approach for the management of the FMS at BMW and incubed IT. In this thesis the declarative method Answer Set Programming will be used in the existing FMS at BMW and incubed IT to model and solve the task assignment problem. The resulting implementation will be evaluated regarding the applicability in the FMS at incubed IT and BMW.

1.2. Contribution

The objective of this master thesis is to evaluate the benefit of using a declarative modeling method and a high-performance solver for the multi-robot assignment problem. Thus, the existing systems at incubed IT and BMW are analyzed. Elements of the system in where declarative methods could provide a benefit are identified. After evaluating the possibilities of ASP to replace these parts of the system, declarative programs are set up for BMW and incubed IT. The focus is laid on an easy to understand and maintainable representation and the performance of the solving. Therefore different solving approaches will be analyzed. The new program will than be compared to the existing implementation in regards of the runtime and the quality of the provided solutions. The result will be used to give an outlook of the potential and limits of ASP as an modeling paradigm for assignment problems.

1.3. Document Structure

The thesis focusses on the improvement of the current FMS of the autonomous intralogistics transport robots at incubed IT and BMW Group. One main challenge is thereby the replacement of the current task assignment strategy.
In chapter 2 of this thesis different task assignment problems are introduced. The general definitions are extended by problem definitions focusing on the assignment of robots. Moreover, the imperative and declarative programming paradigms are introduced and compared. In section 2.3 the declarative method Answer Set Programming is presented together with information about the solving approaches which can be used to solve the given problem. In this thesis Potassco, an answer set solving collection is used. This collection is introduced as well as its input language is explained in detail.
In chapter 3 related research is discussed. Different strategies for an optimal task assignment are shown and application areas of ASP, where declarative methods provide a benefit are listed as well.
In chapter 4 the intralogistics strategies at BMW and incubed IT are presented.
Based on the provided related research and the introduction in the existing intralogistics strategies at BMW and incubed IT a motivation why ASP could provide a benefit in the current system is given in chapter 5.
In chapter 6 an ASP-based implementation of the task assignment for the existing fleet management systems (FMS) is presented. ASP programs are implemented in areas in the FMSs, where declarative programs can be a benefit. Steps to increase the performance are explained and the integration of the ASP encoding in the existing FMSs is highlighted.
In chapter 7 the results of the implemented solution using ASP for the assignment problems are shown. The performance of the modeling and the quality of the resulting assignments are compared with the existing imperative implementations. The implementation effort of the new ASP encoding is analysed as well as the effort of the integration in the overall system.
In chapter 8 the results are discussed, and in chapter 9 an outlook is provided.

2. Prerequisites

This chapter describes some prerequisites that are relevant for this thesis. After a general description of the task assignment problem approaches for solving one-dimensional and multi-dimensional task assignment problems are introduced. Furthermore, an outlook will be given on solutions to robot-specific task assignment problems.
In the second section languages used to describe task assignment problems are stated and compared. In the third section one of these languages, namely the declarative method Answer Set Programming, will be explained in detail.

2.1. Task Assignment Problem

Based on a general description provided by [9, p. 1] the Task Assignment Problem (TAP) can be described with the following example:
A set of n jobs (i = 1,. . . , n) and a set of n workers (j = 1,. . . , n) are given. Every worker can do every job in a known duration. This time can be defined individually for every job-worker combination. The task assignment problem focuses on the optimal assignment of jobs to workers to fulfill an optimization criterion, like the minimization of the production time.
More formally described is a set of pairs of persons and objects defined as an assignment. Thereby every person i can be only assigned to one object j and every object j can be assigned to only one person [7, p. 9].
To solve TAPs optimally a rating matrix $R = (r_{ij})\ , i, j \in \mathbb{N}_0$ is introduced. In the matrix every job-worker pair (i,j) is graded with a rating element r_{ij}, representing the assignment costs with positive integers [45]. Such a rating

element can, for example, be the time a worker j needs to finish the job i, like it is in the example above. Given the rating matrix

$$R = \begin{bmatrix} r_{1,1} & \cdots & r_{n,1} \\ \vdots & \ddots & \vdots \\ r_{1,n} & \cdots & r_{n,n} \end{bmatrix} \tag{2.1}$$

The TAP is completely solved by choosing a set of n assignments. To solve it optimally, the set has to be chosen in a way that the sum of rating elements for all n assignments is optimized.
To solve the optimization problem, an objective function is introduced. This function is, depending on the optimization goal, minimized or maximized and given by the equation

$$\sum_{i=1}^{n} r_{i\phi(i)} \tag{2.2}$$

where $\phi(i)$ is a bijective mapping of the two sets jobs and workers with n elements [9].
The complexity of the assignment problem depends on the number of optimization criteria that affect the dimension of the rating matrix and by that the dimension of corresponding objective function. The one-dimensional assignment problem has been introduced with the example at the beginning of this chapter: Given a set of n items and n other items, the optimal solution can be found using a one-dimensional objective function (see equation 2.2). Problems in this dimension can be solved by polynomial-time algorithms, like the Hungarian Method (see section 2.1.1).
Problems with more than one optimization criterion are described by multi-dimensional objective functions and called multi-dimensional assignments problems. Example for such a multi-dimensional assignment problem, the *timetabling problem*, is provided by [9, p. 8]: Given a set of n courses, n time slots and n available rooms, a three-dimensional rating matrix is set up.

$$R = r_{ijk} \ , i, j, k, r_{ijk} \in \mathbb{N}_0 \tag{2.3}$$

Solving such multi-dimensional assignment problems is NP-hard.
In the next chapters approaches for solving the one-dimensional and multi-dimensional assignment problem are presented. Another focus is also laid on the specialised task assignment of industrial transport robots.

2.1.1. One-Dimensional Task Assignment Problem

To find the optimal solution for one-dimensional task assignment problems a various number of solution approaches is introduced in literature. Some of the most popular once are introduced in this work.

Solving Approach: Integer Linear Programming

One approach is the Integer Linear Programming [49]. To solve the TAP, the objective function of the task assignment problem (see equation 2.2) is minimized, leading to the *linear sum assignment problem* [9, p.5]

$$min_{\phi \in S_n} \sum_{i=1}^{n} r_{i\phi(i)} \tag{2.4}$$

where S_n is the set of all permutations.
In order to solve this problem for the one-dimensional assignment problem, two sets and the rating matrix $R = r_{ij}$ with

$$(i, j = 1, 2, ..., n). \tag{2.5}$$

are given and an additional binary matrix X is introduced [9]:

$$X = (x_{ij}) = \begin{cases} 1 & \text{if } j = \phi(i), \\ 0 & \text{otherwise,} \end{cases} \tag{2.6}$$

Using this binary matrix, the TAP can be modelled as followed [49]:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} r_{ij} \tag{2.7}$$

$$s.t. \sum_{i=1}^{n} = x_{ij} = 1, \tag{2.8}$$

$$\sum_{j=1}^{n} = x_{ij} = 1. \tag{2.9}$$

For a better understanding a given TAP will be solved using this introduced model. The problem consists of workers w_i $(i = 1, ..., n)$ and jobs t_j $(j = 1, ..., n)$. The rating matrix R represents the time a worker needs to finish a job. The aim of the TAP is to reduce the total sum of working times required by every worker to finish his job.
In equation 2.7 the sum of rating elements r_{ij} of all assigned sets (worker - jobs) is taken and minimized. To ensure the use of only assigned rating elements equation 2.6 sets x_{ij} to zero for all unassigned rows and columns. Equation 2.8 and equation 2.9 are additional constraints. Equation 2.8 is fulfilled if and only if every task j is be taken over by exactly one worker. Equation 2.9 states if every worker is assigned to only one and not multiple or no jobs.
This model is the first approach to solve the TAP but has high computation costs. Numerous studies introduced solution approaches for the TAP with decreased computational costs, for example the Hungarian Method [45] or the Auction Algorithm [7].

Solving Approach: Hungarian Method

The Hungarian Method, first introduced by J. W. Kuhn [45], is a primal-dual solution approach for the TAP. To solve the TAP with a given nxn rating matrix $\mathbf{R}$ following steps, stated by M. Flood [24], have to be executed:

1. The smallest element of $\mathbf{R}$ is subtracted from all other elements in the rating matrix. The resulting matrix $\mathbf{R_1}$ contains positive elements and at least one null element.
2. A minimal number of lines is drawn through the rows and columns of the matrix $\mathbf{R_1}$, covering all null elements. In case the number of lines equals the size n of the rating matrix a solution is found: There are no two null elements in the same line, and the positions of all null elements in the rating matrix $\mathbf{R_1}$ constitutes the solution.
3. In case the number of lines is smaller than n, the optimal allocation is not yet found. The smallest element s_1 in the rating matrix that is not covered by a line is taken and added to all elements covered by a line and subtracted from all elements of $\mathbf{R_1}$. The addition is necessary to avoid negative costs in the matrix.

4. Step 2 and 3 of the algorithm are repeated until the number of lines of step 2 equals the number of null elements.

A practical example of the Hungarian Method is given in figure 2.1. In there a 3 x 3 rating matrix is given, and the optimal assignments of jobs j to workers i have to be found.

r_{ij}	job 1	job 2	job 3
worker 1	10	20	70
worker 2	35	25	35
worker 3	70	10	50

(a) Initial rating matrix, smallest elements are marked blue

r_{ij}	job 1	job 2	job 3
worker 1	0	10	60
worker 2	25	15	25
worker 3	60	0	40

(b) Step 1: The smallest element ($r_{ij} = 10$) is subtracted from all elements

r_{ij}	job 1	job 2	job 3
worker 1	0	10	60
worker 2	25	15	25
worker 3	60	0	40

(c) Step 2/1: Lines are drawn through rows and columns covering all null elements

r_{ij}	job 1	job 2	job 3
worker 1	0	10	35
worker 2	25	15	0
worker 3	60	0	15

(d) Step 3/1: Smallest uncovered element of rating matrix ($r_{ij} = 25$) is added to all line-covered elements and subtracted from all elements

r_{ij}	job 1	job 2	job 3
worker 1	0	10	35
worker 2	25	15	0
worker 3	60	0	15

(e) Step 2/2: Drawing lines through rows and columns covering all null elements. Number of lines equals matrix dimension n, solution is found

Figure 2.1.: Exemplary solving approach for a TAP using the Hungarian Method

After applying the just introduced algorithm steps for the Hungarian Method the optimal assignment is shown in table 2.1.
Using the Hungarian Method to solve the TAP instead to the computational approach shown in equation 2.4 the runtime can be reduced to $O(n^4)$ using Floods approach. The currently fastest algorithm for the Hungarian Method has a runtime of $O(n^3)$ [9].

Worker	Assigned Job	Costs
worker 1	job 1	10
worker 2	job 3	10
worker 3	job 2	35

Table 2.1.: Optimal assignment for the assignment problem in figure 2.1

Solving Approach: Auction Algorithm

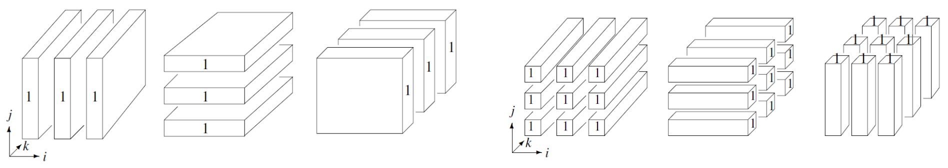
A second approach for solving the TAP is the Auction Algorithm. This method is first introduced by Bertsekas [7] in 1988 and an consists of two phases for every iteration step. These phases are the following:

- **Biding Phase:** For every unassigned worker i a job j is found which offers the maximum value $a_{ij} - p_j$, where a_{ij} is the integer value of worker assignment and p_j is the price for the worker to be assigned to the job.
  A biding value γ_i is computed using the following formula:
  $$\gamma_i = v_i - w_i \tag{2.10}$$
  where v_i is the best object value
  $$v_i = max_{j_v \in A(i)}(a_{ij_v} - p_{j_v}) \tag{2.11}$$
  and w_i is the second-best object value
  $$w_i = max_{j_w \in A(i), j_w \neq j_v}(a_{ij_w} - p_{j_w}) \tag{2.12}$$
  If there exists only one object in A(i) and w_i can't be defined using the equation 2.12 w_i is set to $-\infty$.
- **Assignment Phase:** For all jobs j that are set to be the best object for a worker the bidding values are compared. The highest bidder is chosen and the corresponding job is assigned to the bidding worker.

The algorithm is continued until all workers are assigned.
The runtime of this algorithm highly depends on the input data. The worst case runtime $O(n^3)$ is the same as for the Hungarian Method [9].

2.1.2. Multi-Dimensional Assignment Problem

Multi-dimensional Assignment Problems are an extension of the generalized, one-dimensional task assignment problem. Such problems are described with more than one constraint.
Starting at a one-dimensional TAP and adding an additional optimisation parameter, the TAP becomes a multi-dimensional Assignment Problem. 3-dimensional Assignment Problems can be grouped in two models, the axial and planar 3-index assignment problem [9]. The graphical presentation of the constraints of both problems can be seen in figure 2.2. In both images the three constraints r_i, r_j and r_k (with $i, j, k = 1, 2, ..., n$) are represented by the coordinate axes. Whereas in axial problems only two permutations are given in planar systems $n > 2$ permutations have to be considered.



(a) axial 3-index assignment problem (b) planar 3-index assignment problem

Figure 2.2.: Graphical representation of constraints for 3-index assignment problems [9, p. 306]

The axial 3-index assignment problem has the following rating matrix [9]:

$$R_{3x3} = r_{ijk}, i, j, k, r_{ijk} \in \mathbb{N} \tag{2.13}$$

The objective function with two permutations ϕ and ψ can be stated as

$$min_{\phi,\psi \in S_n} \sum_{i=1}^{n} r_{i\phi(i)\psi(i)} \tag{2.14}$$

where S_n is the set of all permutations [9].
The integer linear program of the axial 3-index assignment problem is defined as [9]:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} r_{ijk} x_{ijk} \tag{2.15}$$

$$s.t. \sum_{j=1}^{n} \sum_{k=1}^{n} = x_{ijk} = 1, (i = 1, 2, ..., n) \tag{2.16}$$

$$\sum_{i=1}^{n} \sum_{k=1}^{n} = x_{ijk} = 1, (j = 1, 2, ..., n) \tag{2.17}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} = x_{ijk} = 1, (k = 1, 2, ..., n) \tag{2.18}$$

The solving of this problem is NP-hard.

2.1.3. Multi-Robot Task Allocation

In this section an overview of multi-robot task allocation, a special form of the multi-dimensional assignment problem, is provided.
Aim of the multi-robot task allocation (MRTA) is an optimal assignment of a set of robots to a set of tasks under consideration of robot-specific constraints. The complexity and appropriate solving approaches of the task assignment problem depend on these constraints. The MRTA can be visualized in 3 dimensions, representing the types in which the MRTA is classified. Looking at figure 2.3 these types are the following [40]:

- Robot Type: **Single-Task (ST)** robots execute one task at a time, **Multi-Task (MT)** robots handle multiple tasks simultaneously.
- Task Type: **Single-Robot (SR)** tasks require one robot to be finished, **Multi-Robot (MR)** tasks need multiple robots to be performed.
- Allocation Type: If there is only instantaneous allocation of tasks to robots possible for the given set of robots, tasks and environment, the Allocation Type is called **Instantaneous Assignment (IA)**. A **Time-extended Assignment (TA)** is an allocation type where current and future tasks are assigned to robots.

Depending on the combinations of types, the solving of MRTA problems is more or less difficult. In the following different MRTA-dimensions with suitable solution approaches are introduced.

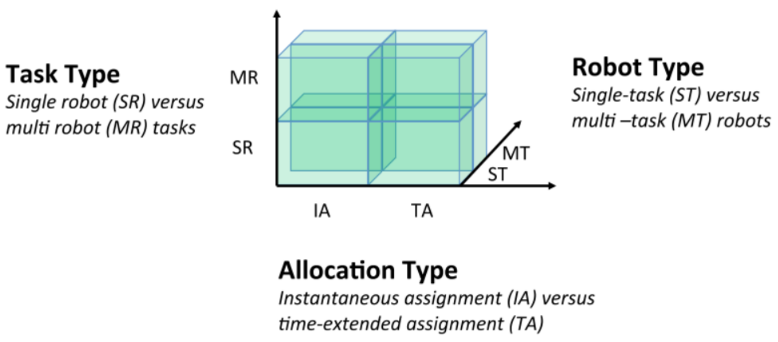

Task Type
Single robot (SR) versus multi robot (MR) tasks
MR
SR
IA
TA
MT
ST
Robot Type
Single-task (ST) versus multi –task (MT) robots
Allocation Type
Instantaneous assignment (IA) versus time-extended assignment (TA)

Figure 2.3.: Multi-robot task allocation dimensions [44]

ST-SR-IA This is a straightforward multi-task allocation and can be broken down to the 1-dimensional task assignment problem (see section 2.1.1). Instead of workers to jobs robots are assigned to tasks. Multiple solving approaches, for example the Hungarian Method or the Auction Algorithm, exist for the 1-dimensional TAP and have been introduced prior. The problem can be solved using one of the two algorithms in $O(n^3)$ time [40].

ST-MR-IA In this scenario, a robot cannot finish a task alone but needs one or more other robots to complete this task. It can be best solved using heuristic approaches, as otherwise the calculation is NP-hard [40].

MT-SR-IA In this problem one robot executes multiple tasks simultaneously. From a mathematical point of view, the problem can be solved in the same way as the ST-MR-IA problem: The multiple components element *MR* of the ST-MR-IA problem is replaced by the multiple components element *MT* of the MT-SR-IA problem [44].

MT-MR-IA To finish a task a set of robots has to work on it. Additionally, every robot is capable of working on more than one task [44]. Solving this problem is NP-hard. As the problem structure is equal to the set covering problem, it can be solved with the same algorithms, for example the Greedy approximation algorithm [40].

ST-SR-TA This scenario describes the same problem as *ST-SR-IA* with an additional planning component. To determine the schedule for every robot is an NP-hard problem [44]. For a reduction of the computational costs Gerkey [40] mentions another solution approach: First, every robot is assigned to one task. After finishing the first task, every robot is assigned to one of the remaining tasks until all tasks are finish. With that, the runtime reduces to the ST-SR-IA runtime of $O(n^3)$.

ST-MR-TA This scenario combines the multi-robot task type with a scheduling component. Solving this problem is NP-hard. To reduce the computational costs the ST-MR-TA problem can be solved similarly to the *ST-MR-IA* problem. After assigning all robots to a task, the remaining tasks can be assigned in an online-fashion as soon as the robots finished the previous task [40].

MT-SR-TA In this scenario a scheduling problem for multi-task robots and single-robot tasks has to be solved. Reversing robots and tasks the *ST-MR-TA* problem represents the problem. Due to that this problem can be solved using the same approach [40].

MT-MR-TA This problem can be seen as an extension of the MT-MR-IA problem: Every robot can execute multiple tasks in parallel, and a subset of robots is needed to fulfill one task. A scheduled assignment has to be found for this scenario. This problem is NP-hard and not even heuristic approaches are capable of solving this problem. As already the *MT-MR-IA* problem is NP-hard, not even the avoidance of planning by an online-fashioned assignment can reduce the computational costs [40].

As seen, the multi-robot task assignment can be broken down into eight scenarios. Less complex MRTA problems, like *ST-SR-IA* and *ST-SR-TA*, can be solved in polynomial time using approaches like the linear sum assignment problem (see equation 2.4). The solving of more complex scenarios is NP-hard. To reduce the average computation costs for these scenarios heuristic approaches are considered. A further reduce can be achieved using an online-fashioned assignment for problems in the allocation type dimension of time extended assignments (TA).

2.2. Programming Paradigms

To solve the just introduced TAP different programming languages can be used. The programming languages are split into imperative and declarative paradigms. The difference between the two programming paradigms is the general handling of the problem. Using imperative methods in the code, it is described how to solve the problem with search-based algorithms like the Hungarian Method or the Auction Algorithm. Using declarative methods it is described what the problem is, but the solving is taken over by underlying heuristics and solving algorithms [5, 31].
The benefits and challenges of both paradigms are described further in the next sections.

2.2.1. Imperative Methods

Imperative languages (lat. imperare = to command) are instruction-oriented languages. The fundamental strategy is the assignment of values to variables. Problems are solved by a sequence of instructions. The order of the execution of instructions is relevant for a successful problem resolution [23].
The following code example is written in the imperative programming language C#. In there a list of integers is provided. All numbers that are below the maximum value 5 are saved in a new list. The foreach-statement is a sequence of instructions, in where every value of the list is compared to the value 5 and if the value is smaller than 5 it is assigned to the variable *output*.

Listing 2.1: C# Algorithm for sublist assembling

```
List<int> input = new List<int> { 2,4,6,8,10 };
List<int> output = new List<int> {};
foreach(var val in input){
  if(val < 5)
    output.Add(val);
}
```

Imperative languages are split in two subsets: The object-oriented and procedural languages. Popular procedural languages are for example C and PASCAL. Object-oriented languages are next to other C# and Java [59].

2.2.2. Declarative Methods

Declarative languages (Latin declarare = to disclose) are languages oriented on the describing of problems. In opposite to imperative languages they contain no instructions but only mathematics functions [23]. Problems are characterized by describing the environment and specific conditions of the solution with expressions and rules. The solving process of the problem is taken over by a specific solver that runs in the background [31].
A small code example, written in the Answer Set Programming (ASP) syntax, is given below. As in the code example 2.1 a list of input values is given. All values below the maximum number 5 are saved in a new list. In this code no sequence of instructions is required to solve the problem. Only a problem description is given in the second row of the code snippet. Internal solvers overtake the problem-solving process.

Listing 2.2: ASP Algorithm for sublist assembling

```
input(2;4;6;8;10).
output(I) :- input(I), I < 5.
```

Popular declarative languages are ASP, PROLOG and CSP [59].

2.2.3. Comparison of the Methods

Declarative programs describe what the problem is, whereas imperative programs describe how to solve the problem. Therefore the coding structure of declarative methods is usually shorter and more concise than the coding structure of imperative methods. That leads to programs that can be set up and extended faster than imperative programs. By that they are easier to understand and errors in the code can be found faster, as not as much code has to be reviewed. An exception are problems that are difficult to be described by rules and expressions, as for these problems the complexity of the declarative coding structure can become larger than the imperative one for describing how the problem is solved.
Task assignment problems consist of only one rule, the assignment of every worker to a job. Possible solutions are often restricted by some expressions, like an optimization condition or the consideration of an environment in where some workers can do only some jobs. These problems can be described with a lean declarative program.
In the previous chapter different approaches for solving the task assignment problem are introduced. Depending on the problem environment the solving of the problem with imperative methods, using the introduced approaches, can be NP-hard. Using declarative methods the solving process is taken over by internal solvers. Declarative programs with powerful solvers running in the background are a promising approach to solve task assignment problems [3, 27, 59].

2.3. Introduction to Answer Set Programming

The difference between imperative and declarative programming languages has just been introduced. A well-known declarative language is *Answer Set Programming* (ASP), often also called *AnsProlog* or simply *A-Prolog* [5]. This logic programming language is particularly suitable for solving

> knowledge-intense combinatorial (optimization) problems [43, p. 1].

The language is based on answer set semantics, as the name *Answer Set Programming* suggests. The solution of a given problem is provided as an output of answer sets [8].

2.3.1. Motivation for ASP

The motivation for the use of ASP is the simple expression of search and optimization problems with a collection of rules in logic format. Given an input that describes the initial problem using constraints and rules, the problem is solved in two steps [31]:

1. A grounder is applied to the initial program and turns it into a finite propositional form.
2. A solver uses this propositional form to compute solutions for the problem and gives back the solutions in form of answer sets.

The specific characteristic of declarative methods, to describe what the problem is, is valid for ASP as well. As usual for declarative languages, the user describes the problem but does not solve it. Instead different solvers can be applied to find the optimal answer set for the given problem.
A remark will be given on the general layout of ASP, the grounder and solver respectively. It will be looked into the language syntax and remark on integrated development environments and application programming interfaces will be provided.

2.3.2. Differentiation to Other Declarative Languages

ASP is only one of many declarative methods. In this section the main differences and application fields of ASP compared to the two other popular constraint solvers Prolog and CSP are shown.

ASP and Prolog

Prolog is one of the most common logic programming languages. The syntax of Prolog and ASP are very similar, thereby ASP is often also called *AnsProlog* or *AProlog*, short for Answer-Set-Prolog [5].
Comparing Prolog with ASP some major differences can be observed. In ASP database techniques lead to a basic data structure of tuples and terms. In Prolog nested terms and variables via unification are the basic data structure. Comparing the solution encoding, Prolog computes solutions using query answering, whereas in ASP the solution is encoded in answer sets. In other words, Prolog does proof finding, ASP does model finding. Another major difference between both logic programs is the matter of ordering. In ASP the ordering of literals in the rule bodies does not matter. In Prolog a modified order of literals cause different solving steps. This leads to the ability of the user to control the program execution and can define how a solution can be found. In ASP the programmer is not allowed and able to control the solution search [8, 31].
Some other difference can be found observing the structure of rules. In ASP disjunctions in the head of rules are allowed, in Prolog they are not. The ordering of literals in the body of a rule does not influence the solving of the ASP, but in Prolog the literals are processed from the left to the right and by that the ordering takes influence in the solution finding process [5].
Summarising the above it can be said, that programming is easier with ASP than Prolog, as the order of literals and constraints does not affect the solving process. By that the programmer does not take influence on the solving structure. A language that is used to solve assignment problems must not only return the proof that an assignment is possible but also the corresponding answer set. As Prolog is proof finding, the ASP as model finding algorithm fits the needed requirements for assignment problems better.

ASP and CSP

For Constraint Satisfaction Problems (CSP) the language syntax is the following: Given are a set of variables and a set of possible values for each variable. The values are often called the domain of a variable. Two types of

constraints are set up based on the variables and values. One constraint type defines allowed combinations of variables and values. The other constraint type defines the forbidden combinations of variables and values [5].
For a CSP answer sets are generated by assigning variables to finite domains of values and solving the CSP over these finite value domains. In ASP a grounder compiles the program in a propositional form. A solver is finding a problem solution over binary domains, where all variables are represented as propositional atoms. The solving over a binary domain in ASP in opposite to the solving over the finite value domain in CSP leads to a more efficient solving process in ASP.
This more efficient solving process is influenced additional on the language syntax. The CSP supports a high-level language following mathematical notations. These notations come with many different sets, functions, and relation theorems. On the opposite ASP consist mainly of natural language statements and definitions with some additional syntactic sugar, what makes this syntax easier to understand and allows an easier modeling [8].
For problems that required mathematical notations the use of the CSP instead of ASP could give an benefit, but these notations are rarely used for multi-robot task allocation problems. The runtime and required computational costs for the solving of CSPs over finite value domains is expected to be worse than the runtime and costs for the solving of ASP programs over binary domains. To solve multi-robot task assignment problems ASP is highly likely the more suitable declarative method than CSP.

2.3.3. Solving Architecture

In this chapter the solving architecture of Answer Set Programming is introduced.
The basic solving process of ASP is shown in figure 2.4. A given problem is first modelled in a logic program with stable model semantics (see section 2.3.5). For the solving process in a first step a grounder encodes the given logic program in a propositional form. A solver uses this propositional program to compute stable models. After the solving process is finished, the stable models are used to decode the solution of the given problem [31].

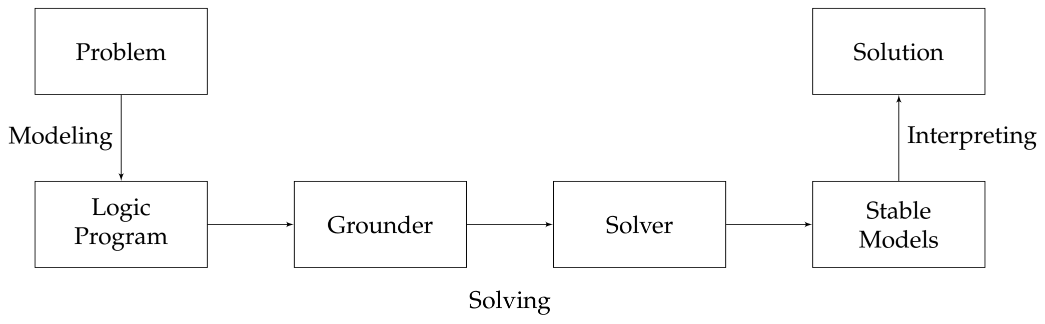


Figure 2.4.: The solving process of ASP [31, p.3]

Grounder

The grounder translates a logic program into a variable-free Boolean program. The requirement therefore is a logic program with stable model semantics. The size of the given logic program is often expanded exponentially to the size of the grounded program. As the grounded program processes the problem sets to the solver the performance of the solving process depends significantly on the grounder [36].
Mostly the grounders Lparse [56], DLV [46] and Gringo [28] are mentioned and used in research projects, but recently another promising grounder, named I-DLV [12], was released.

Lparse Lparse is a very early grounder and is based on the input of ω-restricted programs [57].
The ω-restriction has impact on the supported language syntax. Every variable that is used in a rule has to occur in a body literal [30]. This literal has to be positive and the corresponding predicate must not be recursive mutual and is not defined with choice rules [17, p. 88]. Lparse supports priorities to weight constraints. This priority is set depending on the input order of the constraints but cannot be chosen variably.
Lparse supports basic literals, the extended literals weight and constraint rules and the additional conditional literal [56]. The output-language of Lparse is Smodels, an intermediate format which can pass the program from the grounder to the solver and is supported by many different solvers [31]. Smodels has a *Domain-restricted* rule syntax:

> Every variable in a rule must occur in a positive *Domain predicate*, which are predicates not defined via negative recursion or using 'choice rules' [17, p. 88].

DLV DLV is the only grounder presented in this work running with only one specific solver, namely the DLV solver. The use of the grounder DLV with another solver is not possible [25]. The input program has to be a logic program with stable model semantics but must not be ω-restricted as required for the Lparse grounder [31].
The language syntax of DLV has in comparison to Lparse some improvements: This grounder is the first introduced grounder supporting disjunctive headers [42]. For all variables that are used in rules it is defined that they have to occur in a body literal. In distinction from Lparse the literal does not have to be positive. Furthermore, the predicate of the body has not to be a built-in comparison predicate [17, p. 88].
Like Lparse the DLV grounder supports priorities. Therefore, weak constraints are used [25].

Gringo Gringo is a development based on the Lparse grounder. Both use the same output language Smodels, but Gringo does not require restricted input programs as Lparse does. Like for the DLV grounder, the program has to be a logic program with stable model semantics to be solved [31].
As Gringo is based on Lparse the language semantics of this grounder is based on the one of Lparse [31]. Both follow the ASP-Core-2 language (see [11]), but with Gringo additional aggregates are implemented, for example the aggregate functions #*sum* , #*count* and #*avg*.
As for the two previous introduced grounders priorities can be set in Gringo. The different weighting of priorities does not depend on the input order like in Lparse, but on weighting parameters that can be set individually for every priority.

I-DLV The I-DLV is one of the newest developed grounders and is based on DLV. The new grounder provides some improvements. First of all annotations, but also the availability of external computations and interoperability. Furthermore, this grounder can be used with different solvers [12, 25].

Conclusion Comparing the just introduced grounders the following can be observed: The grounder Gringo and Lparse use the output language Smodels, the DLV grounder however uses a specific output language of DLV. As a wide range of solver use the input language Smodels, Gringo and Lparse can be combined with a wide variety of solvers. In the opposite, the DLV grounder can only be used with the corresponding DLV-solver. Using the grounder Gringo, a couple of aggregates can be applied that are not supported by Lparse. As this leads to a simpler implementation of problems Gringo is the currently most popular grounder [25, 42]. The newly introduced I-DLV is not widely in use, but first research projects show that this grounder is a promising approach.
Taking all arguments into account, the grounder Gringo is the best choice of a state-of-the-art grounder. Not only the output-language but also the numerous aggregate functions and priority implementations seem to give promising implementation possibilities.

Solver

The finding of answer sets for a given problem is NP-complete [17]. Different solvers have been developed to reduce the computational costs for the finding of answer sets. They use different algorithms and solving approaches. On the next pages the most-common solvers are introduced and their difference are discussed.

Smodels The solver Smodels is one of the first developed solver for Answer Set Problems [36].
The solving algorithm of Smodels is based on the Davis-Putnam-Logemann-Loveland (DPLL) procedure. This backtracking-based search is extended by interference rules to be suited for ASP. During the backtracking search the Smodels solver checks for unfounded sets and builds a material implication

graph. With that the solver monitors the number of unfounded sets [34, 35].

DLV The DLV solver is similar to Smodels an early developed solver for ASP. The solver is, similar to Smodels, based on the DPLL algorithm and extended by interference rules to be suited for ASP [25, 31]. The solver is, in opposite to Smodels, extended with lock-back heuristics and back jumping techniques [36].

WASP WASP is, in opposite to Smodels and DLV, based on the Conflict-Driven Clause Learning (CDCL)-style approach. Characteristics of the CDCL procedure are the backjumping-based search, the looping of no-goods and a conflict analysis [31].
WASP is the first solver that introduced an anytime algorithm for cautious reasoning and an stratification technique. In early versions WASP used a modified DLV-grounder, but starting with WASP 2.1 the grounder Gringo is used [2].

Clasp The Clasp solver combines multithread solving mechanisms of CDCL with non-chronological backtracking [36] and cutting-edge techniques, like heuristics, to solve Boolean constraints [34].
A significant number of variants of Clasp have been published in recent years. One variant is the solver *Claspar*, in where Clasp is distributed on large clusters, or *Claspfolio* which combines elements of the ASP solver Clasp and the CSP solver Gecode [31, p. 150].
The solving process of Clasp can be modified by choosing specific heuristics that are used in the solving process. This can lead to an improvement in the computational costs for the solving process, as some heuristics fit some problems better than others. The different heuristics that can be used in Clasp are introduced below.

- **Vsids**: This look-back heuristic is derived from the CDCL-based Chaff-like SAT solver [48]. Variables are chosen to be used during the decision-making progress depending on the global activity of these variables [33].

- **Berkmin**: Like the Vsids heuristic Berkmin is a look-back heuristic derived from the CDCL-based Chaff-like SAT solver. The used algorithm is an improved version of the Vsids approach. The scope of variables that are used for decision making depends not on the globally most active and free variables, but on the age and activity. The variables that are most recently used in unsatisfied conflicts, but not yet solved, are selected. Thereby Berkmin reduces the impact of previous conflict causes [33, 41].
- **Vmtf**: Another modification of the Vsids heuristic is Vmtf. This look-back heuristic is not a Chaff- but a Siege-like decision heuristic. The difference to Vsids and Berkmin is the selection of a scope that is similar to an online sorting algorithm [33, 48].
- **Unit**: This heuristic is a Smodels-based heuristic [55].
- **None**: By selecting this solving approach an arbitrary static ordering is applied [29].
- **Domain**: With this argument, user-domain specific heuristics can be applied. Next to others a command line structure-oriented heuristic can be set up, specifying the modifier and the atoms the modification adjust. For instance by choosing the domain heuristic with the option *–dom-mod=5,8* a false statement is applied to all atoms that appear in optimization statements [29].

Not only heuristics can be set to improve the solving process but also different solving mechanisms are provided by Clasp. Some of them are introduced below:

- **Opt-strategy**: This argument modifies the Smodels enumeration algorithm. It can be chosen between a branch-and-bound-based and an unsatisfiable-core-based optimization. The selection of optimization can be fine-tuned by applying optional parameters. For instance with the argument *opt-strategy=bb,1* the solving algorithm is modified to be a branch-and-bound-based hierarchical algorithm that is often used to improve the solving of problems with multi-criteria optimization [29, 32].
- **Opt-heuristic**: In Clasp, a sign selection is applied based on the type of the variables. Bodies are preferable set true, and atoms are set false. Using the solving approach *–opt-heuristic=1*, the signs selection is altered [29].

- **Parallel-mode**: As an extension of the different solving processes additionally multi-threading can be applied. It can be chosen between the competition-based and the splitting-based search and the number of used threads [29].

2.3.4. Potsdam Answer Set Solving Collection

The Potsdam Answer Set Solving Collection (Potassco), a software developed by multiple researchers from Germany and Austria, is a collection of various ASP tools, like the grounder Gringo or the solver Clasp. The functionalities of the ASP grounders and solvers are extended by different functionalities, like the integration of python functions [29, 31].

Different Solving Collections

Potassco provides different solving systems in where grounders and solvers are combined. In what follows the most popular combinations are introduced.

Clingo In the monolytic system Clingo the grounder Gringo is used with the solver Clasp. The benefit of the combination of both systems is that, after passing the input data and problem description into the grounder, the user must not consider the information passing trough system components but gets back the optimal answer set solution in a human-readable format [31].

Clingcon This system is developed for applications that can be modelled easier by a combination of Boolean and linear integer constraints, like for instance fine timing applications. The grounder Gringo is used with a the ASP solver Clasp and the CSP solver Gecode and behaves similar to State-of-the-Art CSP solver [4, 31, 43].

IClingo For PSPACE-decision problems like automated planning applications the system IClingo is designed. Stable models are not solved repeatedly to find the best solution but computed incrementally. Thereby redundancy is avoided, as not the entire extended problem is solved repeatedly, but the problem is step-wise solved and extended by adding more and more extensions until the whole problem is solved [31].

OClingo The system OClingo is extending the functionalities of IClingo by online functionalities and acts like a server waiting for client requests. Whereas IClingo terminates after the generation of a stable model, OClingo continues running and waits for new requests [31].

External Python Functions

One advantage of using POTASSCO is the availability to integrate external python functions in ASP. By that functions that are hard to describe in ASP can be implemented in Python and be executed from the ASP program [29].

2.3.5. Potassco Input Language

All solving collections that are provided by Potassco are using the grounder Gringo. The language syntax that is required by this grounder is the logic programming syntax following the ASP-Core-2 input language format (see [11]) and is adapted by some extensions [28].
An overview of the language that is used as input language for Gringo version 4.3.0. is given in the following paragraphs.

Language Syntax

In ASP a problem is described by a finite set S of rules. This set of rules is named logic program P and consists of a set A of atoms [17].

Every rule follows the syntax

$$a_0 \leftarrow a_1, \ldots, a_k, \sim a_{k+1}, \ldots, \sim a_l \tag{2.19}$$

Where $0 \leqslant k \leqslant l$ and every $a_i \in A$ is an atom for $i = 0 \leqslant k \leqslant l$ [39].

Notations The atom a_0 on the left side of the condition character '$\leftarrow$' is the $head(r)$ of the rule. The $body(r)$ of the rule contains all atoms $a_1, ..., a_l$. The body is split in the positive part $body(r)^+$ with atoms $a_1, ..., a_k$ and the negative part $body(r)^-$ with atoms $a_{k+1}, ..., a_l$. All positive atoms $a_1, ..., a_k$ and default negated atoms $ak + 1, ..., a_l$ are called literals. The theory of default negation is explained in detail later in this section. Every atom a is expressed as $p(t_1, ..., t_n)$ where p is a predicate and $t_1, ...t_n$ are terms. A term can be a variable or a constant. In case no variable occurs in a term or atom they are named grounded terms or grounded atoms, respectively. If $body(r)^- = \varnothing$ the rule is named positive rule. If $body(r) = \varnothing$ the rule is named fact and for simplification the condition character '$\leftarrow$' is left away. If $head(r) = \varnothing$ the rule is named constraint [31].

Terminology Functions, predicates and constants are described by lower-case letters or strings with a small initial character. Variables are indicated by upper-case letters and strings starting with a capital letter [5].

Notation Conventions In ASP the syntax of logic programming used. Nevertheless, the notations of general logic programming and ASP are slightly different and listed in table 2.2.

	If	And	Or	Default Negation	Classical Negation
ASP Syntax	:-	,	|	not	-
Logic Programming Syntax	$\leftarrow$	,	;	$\sim$	$\neg$

Table 2.2.: Notation conventions of ASP and logic programs [31, p. 11]

Language Semantics

In ASP the semantics follow the stable model semantics of logic programs [31]. The idea of this approach is to replace all variables in rules (see equation 2.19) by constants so that only grounded atoms are given in a logic program P. This task is overtaken by the ASP grounder. From a program P the Herbrand model P_S for a set S of atoms from P is obtained by [39, p. 4]:

- deleting each rule with a negative atom $\sim a$ in its body and with $a \in S$
- deleting all negative atoms $\sim a$ from the remaining rules

If the resulting minimal Herbrand model P_S coincides with the set S of atoms, then the set S is a stable model of P and returned by the ASP solver as answer set [31, 39].

Language Extensions

Additional language extensions are provided to increase the capabilities for describing complex problems with the ASP syntax. Several extensions are introduced in the next paragraphs.

Negations In table 2.2 two negation notations are listed: Default and classical negation. The behaviour of these negations differs by the information that is provided for an atom or not. In case of a default negation no information is necessary to hold the statement (negation as failure), whereas in classical negation a proof for negation is necessary [5].

Example 2.1: Negations

Considering a robot who needs to unload a container on a specific place, but only if place is not occupied by of other goods or vehicles. This problem can be described with the following rules:

```
unload(P,S) :- place(P), robot(S), not place_occupied(P).
unload(P,S) :- place(P), robot(S), - place_occupied(P).
```

Assuming that no information of the third atom *place_occupied* is provided, it is not known if this atom holds true. The rule with a default negation (line 1) holds true, but for the classical negation in the second line of the code snippet it has to be proven that the station is free before a robot would unload the good [17].

Integrity Constraints Integrity constraints are used to state which combinations of literals are not allowed. As defined previously in this section constraints are rules without a body. If all literals of an integrity constraint hold true this answer set is forbidden [5].
Related to the stable model semantics a given integrity constraint with a set A of atoms a_i

$$:-a_1, a_2, ..., a_m. \tag{2.20}$$

can be translated in a normal logic rule by adding an additional atom $x \notin A$:

$$x :-a_1, a_2, ..., a_m, not\ x. \tag{2.21}$$

Applying the two steps introduced by Gelfond and Lifschitz [39] it can be seen that integrity constraints do not add or alter answer sets but only can eliminate them [31].

Example 2.2: Integrity Constraints

Given is following situation: Two containers have to be delivered at the same time but not with the same robot. This problem can be described as followed:

```
:- same_time(box1, box2), transport(box1,robot1),
   transport(box2,robot1).
```

A constraint does not hold if literals in the body all hold. In this example the constraint would hold if box1 and box2 are transported at the same time and robot1 is set to transport box1 and box2.

Conditional Literals Conditional literals in ASP are notated by the character ':'. They are of the form

$$a_0 : a_1, \ldots, a_n \tag{2.22}$$

Every a_j is a literal and all literals behind the mathematical set notation ':' are called condition [31].
Conditional literals can be used in the head as well as in the body of a rule. As a_0 and the conditions a_1 to a_n act like the head and body of a rule, conditional literals behave as nested implications [29].

Example 2.3: Conditional Literals

Given two robots and five stations. The robots have to deliver containers to the same place, but every robot can drive to only specific places. ASP is used to find a place both robots can deliver the containers.

```
1  robot(robot1).
2  robot(robot2).
3  place(place1; place2; place3).
4  allowed(robot1) :- not on(place1).
5  allowed(robot2) :- not on(place1; place3).
6  unload :- allowed(X) : robot(X).
7  on(X) : place(X) :- unload.
```

With the conditional literals in row 6 and 7 the following rules are instantiated:

```
1  unload :- allowed(robot1), allowed(robot2).
2  on(place1); on(place2); on(place3) :- unload.
```

The resulting answer set returns '*on(place2)*' to be the only possible place where bot robot can deliver the containers.
Not only conditional literals are introduced in the example above, but also a useful writing convention: Row 3 is simplified expression for:

```
3  place(place1).
4  place(place2).
5  place(place3).
```

Intervals Intervals are used to generate multiple terms in a row [29].

Example 2.4: Intervals

For a problem multiple terms have to be set with values in an increasing order. This can be done the following way:

```
set_val(1).
set_val(2).
set_val(3).
set_val(4).
```

Using intervals the just introduced code snippet can be simplified to

```
set_val(1..4).
```

Boolean Constraints For problem instances where values are compared, literals that are always true or false are required and provided as the Boolean constraints *#true* and *#false* [29].

Example 2.5: Boolean Constraints

Following program is provided, where robots are sorted by decreasing priorities.

```
robot(robot1,3).
robot(robot4, 4).
robot(robot2, 5).
robot(robot3, 1).
next_task(R1,R2) :- robot(R1,P1),robot(R2,P2), P1 > P2,
  #false : P1 > P3, robot(R3, P3), P3 > P2.
```

The resulting answer set is:

```
next_task(robot2,robot4) next_task(robot4,robot1)
  next_task(robot1,robot3)
```

In the first four rows facts are stated, allocating robots with corresponding priorities to the atom *robot*. In the fifth row the provided priorities are sorted in decreasing order and saved in the atom *next_task(R1,R2)*. Commas separate not only literals in rule bodies, but also conditions. The rule in line 5 holds only for values $P1 > P2$ and if the conditions after the colon returns false. This occurs if there is no value $P3$ with $P1 < P3 < P2$ [29].

Choice Rules Choice rules are of the form

$$\{a_0; ...; a_m\} : -a_{m+1}, ..., a_n, \text{not } a_{n+1}, ..., \text{not } a_p. \tag{2.23}$$

where $0 \leqslant m \leqslant n \leqslant p$.
The main idea of choice rules is to provide a rule with a body and a set of head literals. If the body holds true a subset of head literals is fullfilled [31].

Example 2.6: Choice Rules

A robot stopped a delivering process. Based on some status information a reason for this unexpected behaviour has to be selected.

```
{reason(trafficInterference); reason(wrongLaserData);
  reason(pathBlocked)} :- status(obstacledetected).
```

The reason for the stopped delivery process is may or may not a traffic interference, a detected obstacle that blocks the travelling path, or an incorrect laser scan. The ASP solver returns all possible combinations of the reasons as answer sets.

Cardinality Constraints Cardinality constraints are an extension of the choice rule. Using cardinality constraints, subsets with a minimum (l_1) and maximum (l_2) number of atoms in the answer set can be specified. Cardinality constraints can be used at the head and body of a rule separately, but also on both sides [31].

The general syntax of cardinality constraints in the head is the following:

$$l_1\{a_1; \ldots; a_m\}l_2 : -a_{m+1}, \ldots, a_n, not a_{n+1}, \ldots, not a_p. \tag{2.24}$$

Example 2.7: Cardinality Constraints

After arriving at the warehouse of a logistic centre, a robot picks containers, but not more than three due to a limited loading capacity.

```
1{pick(box1); pick(box2); pick(box3); pick(box4)}3 :-
   at(warehouse).
```

The ASP solver returns 14 possible answer sets, where at least one, but not more than three containers are assigned to be picked by the robot.

Conditional Literals in Cardinality Constraints Conditional literals can be combined with weight rules to reduce the number of instantiations in a rule [31].

Example 2.8: Conditional Literals in Cardinality Constraints

Given 3 robots and a box that has to be picked up and delivered by one of the robots. This problem can be described by the following program:

```
1  robot(id1).
2  robot(id2).
3  robot(id3).
4  1{assign(R,box1):robot(R)}1 :- pick(box1).
```

The conditional literal in line 4 expands to the cardinally constraint

```
1{assign(robot1,box1); assign(robot2,box1);
  assign(robot3,box1)}1 :- pick(box1).
```

The ASP returns three possible solutions:
assign(robot1,box1), *assign(robot2,box1)* and *assign(robot3,box1)*.

Optimization Statements For problem instances that focus on optimization problems, the input language provides optimization statements. Cost functions can be minimized (*#minimize*) or maximized (*#maximize*), and the different optimization statements can be prioritized.
The syntax of the minimization statement is the following [29]:

$$\#minimize\{w_1@p_1, t_1 : L_1, ..., w_n@p_n, t_n : L_n\}. \tag{2.25}$$

Where w are the integer costs to be summed and optimized, p are priority integers, and t and L are terms and literals. The priorities $@p$ are optional and sorted in an increasing order.

Example 2.9: Optimization Statements

A container has to be picked up and delivered by a robot. A list of robots is provided with additional information about the battery level of each robot and the distance to the container. The optimal robot to pick up the container has to be found. Most important is to select a robot with a high battery level, additionally the travelling distance to the container should be minimized. The Gringo input looks like the following:

```
robot(robot1, 70, 120).
robot(robot2, 30, 70).
robot(robot3, 70, 300).
1{assign(container1, I, B, D) : robot(I,B,D)}1 :-
  pick(container1).
#maximize(B@2, I, B : assign(container1, I, B, D)).
#minimize(D@1, I, D : assign(container1, I, B, D)).
```

Applying only the maximization criterion for the battery level,two optimal answer sets would be found:
assign(container1, robot1, 70,120) and *assign(container1, robot3, 70,300)*.
With the second optimization criterion, the minimization of the travelling distance, the two answer sets are reduced to only one optimal solution: The assignment of *robot1* to the container.

Special Gringo Input Language Features

Next to the introduced language extensions Gringo provides special aggregates and directives.

Aggregates Aggregates support the return of specific values from a set of items, for instance the sum of all items (#*sum*), the maximum or minimum value of the selected items (#*max* and #*min*), or the number of items (#*count*) [29, 31].

Directives The directives that are provided by Gringo can be split in three groups [29, 31]:

- Comments: Within the code comments can be set. A comment over one line is marked with and percent sign %, comments over multiple lines are started with %* and end with *%.
- Output: Parts of the program can be displayed in the terminal where the program is called. The parts that should be displayed are marked with the prefix #*show*.
- for some problem instances constants are required. The constants can be defined in the program with the prefix #*const*.

2.3.6. Development Tools

In the following Integrated Development Environments (IDE) and some Application Programming Interfaces (API) are introduced that support ASP.

Integrated Development Environments for ASP

For the just introduced language that is used by the grounder Gringo different rules, constraints and optimization criteria have been shown. Encoding a new problem in ASP is, as in all imperative and declarative languages, error-prone. In the following Integrated Development Environments (IDE) for ASP are introduced. With integrated syntax checking and debugging options the required time to find errors can be reduced.

ASPIDE ASPIDE, the Integrated Development Environment for Answer Set Programming, is supporting the whole development of ASP instances, starting with the problem definition up to the deployment of the result to applications. The IDE is an editing tool that provides additional graphical tools for the programming composition, debugging, the configuration of solver execution, output handling, and others. Supported by the ASPIDE is the language syntax of ASP-Core-2 and the syntax of DLV. That leads already to the limiting factors of this IDE. It supports only the DLV, but not Gringo, and by that it can not be used with Potassco [22].

SeaLion SeaLion, the Support Environment for ASP, is a source code editor that supports the two grounders Gringo and DLV. The solver DLV can be used just as the solver of Potassco. The IDE is installed as a plug-in for the Eclipse Platform or as a standalone package. Next to others SeaLion supports syntax highlighting and checking, code completion and debugging features. UML class diagrams can model data structures, and answer sets can be visualized in instance diagrams. However, the IDE is no longer actively developed and supports only Clingo V3 even though the state-of-the-art version of Clingo is V5.3.0 [10].

Application Programming Interfaces for ASP

For many problem instances were ASP is used, it is required to integrate the declarative code in an imperative environment. Different APIs that support this integration are introduced in the following.

Potassco Python-API and C-API To integrate an ASP module in an existing system APIs are required. POTASSCO provides two APIs, one written in Python [14], on in C [13]. With these APIs, the POTASSCO system Clingo can be imported as a library in an existing system and be executed within this system.

Jasp In [21] a framework is introduced with which ASP encodings can be integrated into a Java environment. The ASP code is embedded in the Java Code and can access all variables of the environment. The resulting answer sets of the ASP are stored in Java objects. This new language *Jasp* is implemented in the *JDLV* framework. Currently, only the DLV grounder and solver can be used with this plug-in. The numerous solver provided by Potassco are not supported.

3. Related Research

In this chapter current research projects are introduced. The main focus is thereby the task assignment for industrial transport robots and applications of ASP, especially POTASSCO.

3.1. Task Assignment for Industrial Transport Robots

Intralogistics robots transport goods within a fixed time window in a specific order. Starting at a warehouse, the robot picks up a good and delivers it to another location. This specific assignment problem is often called multi-robot task allocation with temporal and ordering constraints (*MRTA/TOC*) and described by the *ST-SR-TA*-problem with additional constraints [51]. Different optimization objectives to find suitable assignment sets are published in recent years. A small selection is presented here.
The goal for the optimization objective *MiniSum* is to minimize a total distance. This distance contains the sum of all distances of the traveling robots [15].
Similar to the *MiniSum*-approach using the *MiniMax* objective the sum of costs of all robots is taken. The difference is the optimization: In this approach not the total distance is optimized, but the makespan over all tasks that are executed, leading to a solution in which the tasks are finished as fast as possible [26].
At the optimization objective *Lateness or Tardiness Minimization* the tardiness is minimized. Given a set of tasks and robots, the tardiness is the difference between the earliest scheduled start time of a task and its earliest possible start time using one of the robots. In case more than one assigned pair have

the same tardiness, the best assignment of task and robot is the one with the shortest travel time of the robot [60].

3.2. Use Cases of ASP

Declarative Programming and especially Answer Set Programming is a promising technology for many research and industrial problems. Different industrial fields are affected, from online platforms and shift design in different areas through to solving approaches for industrial robot challenges.

3.2.1. Optimal Shift Schedules

Answer Set Programming is a promising approach for search and approximation problems [8]. One of these problems is the shift problem that has been solved by ASP already in different applications.

An early published application of ASP is the shift plan design or workforce assignment problem. In [16] a shift plan is designed for nurses in an Italian hospital using Answer Set Programming. This plan generation is affected by some constraints like a couple of shift types with different times and different workforce resources or by statutory holiday entitlement and rest days. Using the system Clingo and the solver WASP with the grounder Gringo to design the shift plan for 164 nurses, the calculation for a year of planned shifts took around 50 minutes for Clingo, the solving process using WASP was interrupted as the execution time was significantly higher. It is shown that ASP system Clingo is a useful tool to support the shift planning of the head nurse.

Not only in hospitals ASP is used for shift planning. In [52] the system DLV is used to plan the shifts of seaport workers. Difference to the previously introduced nurse scheduling is the focus on the allocation of different qualified employees to shifts where different tasks are needed, depending on the current boat traffic. Additional constraints are next to others an equal workload, skills of workers, and a maximum of weekly working hours. With the DLV-grounder and DLV-solver the shifts for 130 workers were planned. The computation for a month-long shift plan took 8 minutes. Having similar constraints, number of workers and computer specifications the system Clingo in [16] is faster than the solving process using the DLV.

Another shift approximation was introduced in [1]. Like at [52], employees with different qualifications have to be assigned to shifts that require different skills. The difference between both approaches is the focus on work balance optimization in [52] and shift alignment optimization in [1]. The ASP system Clingo is used in normal mode and with the integrated heuristics. As the approach with default heuristics looks quite promising, even better results are expected using user-specific heuristics.

3.2.2. Industrial Applications

Not only shift design problems can be solved with ASP, but also many other hard computational problems. In [19] the application of ASP in E-Tourism platforms is mentioned. Using the system DLV, knowing user specific wishes (like beachside hotels) and a set of different travel offers, the best suiting travel offer is recommended.

Next to that ASP can be used for routing and classification of customers in call centres [47]. A customer calling the hotline is classified before he is forwarded to a representative. Constraints are among others the age, residence, type of insurance contract and the number of previous calls. Using the ASP system DLV the customer is classified in a category and assigned directly to the appropriate human operator, for example an insurance expert for natural disasters.

3.2.3. Task Assignment Problems

Task assignment is another complex problem. In general, a set of tasks has to be assigned optimally to a set of workers. One example of task assignment for multiple robots is the planning of tasks for housekeeping robots [18]. A set of robots has to tidy up a house by filling the dishwasher, putting books on the shelf and make the bed. Common-sense knowledge, like the fragility of glasses and that books do not belong in the dishwasher is defined in external functions. Using the ASP system IClingo it can be observed that in comparison to the implementation with C+ the execution time stays the same, but the computation memory is reduced.

Task Assignment and Routing

Some research projects use ASP to solve task assignment problems with simultaneous consideration of routing strategies.
In [54] Answer Set Programming is used in an order-picking system. Cellular transport vehicles deliver goods stored in multi-level racks to picking stations. Goods are part of order-lines, all goods of an order line are delivered to the same picking station, and only one order-line can be assigned to a picking-station at a time. Two problems are solved using ASP. The assignment of vehicles to delivering tasks and the assignment of picking stations to order-lines.
The task assignment is managed decentralized. Changing the status to idle the vehicle receives data from a server. The data consists of all unassigned orders and the current order-to-picking station assignments. The goal of the task assignment is to assign delivering tasks to vehicles. Thereby three optimization criteria are considered:

- minimization of the distance between the vehicle and the storage of goods
- having an equal workload on all picking stations
- delivering first all goods of an order-line that currently occupies a picking station.

For the most important optimization criteria, the distance between vehicles and storage of goods, an external Dijkstra-algorithm is used.
Comparing the results using the task assignment, picking station assignment and the simple FIFO[1]-protocol the task assignment shows the best performance, followed by the picking station assignment. An increase in

[1]First in, first out

orders and vehicles and by that an increase in traffic volume reduces the overall performance. Regarding the computational costs it is shown that the most time consuming part is the path calculation.

Another task assignment and routing solving approach using ASP is introduced in [38]. A set of vehicles is used in a car assembly, supplying the assembly lines by executing tasks. Every task has a sequence of subtasks whose elements are halt nodes and describe the intermediate goals of the vehicle, like assembly lines or warehouses. In every task a full container is delivered to an assembly line, there an empty container is replaced by the full container and the empty container is transported to a recycling facility. A hard constraint is to finish the delivering of a container to the assembly line within a deadline. To solve the task assignment and routing problem the ASP system Clingo is used. The result of ASP is compared to *OpenTCS*, a default scheduler provided by Fraunhofer. The benefit of ASP is the ability to handle multiple vehicles that have to visit the same station. OpenTCS breaks while solving such a situation. However, during the run of different simulations it is shown that the computation effort of ASP is significantly higher than the one of *OpenTCS*. Like in [54] the reason for that is most likely the path planning.

In [50] another approach for solving the task assignment and path planning problem with ASP was introduced. An unequal number of tasks and robots is allowed, tasks have deadlines and can be part of a task-group that has to be executed in a fixed order, and the robot has to visit checkpoints on the way. Two optimization goals are introduced, the minimization of the make span and the total path costs. The difference between both is that for the first optimization the total time is minimized whereas for the second optimization the total distance is minimized. It is stated that ASP is more scalable for task assignment and path planning algorithms than imperative methods. Using the ASP system Clingo it is shown that for small problem instances existing imperative solving methods are better, but for difficult problems with many constraints ASP shows better results.

A further task assignment and path planning algorithm was presented in [53]. Difference to the previous methods is the focus on robot teams and a strictly decentralized approach. Given is a set of teams and robots of different types. Due to the decentralized approach robot teams do not talk to each other but answer Boolean questions to a server. An additional constraint is the ability to exchange team members. Teams can borrow and lend robots from other teams in case of a lack of resources, but the number of borrow and lend executions is limited by time windows. The path planning computation effort is reduced using lower and upper bounds, see [58]. It is shown that due to the parallel computation of multiple teams of robots the computational costs was reduced significantly compared to a scenario of robots without teams.

Task Assignment, Routing and Intralogistics Domains

Until now many ASP applications in industrial areas have been introduced. A difficulty, however, is the integration of the task assignment and routing algorithm in an environment. In [37] the benchmark suite ASPRILO for robotic intra-logistic domains is introduced. A problem domain is set up, where a map with a grid of squares is defined. Every square is set to be a highway, picking station or storage location. For a given set of orders robots drive to storage locations, pick up a shelf and drive it to a picking station. After the requested quantity of goods is picked from the shelve by a worker the robot drives the shelf back to a free storage location. Every square can be occupied by not more than one robot. The implementation of charging stations is planned for a later version of ASPRILO.
The instance generator for the map requires the grid dimension, the number of orders, robots, shelves and picking stations for a successful set up. The generation of the map instance is built on a multishot ASP that is controlled by the python API of Clingo.
After the instance generation a solution checker is executed in ASPRILO, proving the success of delivering all provided orders. If the solution checker returns success an ASP is started in where a multi-agent path finding algorithm is implemented. In every time step the robot can move to an adjacent square, pick up or pick down a shelf or can deliver a shelf to an adjacent square. Additionally, a task assignment algorithm with several optimization criteria is implemented in this ASP to find an economic assignment.
The runtime of the ASPRILO with Clingo and Clingcon encoding is analysed for different layout and problem instances. In general, Clingo shows

to have a better runtime for higher scaled instances whereas Clingcon fits better for small instances.
The differences of two calling methods are analyzed. The calling of the task assignment and pathfinding at the same time and the calling of the pathfinding algorithm after an optimal task assignment is found. The results show a significant difference in the runtime of the program. First the ASPRILO implementation with one ASP and the Clingo encoding was analysed. For small layout instances with a map of 11x6 squares a timeout limit of 3 minutes solving time is reached only for problem instances with more than 8 robots. For a medium instance with a map of 19x9 squares this timeout is reached already for problem instances with five robots.
Analysing the runtime of the ASPRILO implementation with two separate ASP encodings for task assignment and multi-agent path finding with the Clingo encoding a significant runtime improvement is shown. For small problem instances with 11 robots a solution is found within 20 seconds instead of reaching the 3 minutes timeout. For medium instances and five robots a solution is found in 59 seconds, and even the instance with 19 robots can be solved within the time limit.
For large problem instances with a map of 46x15 grids no solution can be found within the time limit, independent from the selected encoding and the ASPRILO implementation.

4. Intralogistics Management - An Overview

After analysing current research projects it can be seen that ASP is a promising solving approach for task assignment problems in logistic-related areas. The BMW Group and incubed IT currently face the challenge of an increasing runtime and complexity of their algorithm to solve task assignment problems. By replacing imperative code components of the algorithm by an ASP program it is hoped to receive better performance and comprehensibility of the code.
To implement a task assignment problem solution using Answer Set Programming, in a first step the environment with the corresponding requirements and constraints of a problem must be analysed. Given this information possible applications for ASP can be identified [20].
In the following sections an overview of the intralogistics management at BMW and incubed IT is given.

4.1. Intralogistics Management at BMW

In this section a general overview of the logistics environment at BMW is given and the fleet management system is introduced.

4.1.1. General Overview

The BMW Group one of the the world's leading company manufacturing premium automobiles. In 2018 more than 2.5 Million cars were produced at 20 plants worldwide, where the plant at Regensburg produced nearly

320.00 cars [6].
To ensure a smooth production process in the plants all car parts have to be delivered just-in-time to the correct production sector. In the course of digitalization of the industry the conventional forklifts shall be replaced by robots. The BMW Group plant Regensburg is the leading plant for robots delivering large load carriers automatically to desired locations.

Application of Industrial Transport Robots at BMW Group Plants

In the production process many different parts are needed to build a premium car. However, with the increase of parts the complexity of delivery processes increases. The parts have different sizes, starting from small screws up to trunk lids. Some parts like wipers are needed for every car, optional ordered equipment is only built in some of the cars. This wide range of product sizes and product quantities requires different intralogistics robots for the delivering process and different refill times for parts at the production lines.
Upon receiving a delivery from a supplier, the goods are prepared for the production line. Therefore they are unpacked and reloaded in containers. Depending on the refill times and weight and size of the container, the goods are transported to the production line with tugger trains or forklifts. On the production sequence the containers are unloaded and placed on the line or sorted in flow racks.
In times of Industry 4.0 the operational procedures are digitalised and automatized. Not only the production flow itself is considered therefore but the delivering processes as well. At BMW delivering tasks are step by step overtaken by autonomous logistic robots. Leading plants are defined in where different robots are developed and tested. There are leading plants for robots that replace tugger trains, put containers autonomous into different racks or unpack boxes received from suppliers. The BMW plant in Regensburg is the leading plant for the replacement of forklifts.

Application of Industrial Transport Robots at Plant Regensburg

As one of the leading plants of the BMW Group, the plant in Regensburg develops and tests the Smart Transport Robot (STR). An STR can pick a container at a specified location, like a supermarket[1], and deliver it to a goal station. Given only the coordinates of the different pick up and release stations, the STR drives autonomous to the stations, detects the container to pick and lifts and releases it autonomously.

Smart Transport Robot

The STR is an autonomous robot and used to carry containers. Developed in cooperation with the Fraunhofer institute, the focus of the development was next to the fulfilment of the logistics requirements easy maintenance and affordable costs. To ensure the latter two criteria many vehicle components are used to build up the robot. For example, the BMW PDC (park distance control) sensor is used as well as batteries of recycled electric vehicles.
To provide autonomous driving, the STR is equipped with a 2D laser scanner to detect the environment in driving direction. Obstacles in the back are detected by the PDC sensor and a 3D camera is mounted to support the process of lifting containers.

Map For the delivering process of goods a map of the plant must be provided. This map is generated by scanning the whole plant with a 2D-laser scanner. This is mostly done by driving an STR manually through the area of interest and tracking and saving the laser scans meanwhile.
After the map is generated points of interest (POI) are placed on it. The different POIs are:

- **Waypoint:** orientation point needed for the pick-up process
- **Parking Place:** station for parking the STR
- **Charging Station:** station for charging the STR
- **Dolly Place:** pick-up and delivery stations

[1]Supermarket: storage area of full containers

Delivering Process The STR is developed to transport containers of different sizes and weights. The main focus thereby lies on the large load carrier that are typically used at the BMW production lines. In figure 4.1 an STR is shown transporting a large load carrier. To lift a container the STR has

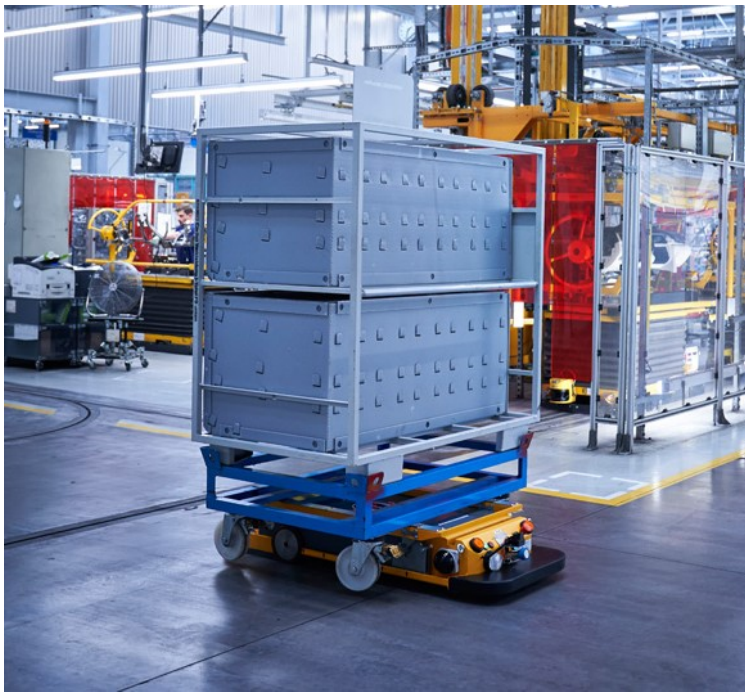
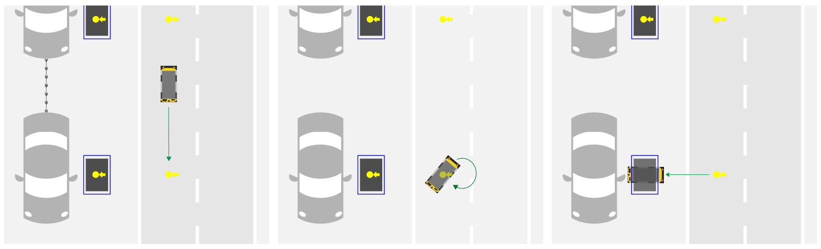


Figure 4.1.: Driving STR with a loaded container

two lift bars, one on either side of the robot. After driving underneath a container he can be lifted by moving these two bars upwards.
The pick-up process has been constructed as followed. A map of the plant, where different POIs are defined, is provided to every robot. Next to every pick-up station a waypoint is set in a 3-meter range. After the STR received a new delivering order it drives to the waypoint of the first task of this order (see figure 4.2a). Arriving at this point the STR turns 90° to have is front side look directly to the pick-up station the container is placed (see figure 4.2b). By taking an image with the 3D camera and comparing this image with the data of the laser scanner the exact position of the container can be detected. Based on this calculated position the STR drives underneath the container and lifts it (see figure 4.2c).



(a) driving to waypoint (b) adjusting position (c) lifting container

Figure 4.2.: Pick-Up process of an STR

After picking up the container successfully the STR drives out of the station forwards or backwards, depending on the given map and the current environment. To release a container the STR drives to the container place that is defined as delivery goal in the task and releases the container as soon as the position of the robot is the same as the coordinates of this container place. In the typical use case of the STR one robot drives to the assembly line, picks up an empty container and drives this container to the empty's storage. The same or another STR drives then to the supermarket, picks up a full container and drives to the assembly line sequence the empty box was removed before. Having only one container with the corresponding parts at the assembly line sequence would lead to an interrupt of production until a new container with the required goods arrives. As this interrupt is an unaccepted behaviour on production lines two containers with the same parts are placed at the assembly line sequences. The assembly line worker empties first one box before taking elements out of the second box. A fixed time window is given in which the STR has to replace the empty container by a new one.

4.1.2. Fleet Management System

At BMW the autonomous logistic process is managed partly centralized and partly decentralized. The navigation is done decentralized by every robot itself, but the goal a robot is assigned to is managed by a centralized Fleet Management System.

Fleet Management System - Tasks

The Fleet Management System is a system where robots are assigned optimally to orders, charging stations and parking places. Additionally, the FMS proves all provided data for correctness and validity.
To provide all the information the robots and the Fleet Management System (FMS) need, an Azure Cloud is set up. From there the FMS is started, allocation orders are communicated through WIFI to the robots, and robots provide their current state to the Cloud as this information is needed for the Fleet Management System.

Fleet Management System - Strategy

In the FMS the optimal assignment of robots to tasks, charging stations and parking places is managed. The optimal assignment is under constant development, and different strategies are analysed and evaluated. In the next paragraphs the current implementation at BMW Group is introduced. This system will be modified within the next few months to be capable of managing some hundreds of robots with an improved strategy for task allocation, charging and parking. Additionally components like a traffic management system will be set up.
In the next sections the assignment strategies of robots to charging stations, parking places and orders are introduced.

Parking and Charging Strategy

In general, the strategy for optimal parking and charging can be split into two different scenarios. In one the robot is currently in a charging station, in the second scenario it is not. In table 4.1 the current charging strategy of BMW is shown. As soon as the battery level of a robot that is on the field is below 25% the robot is send charging. In case a good is currently delivered this process is finished before. After arriving at the charging station, the robot charges up to 40%. If open tasks are available the robot leaves the charging station after reaching this level to process the next task. If no open tasks are available, the robot continues charging up to 60%. In case no other

vehicle with a lower battery level requires the charging station, the robot continuous charging up to 90%. Otherwise the robot leaves the charging station and drives to an appropriate parking place.
The decision about whether the robot goes charging or parking depends on the battery level. If there is another free robot with a lower battery level, this second robot is sent to charging. In case there are no available charging stations the robot drives to the closest parking place.

Battery Level	**Vehicle on the Field**	**Vehicle in Charging Station**
90% - 100%	Send to parking place or charging position if no mission is available	No charging, just parking
<90%		Vehicle can be called for mission or send to parking place in case charging station required
<60%		
<40%		Nothing but charging
<25%	Send to charging station	
<10%	Battery level dangerous, alert for System administrator	Battery level dangerous, alert for System administrator

Table 4.1.: BMW charging strategy

Job Broker

The Job Broker is a specific function in the Azure Cloud, focusing on the allocation of tasks to robots in an optimal way. At BMW the Job Broker obtains a list of assignable tasks and available robots as input information. Based on this provided data an optimal assignment is searched.

Generation of Tasks In case a production line sector runs out of parts, a new order is generated in PMS. Based on the type of order a process chain is generated automatically in the Azure Cloud. This chain consists of all tasks that are needed for the successful execution of the order. A task can, for example, be the pick-up of a empties box at the production line and the delivering of this box to the empty's storage.

Tasks are split into normal and event-triggered tasks. The FMS sends normal tasks to the Job Broker right after the generation of the order. Event-triggered tasks can not be executed immediately after the order generation but are sent to the Job Broker as soon as the trigger is set. Example of an event-triggered task is the delivering of a full box to a production line sector. The box can be delivered successfully only in case the empties box is removed. To ensure that the delivering task waits for another robot that picks up the empties box and sets the trigger after leaving the station.

Current Task Assignment The BMW task assignment is currently based on the FIFO-strategy. Tasks that are generated first have to be executed first. Every task belongs to a fleet. Fleets can be for example different production areas or floors. A task can only be assigned to robots that are in the same fleet.
The optimisation objective of the BMW task assignment is to ensure the fastest delivery for the orders with the earliest creation times. As a measure for the time for delivering the Euclidean distance of the robot to the first goal of the task is calculated. For an empties-container-task the distance from the robot to the production sequence is calculated, for a full-container-task the distance from the robot to the waypoint next to the full container in the supermarket is taken.
Given a list of open tasks, the one with the earliest creation time is taken. Comparing the Euclidean distance of all assignable robots to the first goal of the task the robot with the shortest distance is chosen to be the optimal assignment. In the next step, the second-oldest task is taken, and the Euclidean distance of all remaining robots is calculated. This strategy is continued until no more open tasks or assignable robots are in the lists.
Looking at chapter 2.1.3 this allocation can be classified as an ST-SR-TA. Every robot can handle at most one task, and every task can be finished by one robot. If more tasks than robots are available a scheduling of the assignment (TA) is necessary. Within a high-scaled FMS multiple orders will be generated in a small-time range, and by that the Job Broker will be called every few seconds to assign the new generated tasks. Having such small restarts of the task assignment algorithm the ST-SR-TA can be simplified to an ST-SR-IA, as the scheduling is replaced by a repeatedly called task assignment algorithm.

4.2. Intralogistics Management at incubed IT

In the following section an overview of the application fields and the logistics management of incubed IT is provided.

4.2.1. General Overview

Incubed IT is a robotics-software company, established in 2011 and located in Hart near Graz, Austria. The company develops software for smart transport robots, like the one shown in figure 4.3. The software provides

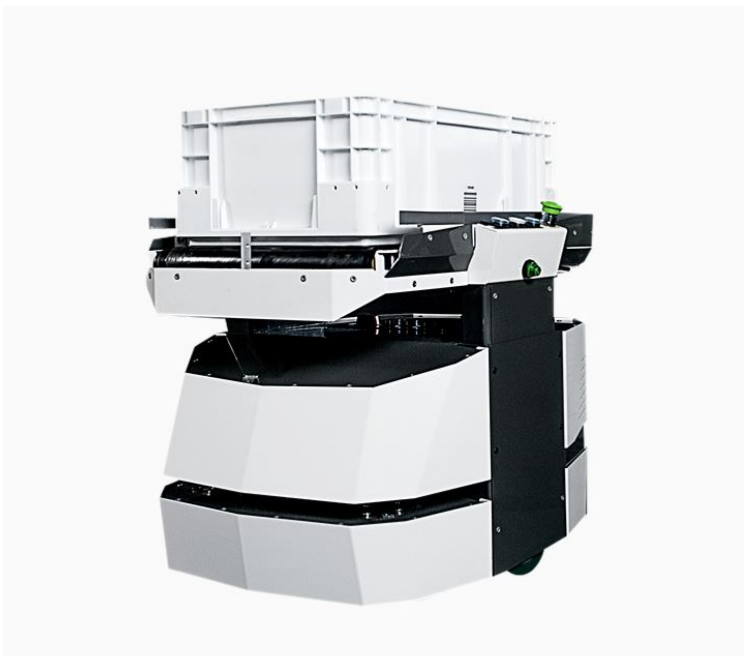


Figure 4.3.: Smart shuttle used with incubed IT software

applications for navigation and fleet management to allow autonomous driving industrial transport robots.

Application Fields of Industrial Transport Robots

Incubed IT focuses on the software development for smart robots. The aim is to provide one solution for a wide variety of different transport robots. The software is able to control autonomous robots that can be used in different logistic fields, like in warehouses of online traders, at logistic centres of supermarkets, and in car manufacturing plants. Different application fields lead to different requirements that must be considered for every customer individually.

4.2.2. Fleet Management System

At incubed IT industrial transport robots are, similar to BMW, managed by a partly centralized, partly decentralized system.
The entire navigation, such as path planning, localization, routing and docking at stations is performed decentralized by the robots individually.

Fleet Management System - Tasks

The Fleet Management System is responsible for all assignments of robots to goals. In this system all orders are created and managed, orders are assigned to robots, charging and parking strategies are pursued, and traffic rules are communicated to the fleet in case of traffic interference. The map the robots are using for navigation is stored in the FMS as well as the parameter configuration used to activate and customize user-specific services. Furthermore, the FMS provides a connection to all external systems, such as WMS or PMS, and to the robots.

Parking and Charging Strategy

At incubed IT, three charging scenarios are considered:

- **Threshold-based charging:** a robot is commanded charging by the FMS when the battery level falls below a state-depending threshold.

- **Fixed time slot charging:** the FMS forces the robot to charge at specified time slots (for example at night or during shift changes).
- **Transfer station charging:** in case a delivery station is combined with a charging station, robots charge during delivery.

The transfer station charging is the only charging scenario that is not managed by the FMS but by the robot itself. As this thesis is focused on the FMS rather than on the internal robot systems this charging type will not be explained further. Instead the first two charging types, namely threshold-based and fixed time slot charging, will be discussed in detail.

Threshold-based Charging In this charging strategy the robot is charging when the battery level falls below a threshold. This threshold depends on the current state of the robot which can be *busy*, *active* or *idle*. The different states are explained in table 4.2.

State	Robot State	Order State
Busy	Automatic mode	Order is executed by robot
Active	Automatic mode	Order available or assigned to robot
Idle	Idle or automatic mode	No order available

Table 4.2.: Robot states at incubed IT

Every state has its specific charging thresholds defining when the robot is entering and leaving the charging station. The different state-dependent thresholds are shown in figure 4.4.
Depending on the specific requirements for the system additional parameters can be set, where some are now briefly introduced.
One parameter specifies if a robot with a battery level below the critical charge limit is allowed to charge while carrying goods or if the robot has first to unload these goods before charging. Another parameter sets the ability to balance charging. Every charging of the robot is count. In case this counter reaches the specific value the user set, the robot does not only charge as specified in figure 4.4 up to the active charge limit but up to the balance charge limit.

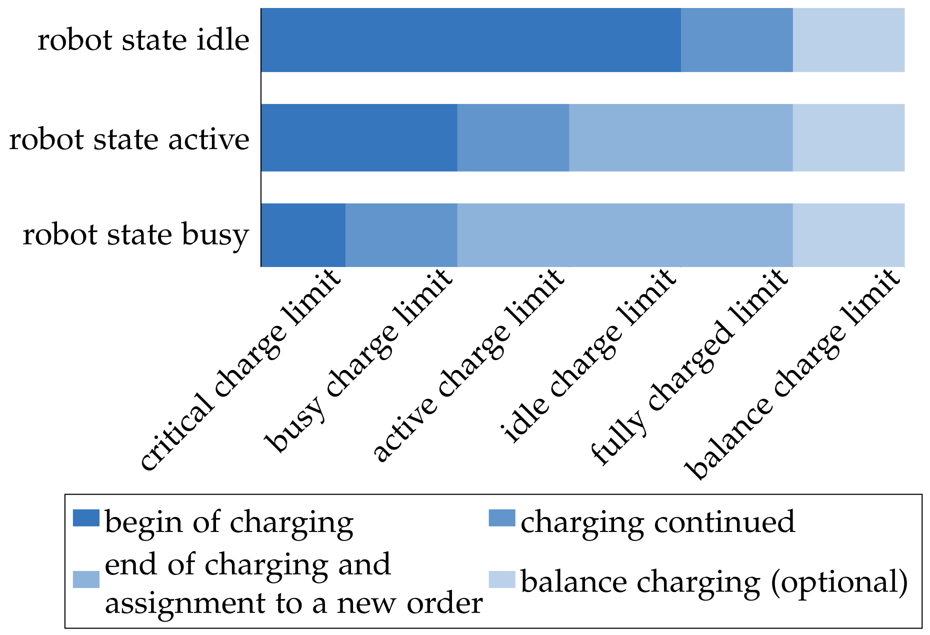


Figure 4.4.: Graphical representation of the incubed IT charging strategy

Fixed Time Slot Charging At fixed time slot charging the robot is forced to charge at a specific time. This time is defined by the user and can, for example, be used to ensure that every robot is charging at night and is fully charged the next morning.

Priority of Charging Stations and Parking Places Charging stations and parking places have a priority attribute. By using this parameter it can be controlled whether the robot should prefer some charging stations instead of others. An application is for example a set of charging stations where some are superchargers and some are normal charging stations. By setting different priorities, it can be ensured that robots are preferable assigned to superchargers.
The algorithm behind these priorities works as followed. In a first step a list is generated containing all charging stations in a priority-decreasing order. In the next step the Euclidean distance is calculated for all chargeable robots to all charging stations. But only stations with the highest priority are considered. If all highest-prioritized charging stations are assigned, the

next smaller prioritized charging stations are analyzed. The same algorithm is applied for the priority-based assignment of parking stations.

Task Assignment

The task assignment at incubed IT is significantly more complex than the one at BMW. The main reason for that is the variety of different application fields and the higher number of required constraints that have to be considered.

Types of Tasks There are three task types provided at incubed IT.
The first one is a normal task. As soon as this task is available a robot can be assigned.
More complex than this are the so called *Chained Orders*. Such orders consist of n tasks ($n > 1$), where every task has a predecessor who has to be finished before the same robot can execute the current task. The robot is not delivering other orders until all tasks of the chained order are finished.
The third task type is named *parallel order*. Some intralogistics robots can deliver more than one good at a time. At the current implementation at incubed IT parallel orders can be defined that consists of a set of tasks. All tasks have the same pickup and delivery station and are delivered by the same robot simultaneously.

Priority of Tasks Every task has a priority-attribute which can be set at the task generation. This priority indicates which task should be preferred and executed first. To ensure that tasks with a very low priority are executed ageing of orders can be activated. By that the FMS increases the priority of a task each time the task is not assigned to a robot.

Status of Tasks One of the parameters of the tasks is its status that is generated and modified by the FMS. The status indicates for example if a task was just generated (status new), is assigned to a robot and drives to the first station (status assigned) or is already finished (status finished).
For the task allocation mainly accepted orders are of interest. New tasks

are first reviewed to ensure that all parameters are set correctly and none is missing. If the task is completely defined the status is set to accepted by the FMS and ready to be assigned to free robots.

Assignment Algorithm Two algorithms can be chosen for the task assignment: *FIFO* and the *global optimum*. At *FIFO* the task that was generated first has to be executed before a later generated one. The *global optimum* algorithm chooses the best assignments based on the order priorities and the robots traveling costs. The traveling costs consist of the Euclidean distance between the robot and the first goal of the assigned task. The costs can be further modified by adding user-defined costs for some specific assignment scenarios.

Allocation of Delivering Robots Sometimes it can be useful to not assign a task to a free robot but rather a robot that is currently delivering goods to a station. That is the case for scenarios where it is faster to assign a robot that finishes the delivering process of a good and drives to the first station of the assignable task than choosing a robot far away from the first station of this task. To ensure this beneficial behavior not only accepted orders are observed at the task allocation but delivering orders as well. The traveling costs used at the *global optimum* algorithm consist for the assignment of delivering robots of the following components:

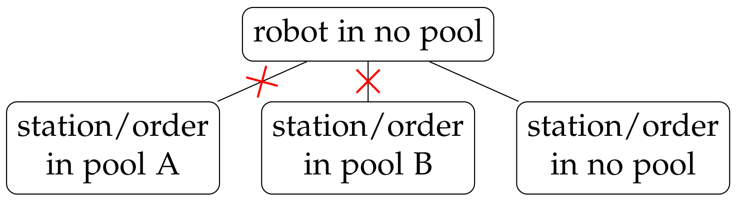
1. Euclidean distance from the current position of the delivering robot to the delivering goal
2. Euclidean distance from the delivering goal to the first station of the new task
3. user-defined additional costs for assigning a delivering robot

Allocation of Charging Robots In case a charging robots battery level reaches the active charge limit, the robot can leave the station for a new delivering process. To ensure that currently free robots are favored over robots in charging stations, so-called *charging costs* are introduced. These costs are set individually by the user and are used to modify the traveling

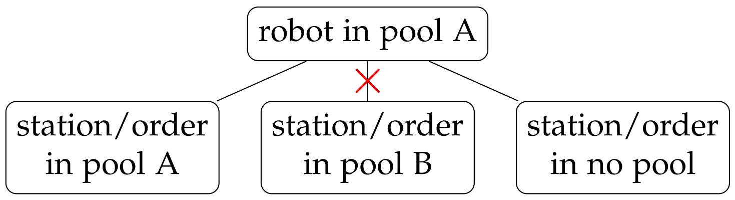
costs at the *global optimum* algorithm.
By adding the charging cost **cc** to the normal traveling costs it is ensured that a robot that is free and less than **cc** meters further away from the first goal of the assignable task than the charging robot, the free robot is preferred. Only if the free robot is more than **cc** meters further away from the first goal than the charging robot leaves the charging station and taking over the task.

Pools

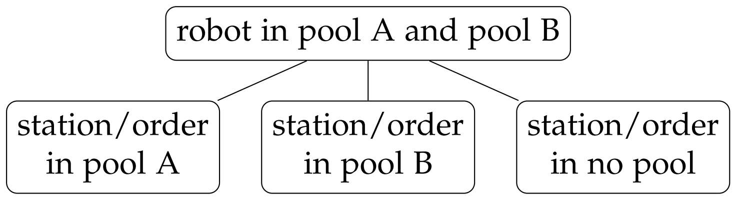
Where at BMW Group fleets exist at incubed IT pools split robots, orders, and stations in groups. At BMW Group every robot and every task is assigned to exactly one fleet. At incubed IT a robot can be assigned to multiple pools. By that the tasks and stations are split onto groups only accessible to robots assigned to the same pool. Pools are for example used to allow only assignments of robots to charging stations that have the correct docking system for the robot type.
The allowed assignments of vehicles to orders and stations, depending on their pools, are described by three examples shown in figure 4.5. If a robot is in no pool he can only be assigned to orders and stations in no pool. If a robot is in one or more pools, he can be assigned to orders and stations that are in the same or no pools.



(a) Robot assigned to no pool



(b) Robot assigned to one pool



(c) Robot assigned to two pools

Figure 4.5.: Allowed assignments for different pool combinations at incubed IT

5. Motivation for an ASP-based Intralogistics Management

The BMW Group and incubed IT currently face both the same challenge in the development of an assignment algorithm for autonomous robots and their tasks: the need for increased complexity and scalability.
The development of autonomous robots started in small scenarios. With the progress of development, the driving area is split into different sectors in where only some of the robots are allowed to drive. The strategy of charging the vehicles is adapted to ensure a battery-saving charging and a sufficient high number of available vehicles for assignable orders. These and other environmental conditions lead to a significant increase in the complexity of the fleet management system that is used to manage all robots.
The second challenge both companies currently face is scalability. The increase of robots managed simultaneously leads to an increased runtime of the task assignment problem solving that has to be considered during the development process. The runtime of an imperative program depends directly on the implementation by the programmer. Experienced computer scientists can find fast algorithms, but the development of these algorithms takes a long time and leads to a significant increase in the development cost. Using the declarative ASP system Clingo, a programmer describes the task assignment problem but not the solving steps to find a solution. The problem solving is overtaken by the internal solver Clasp. The runtime of the program depends mostly on the selected solving algorithm and heuristics. A solving approach with a good performance can be found faster by testing different solving settings for Clasp than implementing different solving algorithms in imperative programs.
The benefits of ASP have been shown in different research projects and publications. In [50] Clingo shows better results for the solving of task

assignment problems with many constraints than imperative methods. Furthermore, the computational costs can be reduced using ASP [18]. With the use of different solving approaches, like the heuristics in [1], a fast and simple improvement of the results is expected.
In the next chapter we present components of the existing Fleet Management Systems at BMW and incubed IT that were replaced by an ASP encoding and solving. As the Potassco system Clingo showed promising results in related research topics this solver suite will be used. The complexity of the existing fleet management systems increased strongly by adding constraints for the task assignment problems. It is expected that using Clingo instead of an imperative method the code complexity decreases and the performance increases as in [50]. A significant difference in declarative methods in comparison to imperative methods is the description of the problem itself rather than the description of the problem-solving process. Currently, a huge effort is required to advance the runtime of the fleet management system. With the use of different solving approaches of Clasp, the required time for performance improvement is expected to be decreasing.

6. Implementation of ASP in the Fleet Management System

The aim of this master thesis is to improve the Fleet Management Systems at BMW and incubed IT. Existing imperative described task assignment solving algorithms are replaced by an ASP-based program. Thereby the ASP system Clingo is used.
In [20] development steps are introduced that allow to introduce an ASP-based task assignment into an existing system. The steps are the following:

1. Identify the needs
2. Design a valid specification of the problem
3. Performance engineering
4. Integration into the existing environment

An ASP is implemented in both Fleet Management Systems of BMW and incubed IT following these steps. As both systems describe different environments with various constraint and distinctive optimization criteria, the development steps are handled separately for both companies.

6.1. Identification of Needs

ASP is known to be a suitable solution for problems that are described by rules and are facing mainly search and optimization challenges [8]. These rules are on one hand constraints and on the other hand optimization criteria.
Referring to [20] in a first step an existing system has to be analysed. Problems that are not satisfactorily solved in the given systems have to be

identified and the requirements needed to solve the problem need to be defined.

6.1.1. The BMW Use Case

In order to find the currently not satisfactorily solved problems in the BMW Fleet Management System the tasks of the system are identified and improvement possibilities are derived.
One task of the FMS is the validation and proof for correctness of the input data. For example, if the parameters of a charging station indicate to be booked by a vehicle for a long time, but no vehicle arrived, it can be assumed that the input parameters for this charging station are incorrect. In such a case the FMS unbooks the station so that the station is available for all other vehicles again.
Another task of the FMS are database requests and modifications. The data of all vehicles as well as all assignable tasks and stations are stored in the database. Because the FMS maps the assignment process, it requests in a first step all needed informations, such as available robots and unassigned orders, from the database. After the assignment the FMS publishes the data with modified parameters, like the booking status and availability, back in the database.
The third task of the FMS solves an assignment problem that is a typical search and optimization challenge. Given a set of vehicles, tasks, and stations, the goal is to assign the vehicles optimally under consideration of different constraints.
Analyzing all tasks the FMS is dealing with, the following application fields for the ASP can be found. The database requests, parameter checks, and the update of the database and parameters is done very efficiently in the existing C# encoding. However, under consideration of increasing complexity, it is expected that the task assignment problem of the FMS can be improved using ASP instead of imperative methods. Given up-to-date data of the vehicles, tasks, and stations this data can be adapted for the use within an ASP-based task assignment. The adapted data is passed to the declarative program, solved in there and as a last step passed back to the FMS. The FMS then publishes the modified data to the database.

As mentioned in [20] the application requirements have to be documented properly. In this ASP program an optimal assignment of vehicles to tasks, parking places, and charging stations has to be found. As the assignment to tasks and the assignment to charging stations and parking places is dealt separately in the FMS, it will be handled the same way using ASP by generating two programs. One ASP program is facing the optimal assignment of tasks. Another program is assigning vehicles to charging stations and parking places. Within the FMS first the task assignment program is executed. In a second step the park and charge assignment program is executed. By that it is made sure that available vehicles are more likely assigned to tasks than to charging stations and parking places.

Task Assignment

Assuming that all available vehicles and unassigned tasks are given, the optimal task assignment can be described by following rules and optimization criteria.

Rules for the Task Assignment At BMW the mission strategy *FIFO* is applied, which means that earlier created tasks have to be executed first. By that the criterium for the selection of tasks, formulated as a constraint, is not to assign a task if there is another appropriate task with earlier creation time assignable.
The rules for the selection of vehicles are the following. The task and the vehicle that are assigned must be in the same fleet. Vehicles on the field must have a battery level at a minimum of 25% and charging vehicles a battery level of 40% to be assigned to tasks (see table 4.1).

Optimization Criteria for the Task Assignment The optimal assignment of vehicles to tasks is based on the traveling costs that are set to be the Euclidean distance between robots and the first goal of the assigned task. The used optimization criterion ensures the lowest traveling cost for the tasks with earliest time of creation.

Park and Charge Assignment

In case that after the task assignment algorithm was executed unassigned vehicles are remaining, because they are in a different fleet than all open missions or less open tasks than free vehicles a given, the free vehicles are assigned to charging stations and parking places. Like before, the application requirements for the optimal assignments of the vehicles, now to parking places and charge stations, can be defined by rules and optimization criteria.

Rules for the Park and Charge Assignment The rules used for this assignment problem are defined separately for vehicles on the field and vehicles currently in charging stations (see also 4.1).
A charging vehicle can only be assigned to a charging station if the battery level is below 90%. Vehicles on the field can be sent to charging stations any time, regardless of the current battery level. Charging vehicles can go to parking places only if the battery level is above or equal 90%, whereas vehicles on the field can go to parking places independent from the battery level.
The assignment of vehicles to charging stations and parking places depends on one more constraint: only if both, the vehicle and station, are in the same fleet, an assignment is possible.

Optimization Criteria for the Park and Charge Assignment Like for the task assignment the distance between vehicles and POIs has to be minimized. Vehicles with the lowest battery levels should always drive to the closest charging station and parking place.
Vehicles should only charge if there is no other vehicle that could use the charging station as well and has a lower battery level.
Third, there should be assigned as many vehicles as possible to charging stations. If all charging stations are in use, as many vehicles as possible shall be sent to parking.

6.1.2. The incubed IT Use Case

Like for BMW in a first step the functions of the incubed IT FMS are analyzed and implementation possibilities are derived.
For a successful task allocation valid input orders are required. The FMS validates all orders for correctness and completeness, as the orders are user-generated and required parameters could be set incorrectly.
The FMS modifies the parameters and requirements of orders and vehicles at specific time intervals. For example, the FMS is increasing the order priority due to the ageing of orders and sets a boolean vehicle parameter to enforce the assignment of vehicles to charging stations at a specified time interval.
Another task of the FMS is the management of traffic interferences. If vehicles detect a traffic interference this error is sent to the FMS and there solved by different traffic rules. This management indicates another topic the FMS is responsible for, namely the communication with external systems, like warehouse management systems or conveyor systems, and the fleet itself.
The last task of the incubed IT FMS is a typical search and optimization challenge with many constraints. This task is the optimal assignment of vehicles to open orders, available charging stations and parking places.
Analysing the overall system it is seen that the FMS manages many topics where the Java based imperative approach is perfectly suitable. For example the modifications of orders and vehicles regarding the ageing of orders and time-depended charging should not be replaced by an ASP-based approach. These modifications depend strongly on the current system time that is easier accessible in Java.
Where ASP could provide a significant benefit is the replacement of the current assignment algorithm of vehicles to orders and stations. Due to many different constraints that affect this assignment problem the ASP-based encoding could give a benefit in the implementation of new constraints. As the runtime of the existing implementation is expected to reaches its limit for very high scaled application areas, it is hoped that ASP will bring benefits in relation to this topic as well.

Task Assignment

For an optimal task assignment all assignable vehicles, open orders, currently delivering orders and finished orders that are predecessors of open orders have to be provided by the FMS.
The optimal assignment is defined by some rules and optimization criteria. At incubed IT two different optimization algorithms are used: *FIFO* and *global optimum*. While the orders are sorted by the time of creation in *FIFO* they are sorted by a priority in the *global optimum* algorithm. By analysing these two algorithms in detail the *FIFO* algorithm can be seen as a special form of the *global optimum* algorithm. A priority can be set for every order based on the creation time of the order, where for an earlier creation a higher priority is selected. In the thesis we will use ASP to implement only the *global optimum* algorithm.

Rules for the Task Assignment The most important rules that affect the task assignment is to assign only one vehicle to every order and to assign only one order to every vehicle.
Available vehicles can only be assigned to orders if the battery level is above the busy charge limit (see figure 4.4).
If the vehicle is not assigned to a pool every order can be delivered by that vehicle. If the vehicle is part of one or more pools, it can only be assigned to orders that are in the same pool or in no pool.
In case an order with a predecessor is given this order can only be assigned if the predecessor is part of the list of all finished orders. All tasks of a parallel order have to be assigned to the same robot.
In the FMS not only available vehicles are considered for the assignment but also currently delivering vehicles. In case the distance a robot travels to finish the current mission and go to the origin of a new mission is shorter than the distance for a free vehicle to the origin of this new mission, the delivering vehicle will execute this mission instead of a free vehicle.

Optimization Criteria for the Task Assignment For a fast and efficient assignment as many orders as possible should be assigned to free and delivering vehicles. For an efficient delivering process the traveling costs have to be minimized. These traveling costs of the assignment sets depend not only on the Euclidean distance but also on two user-specified costs. In case a delivering robot is assigned to an order, the traveling costs is the sum of the Euclidean distance and a user-defined delivery-cost parameter to model unpredictable delivery delays. The same applies for vehicles that are currently charging but with the new assignment leaving the charging stations. The traveling costs are the sum of the Euclidean distance and the user-specific charging-cost parameter to model the impact of the break of a charging process.

Park and Charge Assignment

The FMS provides one list with all robots assignable to charging stations and parking places, one list with all free parking places, and one list with all available charging stations.
Vehicles that are not assigned to orders, whether because of a low battery level or because there are currently less appropriate open missions than free vehicles on the field, can be assigned to parking and charging stations.

Rules for the Park and Charge Assignment The overall park and charge assignment problem at incubed IT is broken down to 5 assignment sets:

1. **Fixed time slot charging:** robots are assigned to charging stations due to a reached time slot.
2. **Critical charging:** robots are assigned to charging stations due to a battery level below the critical charging limit.
3. **Busy charging:** robots are assigned to charging stations due to a battery level below the busy charge limit.
4. **Idle charging:** robots are assigned to a charging station due to not enough appropriate assignable tasks.
5. **Idle parking:** robots are assigned to parking places under two conditions. First due to a very high battery level and no appropriate assignable tasks. Second due to no available charging stations.

Every charging station and every parking place has a priority. As stations with higher priority have to be assigned first it is not allowed to assign stations if there are appropriate unassigned stations with a higher priority.

Optimization Criteria for the Park and Charge Assignment For an optimal park and charge assignment traveling costs have to be minimized. In contrast to the task assignment this time the optimization is split. First the overall traveling cost of all vehicles that are assigned to charging stations is optimized, ignoring thereby the idle charging vehicles. The remaining charging stations and parking places are used to find the optimal assignment of idle charging and idle parking vehicles. This split optimization ensures the smallest possible assignment costs for all robots that have to go urgent charging.

6.2. Design and Validation of Problem Specifications

After the identification of needs and naming the application requirements, the task assignment problem can be modeled using ASP.

6.2.1. Fundamental Design

The fundamental design of the ASP-based solving process is shown in figure 6.1. Firstly we define the following input data sets that are provided by the FMS.

- R: set of j robots $r_1, \dots, r_j \in R$
- T: set of k tasks $t_1, \dots, t_k \in T$
- S: set of l stations $s_1, \dots, s_l \in S$

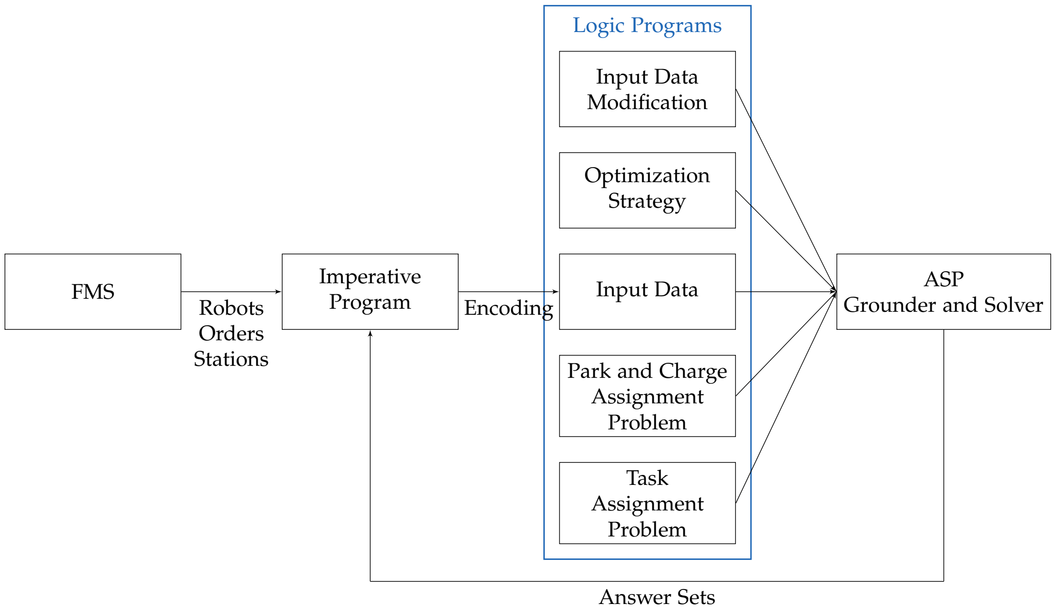


Figure 6.1.: Fundamental design of the ASP-based problem solving

The FMS provides the input data objects $r \in R$, $t \in T$ and $s \in S$ to an imperative program. The ASP-based problem solving design contains some logic programs. One program is containing all input data as facts and is provided by the imperative program. This program changes for every execution of the ASP-based algorithm. The other programs are the same for all executions and describe the assignment problem and optimization. All logic programs are passed to the ASP system. After solving the assignment problem in ASP the resulting answer sets are passed back to the imperative program.

Imperative Program

In an imperative program the provided data objects $r \in R$, $t \in T$ and $s \in S$ are mapped into strings. The strings are properly set to be identified as facts from the ASP grounder. A detailed description of the mapping and required data adaptions for the identification as facts are given in chapter 6.4.
The format of the grounded facts that are provided to the ASP grounder

is shown in listing 6.1. All robots $r \in R$ have the same preamble and the identical number of terms. The terms are represented by different variables for every robot and map robot-individual parameters that are required in the task assignment problem description.

Listing 6.1: Input data as grounded facts

```
robot(Parameter_1,Parameter_2,...,Parameter_m).
task(Parameter_1,Parameter_2,...,Parameter_n).
station(Parameter_1,Parameter_2,...,Parameter_o).
```

The tasks $t \in T$ and stations $s \in S$ follow the same syntax.

Input Data Modification

This program is used for a pre-processing of input data. The goal is to reduce the number of facts and atoms used in the encodings for the description of the assignment problems and optimization goals. The atoms that are always used together are joint to one atom. The terms of atoms required only in some rules are removed from the original atom and stored in a new one.

Task Assignment

Assume a nonempty set of robots R and a nonempty set of tasks T provided by the FMS. The aim of the task assignment problem solving is to generate a collection A of assignments $a_1, \ldots, a_p$. Therefore in a rule a task t is assigned to a robot r and mapped as fact (see listing 6.2). In the rule all parameters of the robots and tasks as well as the literals of the new fact *assign(T,R)* are described by variables.

Listing 6.2: ASP encoding of the task assignment - assignment rule

```
  0{assign(T,R) : robot(R,PR1,PR2,PR3)}1  :- task(T,PT1,PT2).
```

Some assignments of the set A are eliminated by constraints that depend on the implementation requirements of BMW and incubed IT. With the constraints the assignment set A is reduced to the set $B \subset A$. In listing 6.3 an exemplary constraint is shown, restricting the answer sets to have only task assigned to every robot.

Listing 6.3: ASP encoding of the task assignment - constraint

```
:- assign(T1,R), assign(T2,R), T1 != T2.
```

Park and Charge Assignment

Assume a non-empty set of robots R and a non-empty set of stations S provided by the FMS. The aim of the park and charge assignment problem solving is to generate a collection C of assignments $c_1, \ldots, c_q$ of robots to stations. Therefore in a rule a robot r is assigned to a station s and mapped as fact (see listing 6.4). In the rule all parameters of the robots and stations as well as the literals of the new fact *charge(S,R)* are described by variables.

Listing 6.4: ASP encoding of the charge assignment - assignment rule

```
0{charge(S,R): station(S,PR1,PR2)}1 :- robot(R,PR1,PR2,PR3).
```

With constraints some possible assignments c are eliminated and the assignment set C reduces to the set $D \subset C$. In constraints where all parameters are valid the terms of the input facts are defined by variables. For constraints where only some parameters of the input facts are accepted these terms are described by facts.
In listing 6.5 an exemplary constraint is shown allowing only robots in automatic mode with a battery level below 30% to be assigned to a charging station. With the constant term *automatic_mode* only robots in this mode are considered in the constraint. All terms of the atoms that are not required in

the constraint are replaced by an underscore to provide a better readability of the constraint.

Listing 6.5: ASP encoding of the charge assignment - constraint

```
:- charge(_,R), robot(R,automatic_mode,_, Batterylevel),
    Batterylevel >= 30.
```

Optimization Strategy

To find the optimal assignment this program describes the optimization strategy of the assignment problem. Given are facts F with new assignments. These facts are provided by the ASP encodings of the task assignment and the park and charge assignment. In different optimization statements of the ASP language the facts F are used to describe the optimization strategy.

6.2.2. Design at BMW

In the original implementation the overall assignment is split in a task assignment and a park and charge assignment. This structure is maintained in the new implementation and thereby two ASP programs are set up. In one ASP program robots are assigned to tasks, in the second program the robots are assigned to charging stations and parking places.
The newly generated ASP programs require input data that is provided by the FMS. The FMS requests this data of orders, robots, and POIs from a database. Knowing the basic structure of the input data and the needed parameters for the assignment problem solving the required input data for the ASP programs can be defined. Although the exact implementation of the parameter processing from the FMS to the ASP grounder is done in chapter 6.4.2, the structure of the provided input data is defined in this development step. The structure of the input data for the task assignment is shown in listing 6.6, the structure for the park and charge assignment is shown in listing 6.7.

Listing 6.6: Input data for the BMW task assignment

```
mission(missionID1,fleetID1,20190727182952,getstationID1).
getstation(getstationID1,169,232).
robot(robotID1,fleetID1,86,false,737,379).
```

Every mission contains the parameters mission-ID, ID of the assigned fleet, time of creation and ID of the first station of the mission. Every get station contains the ID of the station and the position of the station on the map. The parameters that are stored for every robot are the ID of the robot, the assigned fleet, the battery level, the information whether the robot is in a charging station or not, and the current position of the robot in the map as x- and y-coordinates.

Listing 6.7: Input data for the BMW park and charge assignment

```
parkingstation(idlestationID1,fleetID1,939,112).
chargingstation(dockstationID1,fleetID1,172,960).
robot(robotID3,fleetID1,69,false,130,32).
```

The provided parameters for robots that are used in the park and charge assignment are the same as for the mission assignment. The parking places and charging stations contain information about the ID of the station, the corresponding fleet and the position on the map.

Task Assignment Algorithm

The task assignment is split in three different ASP programs:

- program to modify the input data
- program to find possible answer sets under constraints
- program to select the optimal answer set of assignments

As all three programs are executed simultaneously, the grounder has access to all facts of the different programs. The passing of the input parameters to the various functions, as is necessary for imperative programs, does not have to be considered here.

Program for the Input Data Modification The structure of the input data, like the one in listing 6.6, is very similar to the one of the FMS. To reduce the number of predicates and terms in the task assignment programs and provide by that a better readability and performance of the code, the provided data is modified.
To reduce the number of atoms describing robots and their parameters, only robots with an appropriate battery level for the task assignment are stored as new atoms and are later on used in the task assignment program. To reduce the number of atoms in the task assignment program further, all assignable tasks are provided with the x- and y-coordinates of the get station. By that the atoms of POIs are not required in the task assignment program.
To reduce the number of terms in every atom the input data is further modified. Some terms in the robot and task atoms, like the fleet-ID, are required in only some rules. The input data modification program stores the robots and tasks as new atoms without a term with the fleet-ID. Instead all pairs of robots and tasks, that are in the same fleet and can be assigned, are stored as new atoms containing only the ID of the robot and the task. By that the number of terms per atom for robots and tasks is reduced, but the information of atoms is still accessible by the new fact.

Program for the Task Assignment Algorithm The first rule implemented in this program is the general assignment of a robot to a task. The answer set is then restricted by allowing only assignments if the robot and task are in the same fleet. A constraint prevents the assignment of one robot to more than one task. To ensure the FIFO-assignment the constraint that is shown in listing 6.8 is set up. The assignment of a robot *S* to a task *T2* is allowed only if there is no other appropriate unassigned task *T1* with an earlier time of creation *P1*. The unassigned task is appropriate if it is in the same fleet as the robot.

Listing 6.8: Constraint for the FIFO strategy

```
:-task_free(T1), assign_mission(T2,R1),
  assignabletask(T1,P1,_,_), assignabletask(T2,P2,_,_),
  flotte_ident(T1,R1), T1 != T2, P1 > P2.
```

Program for the Optimization In the task assignment two optimizations are required: the Euclidean distance reduction and the increase of assigned missions. Both criteria are implemented in the optimization program.
During the implementation a first restriction using declarative methods occurred. In the current system the tasks are stepwise assigned to the robots. The earliest created task is assigned to the closest robot, the second task is assigned to the closest of the remaining robots, and so on. As in ASP the assignment problem is solved simultaneously for all missions this stepwise optimization can not be applied. Therefore, the Euclidean distance for all assignments is calculated and summed up. The optimal assignment is found to be the set of assignments with a minimized sum of all traveling costs. In chapter 7.2.1 a detailed discussion about the quality of these two assignment algorithm is provided. A second optimization criterion is set up that is maximizing the number of new assignments.
Using both optimization criteria equally weighted would not lead to the optimal solution. By use of the criterion to minimize the overall Euclidean distance the best answer set is an empty set. To avoid this faulty behaviour the optimization criteria are weighted. The most important criterion is the increase of assigned missions. The resulting answer sets are then optimized concerning the lower weighted distance minimization criterion.

Park and Charge Assignment

As for the task assignment, the overall system is split into three programs: One for the input data modification, one for the general assignment problem and one for the optimization.

Program for the Input Data Modification To reduce the number of terms in atoms that are required only in some rules, these terms are removed from the original atom and stored as new one. Hence the information of the fleet is stored in new facts containing pairs of robot-IDs and station-IDs that are in the same fleet and can be assigned. The modified input data for all robots and stations used in the assignment program are by that reduced by one term in the atoms.
In the current state this program does not have much influence on the overall system, but with an increasing complexity of the FMS in the future this program will be required for some more modifications.

Program for the Park and Charge Assignment The rules that have been defined in the previous chapter (see chapter 6.1.1) are implemented in this program. First the assignment of robots to charging stations is defined. Two different rules are required, as currently charging robots (indicated by the Boolean true as third term of the literal *robot_chargeable*) can only be assigned to charging stations until a battery level of 90% is reached.
The assignment of robots to parking places looks similar. Again, two different rules are necessary. The selection of charging and not charging robots that are allowed to go to parking places is restricted by battery levels.
The number of possible assignment sets is reduced by adding the constraint to only assign robots to stations in the same fleet.

Program for the Optimization To ensure to fulfill all optimization criteria different rules are defined in the optimization program. In comparison to the task assignment in this program more optimization criteria have to be implemented.
Most significant is the increase of number of robots assigned to charging stations. Second-most important is the assignment to parking places. As robots shall be assigned only if there is no robot with lower battery level, a third optimization criterion is the minimization of the sum over the battery levels of all assigned robots. Last but not least the distance of the assigned robots to POIs has to be minimized.

6.2.3. Design at incubed IT

To solve the assignment problems the ASP grounder has to be provided with input data. The FMS saves every order, robot and POI with many parameters, from whom some are only needed for navigation or the loading behavior, but not for the general assignment of robots to POIs.
The basic structure of the data in the FMS is known. For example there exists a list with all delivering robots, a list with all assignable orders and a list with all charging stations. The exact data transfer from the FMS to the ASP grounder is described in chapter 6.4.3. However, input data is needed for the implementation and testing of the ASP programs. Therefore the input data format that is used in the ASP encoding is already defined in this section. Based on the rules described in the previous chapter, the parameters of the individual robots, orders and POIs required for the ASP problem description can be derived. An example of provided input data in ASP is given in listing 6.9.

Listing 6.9: Input data for the incubed IT assignment

```
opentask(taskID1, 4, taskID5, none, poolID1, stationID1).
deliveringtask(taskID10, robotID1, stationID2)
finished_predecessor(taskID5,robotID4).

getstation(stationID1, poolID1, 10, 12).
putstation(stationID2, poolID1, 47 ,16).
chargingstation(stationID3, 50, none, 18, 16).
parkingstation(stationID4, 10, poolID1, 55, 23).

robot_assignable(robotID4, 76, false , 86, 2, 35).
robot_chargeable(robotID2, 48, automatic_mode, none,
59, 7, 70, 35, 30).
robot_delivering(robotID1, 43, 53, 32, 35).

robot_pool(robotID2,poolID1).
```

Every open task contains the parameters ID of the task, priority of the task, ID of the predecessor, ID of a parallel task of a chained order, ID of the

pool and the ID of the get station of the task. Delivering tasks contain the ID of the task, the ID of the robot who is delivering and the ID of the put station. Finished predecessor are provided as a fact with information about the ID of the finished predecessor and the ID of the delivering robot. In the given example shown in listing 6.9 the assignable task with ID *taskID1* has a predecessor with the ID *taskID5*. This predecessor is finished, as a fact named *finished_predecessor* with the predecessor ID is provided.
All stations are represented by information about the ID of the station, the pool and the x- and y-positions. For charging stations and parking places an additional parameter that is placed right behind the station ID sets the priority of the station.
For every robot specific battery levels can be defined. As these battery levels are required for the correct assignment, they are passed to the ASP program as parameters. The facts for assignable robots provide information about the ID of the robot, the battery level, a Boolean value indicating the currently charging robot, the x- and y-position and the battery level for the busy charging limit. The chargeable robots contain information about the ID, the current battery level, an information if the robot is in idle mode or automatic mode, an information about specific charge decisions (none; balance charging; fixed timeslot charging), the current x- and y-position, and the active, busy and critical battery levels. Delivering robots provide the same information as the assignable robots, only the Boolean information if the robot is currently charging is not required.
For every pool a robot is assigned to, a fact is set up with information of the robot ID and the ID of the pool.
To implement the incubed IT assignment strategy with all predefined rules and optimization criteria four programs are set up. One for the modification of the given input data, one for the task assignment algorithm, one program for the assignment of robots to parking places and charging stations and one program for the fulfilling of the optimization criteria.

Program for the Input Data Modification

The structure of the ASP input data, like the one in code snippet 6.9, is very similar to the structure in the FMS. To reduce the number and size of atoms in every rule in the assignment programs the given input data is modified and simplified.

Input Tasks The input data provides all assignable tasks with information about the ID of the tasks, the priority, the ID of the predecessor and others. To reduce the number of considered atoms in the assignment program, the open tasks that can be assigned to a robot are set as new facts if they hold following conditions:

1. if an input task has a predecessor ID, the task is only assignable if this predecessor ID is finished.
2. for orders with a parallel-ID only one of the parallel executed tasks is considered in the task assignment algorithm. If the task is assigned to a robot, all other tasks of the chained order are assigned to this robot.

Applying the two conditions the terms that contain information of predecessors and parallel IDs are no longer required.
To reduce the number of atoms used at the task assignment algorithm the ID of the get station, that is saved as parameter of the task, is replaced by the x- and y- coordinates of this station. By that only the atoms describing tasks are used in the task assignment, but not the atoms for get stations, as the only needed information, the position of the station, is now stored as term of the tasks.
Next to the input data for open tasks the data for currently delivered tasks are modified. The atoms for tasks in delivery are required for the assignment of currently delivering robots to new tasks. The provided input data parameters for tasks in delivery are the IDs of the task, the ID of the delivering robot and the ID of the put station. To reduce the number of atoms in the ASP encoding of the assignment problem, the ID of the put station is replaced by the x- and y-position of this station. With that the atoms of stations are not required in the assignment program.

Input Robots The input data provides two different lists with robots. In one list all *assignable robots* are stored. These robots are currently free or charging and can be assigned to tasks. All robots which can be assigned to charging stations and parking places are maintained in the list called

chargeable robots. As some robots can deliver orders and go charging they are found in both lists.
As assignable robots can only be assigned to a task if the battery level is above the active charge limit (for charging robots) or above the busy charge limit (for not charging robots), they are used in the assignment program only if they have an adequate battery level.
Chargeable robots are split in the input data modification program into three different facts:

1. Balance charging robot: a chargeable robot that has to go balance charging
2. Forced charging robot: a robot that has to go timeout charging
3. Optional charging robot: all robots that are assignable to charging stations but do not need to go forced charging

Pools One rule for the assignment of robots is the consideration of pools. Robots that are in a pool can be assigned to tasks in the same or in no pool. Robots with no pool can be assigned only to orders with no pool. The input data provides information about robots and related pools. In listing 6.7 this information is provided by fact named *robot_pool(S,P)*. In case the robot is in no pool, no fact is provided. To provide the required data to the task assignment problem the given robot-pool relations are extended to facts named *pool_ident*. For every allowed assignment of robots to orders regarding their pools such a fact is set up. To circumvent the not provided information of robots that are in no pool, the constraints as shown in listing 6.10 are set up. In the first row all assignment sets, that have the same pool *P* are registered in the fact *pool_ident*. In the second rule all assignments are accepted where the robot is in a pool and the task is in no pool. The last rule accepts all assignments in where the task and the robot are in no pool. Following the principle of default negation the rule holds if no information of *robot_pool* is given for the robot *S*.

Listing 6.10: Rules for pool-restricted assignments

```
pool_ident(R,T) :- robot_pool(R, P),
            assignabletask(T,_,_,P,_,_).
pool_ident(R,T) :- robot_pool(R,PoolID),
            assignabletask(T,_,_,none,_,_).
pool_ident(R,T) :- robot(R),
            assignabletask(T,_,_,none,_,_),
            not robot_pool(R,_).
```

Program for the Task Assignment Algorithm

After having all required rules and optimization criteria documented in section 6.1.2 the ASP-based task assignment can be set up. The required open orders, the list of all finished predecessors and all assignable and currently delivering robots are provided by the input data modification program.
During the development first any desired robot is assigned to orders. Subsequently additional rules and constraints are added to the ASP encoding to result in a program that returns only answer sets that are fulfilling the rules of section 6.1.2.
In the first rule that is set up any desired robot is assigned to an assignable task and saved as the literal *assign(T,S)*, where the term T contains the ID of the task and the term S the name of the robot. The resulting assignment sets are reduced by the following constraints. Only assignment sets are accepted where the pool rule is fulfilled. Every robot can be assigned to only one order. In case a finished predecessor is given, the robot that delivered this chained task has to deliver the next chained task without an interruption. The implementation of this rule using two constraints is shown in listing 6.11. The first constraint prevents the assignment of a robot *S*, that was used in a chained order to be assigned to the task *T*. This task is not part of a chained order, as the parameter *none* is set in the field for the predecessor ID. The second constraint in listing 6.11 prevents the assignment of a robot *S1* to a task *T* if this task has a predecessor *TPD* that was executed by the robot *S2* .

Listing 6.11: Constraints for predecessors

```
:-assign(T,R), assignabletask(T,_,none,_,_,_),
    finished_predecessor(_,R).
:-assign(T,R1), assignabletask(T,_,TPD,_,_,_),
  finished_predecessor(TPD,R2),  R1 != R2.
```

To assign an order to a robot that is currently in delivery another rule is set up. In the head of this rule, named *assign_to_delivering*, the ID of the new order, the name of the delivering robot and the total traveling costs are saved as literals. The number of found solutions is reduced by some constraints to ensure that not more than one order is assigned to the delivering robot and that only appropriate tasks are assigned. Furthermore, another constraint allows the assignment to a delivering robot only if there is no other free or currently charging robot with lower traveling costs.
In the rule for the assignment of open tasks to free robots only one task of an parallel order is considered. To provide not only this assignment to the FMS but the assignments of all parallel tasks an additional rule is set up. If a task of a parallel order is assigned to a robot, all associated tasks, indicated by the same parallel-order-ID term, are assigned to this robot as well.
To avoid a scheduling of tasks, the tasks that are assigned to delivering robots are not passed to the FMS. Instead these assignments are ignored for the moment. The task assignment algorithm is executed very regulary. As long as the robot is still in delivery, the new task will be assigned to this delivering robot but the assignment is not passed to the FMS. When the robot finished the delivery process the new task is assigned to the robot again and this time passed as part of the answer set to the FMS.

Program for the Park and Charge Algorithm

In the section for rules for the incubed IT park and charge assignment (see section 6.1.2) it is stated that for the park and charge assignment four different assignment statements are used. One statement assigns robots that have to go fixed timeslot charging, one statement assigns all robots with a battery level below the critical charge limit and a third rule assigns robots to idle charging stations. The implementation of the fourth statement, the assignment of busy charging robots, is shown in listing 6.12. The body of the rule holds for robots that are in the automatic mode and not in the idle mode. Furthermore, the battery level has to be between the critical and the busy charge limit. In the head a condition literal is set up, assigning no or one chargeable robot to a charging station and store the assignment in the fact *robot_charge*. The head and by that the assignment holds only if the charging station and the task are in the same pool.

Listing 6.12: Assignment of busy charging robots

```
0{charge(S,R busy) : chargingstation(S,_,_,_,_),
     robot_station(R,S)}1 :-
robot_charge_opt(R,BL,automatic_mode,_,_,_,BCL,CCL),
BL <= BCL, BL > CCL.
```

To assign only one robot to a station and only one station to every robot the constraints in listing 6.13 are defined.

Listing 6.13: Avoidance of double allocations

```
:- charge(S,R1,_), charge(S,R2,_), R1 != R2.
:- charge(S1,R,_), charge(S2,R,_), S1 != S2.
:- assign(T,R1), charge(_,R2,_), R1 = R2.
```

The constraints only hold for answer sets in where no two robots are assigned to the same station (first constraint), no two stations are assigned to one robot (constraint 2) and a robot is not assigned to a charging station and a task simultaneously (third constraint).

Program for the Optimization

Until now the ASP solver published multiple possible answer sets as solutions. These answer sets fulfill all given rules, but only one of the sets is the optimal solution. To reduce the number of possible answer sets to one optimal solution, the ASP optimization program is implemented.
Therefore, optimization criteria with different weights are defined. The most important optimization criterion and by that the highest weighted criterion is the maximization of the number of robots assigned to orders, charging stations and parking places. The second most important is to increase the sum of all priorities: The priority of orders as well as the priority of the assigned charging stations and parking places. The last optimization criteria are the minimization of traveling costs for urgent charging robots and robots that are sent idle charging or to a parking place.

6.2.4. Validation of Problem Specifications

After modelling the task assignment problems using ASP the correctness of the results has to be tested. Therefore multiple test scenarios are set up. The test scenarios map different combinations of robots, tasks and stations in different settings, like multiple fleets or various battery levels. All robots and stations are placed on a 50 x 50 coordinate system where the robots are freely movable. Test scenarios are for example the following:

- given are a set of available robots and a set of open tasks. All robots are in one fleet (at BMW) or pool (at incubed IT), all tasks are in another fleet or pool. This scenario is used to test the functionality of the constraint for fleets and pools.
- given are a set of robots and a set of open orders. Due to the low battery levels of the robots they can not be assigned to the open orders. This scenario tests the consideration of battery levels in the task assignment.
- given are more robots that have to go charging than available charging stations. With this scenario it is tested whether robots with a lower battery level are rather assigned to the charging stations than robots with a higher battery level.

The optimal solution for every test scenario was found by hand. To test the correct functionality of the ASP implementations the answer sets returned by the ASP solver are checked against the known optimal solution.

6.3. Performance Engineering

After implementing the ASP programs and testing the correct functionality in different test scenarios it is mentioned in [20] to have a look at the runtime of the programs. For the implementation of ASP-based solving approaches at incubed IT and BMW it is not only important to have a readable code that can be maintained and extended easily, but also to have a program with a runtime comparable to the existing system. Therefore, the performance of the newly implemented ASP systems is tested for different scenarios, analysed and improved. This process is very complex and time-consuming, as all improvements have to be tested of a correct functionality. The impact of the improvements has to be evaluated and further steps must be considered. The ASP implementations of the task assignment problem of both companies follow the same structure. For given orders, vehicles and POIs optimal assignment sets have to be found under consideration of different constraints. The optimization goal is the same for both companies, namely to minimize the traveling costs and maximize the number of assignments. Although the constraints of both companies are slightly different, the programs follow the same basic structure. Evaluating the performance improvements for one system, these improvements can simply be applied to the other system. Therefore, the code performance improvements are only evaluated for the incubed IT system, as this one is slightly more complex and has more optimization parameters. After that the founded improvements are applied to the ASP programs of BMW.
Not only the code structure itself has an effect on the runtime of the program, but also the selected ASP solver. For the improvements in encoding the different number and complexity of rules and constraints at BMW and incubed IT did not make a difference. For the selection of solving approaches, however, the number and complexity of rules and constraints influence the runtime of the solving algorithm, as some solving approaches are suitable for many different constraints, some approaches are better for

problems with fewer constraints. Therefore, solving approaches are analysed separately for the system of BMW and incubed IT.

6.3.1. Testing Environment

The systems of BMW and incubed IT have been tested on devices with the following specifications. At BMW an Intel(R) Core(TM) i5 with a 1.70GHz processor and 8GB RAM is used. At incubed IT an Intel(R) Core(TM) i5-7200U is used with a 2.50GHz processor and 8GB RAM. On both systems Windows 10 is installed. Clingo is running in version 5.3.0 with Gringo V5.3.0. and Clasp V3.3.4.

6.3.2. Encoding Improvements

For a successful improvement of the system, the limiting code elements have to be found first.
While testing the correct functionality of the ASP encoding during the development progress an observation was made which can now be useful for the performance improvement. Following two scenarios have been given to validate the correctness of the program:

1. Given are five vehicles, all with a high battery level at 85%, and two open missions. All robots and tasks are in the same fleet.
2. Given are five vehicles, all with a high battery level at 85%, and two open missions. All robots are in different fleets (robot 1 and robot 2 in fleet A, robot 3 in fleet B, and robot 4 and 5 in fleet C), the tasks are in fleet A (task 1) and fleet B (task 2).

These test scenarios were used to evaluate whether the constraint for fleets works or does not. The optimal solution with lowest traveling costs for the scenario with only one fleet is the assignment of robot 2 to task 2 and robot 5 to task 1. In the second test scenario the output should not be this assignment set, as task 1 can only be assigned to robot 1 and robot 2 (both in fleet A) and task 2 can be assigned only to robot 3 (both in fleet B).
During the test scenario evaluation not only the code correctness has been

proved, but runtime differences of the two scenarios have been recognized, although the number of input parameters stayed the same. The runtime of the second scenario was significantly faster. Due to the restriction of answer sets by fleets only two possible answer sets had to be optimized, in contrast to the first scenario where 20 possible answer sets had to be optimized. It was shown that the runtime is influenced significantly by the number of possible answer sets the optimization function has to analyze. This indicates, that a system with mostly constraints can be solved faster by the solver than a system with many optimization criteria. The same observation was found in chapter 3.2.3 by [53], where with the use of teams of robots the performance increased.
With this observation the main focus along the performance engineering will be the replacement of optimization criteria by rules. To analyze and improve the runtime first the task assignment and after that the park and charge assignment will be improved. Different problem scenarios are observed to evaluate the impact of the encoding improvements.
After every change in the ASP encoding it is tested if the correct solution is found. Therefore all test scenarios with provided correct solutions, introduced in paragraph 6.2.4, are solved by the new ASP encoding and the obtained results are compared.

Task Assignment

To analyze the improvements of code modifications different test scenarios are set up. The test scenarios represent differently scaled environments and are shown in table 6.1. The number of assignable tasks ranges from 3 to 30 elements, the number of robots is set to two-third of the number of tasks. All robots and open orders are in the same fleet and all robots have a battery level of 90%. Every open task has its own get station. The robots and get stations are randomly placed on a 400 x 400 coordinate system. The robots can move freely in the environment. For the calculation of travelling costs the Euclidean distance from the position of the robot to the get station of the assigned task is considered. In these test scenarios no charging stations and parking places are given.

Test Scenario	Robots	Open Tasks
Scenario 1	2	3
Scenario 2	6	9
Scenario 3	10	15
Scenario 4	20	30

Table 6.1.: Test scenarios for the performance evaluation of the task assignment algorithm

When using Clingo to solve a problem in ASP a parameter can be set (named *–stats*) to indicate that after the solving process detailed statistics of the grounding and solving process are shown in the terminal window. One of these statistic information is the number of constraints that have been affecting the solving process, another parameter shows the number of models found. With a higher number of constraints, the number of founded models from where the optimal solution has to be found decreases. As a lower number of models and higher number of constraints indicate an improve of the runtime these two parameters will be, next to the runtime, evaluated to find an optimal program.

Evaluation of the Original Encoding As a first step of the performance improvement the original encoding of the assignment problem, that was generated by following the generate-and-test methodology, has to be evaluated. In table 6.2 the runtime, number of models and constraints is shown for every test scenario introduced in table 6.1. As a solving process with a long runtime is not acceptable for a real time implementation in the existing system, the solving process is stopped after 30 seconds.
An exponential time increase for the solving process can be seen in the table.

Test Scenario	Runtime [ms]	Models	Constraints
Scenario 1	31	5	18
Scenario 2	1019	39	414
Scenario 3	Timeout	/	/
Scenario 4	Timeout	/	/

Table 6.2.: Results for the task assignment test scenarios with the original encoding

With 9 instead of 3 open tasks the runtime increases from 31 milliseconds to more than 1 second. Instead of 5 answer sets at scenario 1, 39 answer sets are analysed and optimized in the solving process of scenario 2. Finding an optimal answer set for scenario 3 and 4 is not possible within the time limit.

Code Improvement 1: Removing Optimization Criteria 'Number of Missions' As a first performance improvement of the code the optimization criterion to increase the number of assigned robots is replaced by constraints. Instead of maximization statements, as seen in code snippet 6.14, the constraints in code snippet 6.15 are used to maximize the number of assigned robots to tasks.

Listing 6.14: Optimization criterion for maximization of assignments

```
#maximize{I@8, C,S,I :new_assigned_task(T,S,C,I)}.
```

All tasks that are not assigned to free or currently delivering robots are set as the fact *task_free*. All robots that are assignable, but not assigned and not in a charging station or parking place, are set as the fact *robot_free*. All free robots and all unassigned orders that are in the same pool are counted. In case the number of robots is higher than the number of assignable tasks, the first constraint holds and is fulfilled if the number of open tasks is zero. In case the number of robots is smaller than the number of open orders, the second constraint must hold and does so only if the number of free robots equals zero.

Listing 6.15: Constraint for maximization of assignments

```
:-N1 = #count{S1:robot_free(S1),  robot_task(S1,TA)},
  N2 = #count{TA:task_free(TA)}, N1 >= N2, N2 != 0.

:-N1 = #count{S1:robot_free(S1),  robot_task(S1,TA)},
  N2 = #count{TA:task_free(TA)}, N1 < N2, N1 != 0.
```

A further improvement of the encoding, that does not only effect a better readability but also a better performance, is the reduce of different literals that are required in the different constraints and rules. This reduction is achieved by the implementation of the program for input data modification. The replacement of the optimization statement and the reduction of required constraints leads to a better performance of the program. As it can be seen in table 6.3 the runtime for scenario 1 increased, as for very small problem scales the optimization is solved faster than the fulfilment of additional constraints. The reason for that is the increased required time for the grounding process of the program due to the higher number of constraints in the code.
The runtime of all other scenarios decreased, especially for scenario 3 and 4. They can now be solved in less than 30 seconds. It can also be observed that with an increasing number of models the solving process starts to increase exponentially. The runtime per model in scenario 3 is 0.003 seconds, whereas in scenario 4 this is 0.08 seconds. This exponential behaviour of the runtime shows the importance to reduce the number of found models to improve the performance. With a few more found models the runtime increases significantly.

Test Scenario	Runtime [ms]	Models	Constraints
Scenario 1	114 (+ 268%)	3 (- 40%)	85 (+ 372%)
Scenario 2	144 (- 86%)	25 (- 36%)	943 (+ 127%)
Scenario 3	267	78	2388
Scenario 4	18755	234	10901

Table 6.3.: Results for the task assignment scenarios with the first encoding improvements

Code Improvement 2: Distance Constraint To reduce the number of models even further, the impact of the second optimization criterion, the minimization of the traveling costs, is reduced. The criterion can not be replaced completely by a constraint, but the number of possible answer sets that have to be optimized can be reduced. Two new constraints are introduced and shown in code snippet 6.16.

Listing 6.16: Traveling costs constraints

```
:-assign(T1,S1), task_free(T2), robot_task(S1,T2),
 assignabletask(T1,P1,_,_,_,_), cost(T1,S1,C1),
 assignabletask(T2,P2,_,_,_,_), cost(T2,S1,C2),
P1<=P2, C1 > C2.
:-assign(T1,S1),assignabletask(T1,P1,none,_,_,_),
 robot_free(S2), robot_task(S2,T1),
 cost(T1,S1,C1), cost(T1,S2,C2),  C1 > C2.
```

The first constraint forbids the assignment of tasks if there is an unassigned appropriate task that can be assigned to the same robot but has lower traveling costs. In the second constraint the assignment of a task to a robot is prevented if there is another free and appropriate robot with lower traveling costs. Additionally to the new traveling constraints the performance is increased by removing all unused terms in atoms that have been added during the development of the ASP encoding but are not required any more. For example the priority of a task was given as term in the atom of a new assignment, but was never used. Another performance improve showed up to be the modification of rule for assigning delivering robots. The atoms used for distance calculation are moved from the body to the head of the rule.
The impact of the code modifications is shown in table 6.4. For scenarios with a smaller number of robots and tasks the runtime increases slightly due to the higher number of constraints and the by that longer grounding time. The runtime for scenario 4 decreased significantly by a factor of 95%. The number of models decreased for all 4 scenarios by a factor of 67% (scenario 1) up to 94% (scenario 4).

Test Scenario	Runtime [ms]	Models	Constraints
Scenario 1	139 (+ 22%)	1 (- 67%)	94 (+ 11%)
Scenario 2	145 (+ 0.1%)	3 (- 88%)	1294 (+ 27%)
Scenario 3	387 (+ 45%)	11 (- 86%)	3423 (+ 43%)
Scenario 4	850 (- 95%)	14 (- 94%)	25300 (+ 132%)

Table 6.4.: Results for the task assignment scenarios with the second encoding improvements

Park and Charge Assignment

To evaluate the runtime differences, that occur during the improvements for the park and charge assignment, 4 test scenarios are set up and shown in table 6.5. The test scenarios differ in the number of charging stations and parking places and the number of robots that have to go fixed timeslot charging or threshold-based charging. All robots and stations of these test scenarios are in the same pool. The position of robots and stations is set randomly in a 100 x 100 coordinate system. The robots are freely movable in this system.
As in no open tasks are provided in the scenarios, not only the robots with a battery level below the busy charge limit and robots that have to fixed timeslot charging are assigned to charging stations, but all remaining robots are assigned to the charging stations and parking places as well.

		Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**
Charging Stations		1	2	5	10
Parking Places		2	6	10	20
Robots	Fixed Timeslot Charging	0	1	2	4
	BL[1] <CCL[2]	0	1	3	5
	BL $\leqslant$ BCL[3]	1	2	3	5
	BL >BCL	1	2	2	6

Table 6.5.: Test scenarios for the performance evaluation of the park and charge assignment algorithm

Evaluation of the Original Encoding To compare the encoding improvements in a first step the performance of the original encoding is observed. In table 6.6 the runtime, number of models and the number of constraints is shown.

[1]Battery Level
[2]Critical Charge Limit
[3]Busy Charge Limit

Test Scenario	Runtime [ms]	Models	Constraints
Scenario 1	97	3	14
Scenario 2	162	18	302
Scenario 3	Timeout	/	/
Scenario 4	Timeout	/	/

Table 6.6.: Results for the park and charge assignment test scenarios with the original encoding

Results for the first two scenarios are found within milliseconds, but it is shown that for higher scaled problem instances like the one in scenario 3 and scenario 4 the optimal solution can not be found within the 30 seconds time limit.

Encoding Improvements The replacement of optimization criteria for the park and charge assignment turned out to be more complex than for the task assignment. Due to the increased number of assignment sets for forced charging, critical charging, busy charging and idle charging the replacement of the optimization statements by rules turned out to be more difficult. Instead of replacing each optimization criterion by one or two rules, this time the overall system is observed and described by some known and new rules as followed:

- a robot with a fleet needs to be assigned to a POI with the same fleet (constraint already implemented)
- a robot with no fleet can be assigned to any POI (constraint already implemented)
- there can be no robot not charging that needs to charge when there is a robot with a higher battery level in an appropriate charging station, except where forced and balance charging robots are in the appropriate charging station.
- another robot can not be charged in a station if a robot requires forced charging and could be charged in this station
- there can be no robot requiring forced charging and an appropriate charging station is free
- there can be no robot charging in a station if a balance charging robot is unassigned and could be assigned to this station

- a charge station cannot be free if there is an appropriate robot with a battery level below the idle charge limit and unassigned
- a parking place can not be free if there is an appropriate robot that is not assigned to a task or a charging station
- charge station 1 cannot be assigned to robot B if the unassigned robot A exists and robot B has another optional free charge station it can be assigned to
- parking place 1 cannot be assigned to robot B if the unassigned robot A exists and robot B has another optional free parking place it can be assigned to
- vehicle A cannot go charge at parking place 2 if parking place 1 is free, closer and appropriate for robot A

All listed rules are formulated in ASP as rules and constraints. The optimization criterion to maximize the number of assigned robots is not required any more. The two optimization criteria to minimize the traveling costs for urgent charging assignments and the minimal costs for idle charging and parking assignments are still required. The performance of the modified ASP encoding is shown in table 6.7.

Test Scenario	Runtime [ms]	Models	Constraints
Scenario 1	99 (+ 2%)	1 (- 67%)	29 (+ 107%)
Scenario 2	130 (- 20%)	29 (+ 61%)	446 (+ 48%)
Scenario 3	185	47	1799
Scenario 4	1171	416	9691

Table 6.7.: Results for the park and charge assignment test scenarios with the improved code

The performance improvements are very significant for test scenario 3. Whereas in the original ASP encoding the optimal solution was not found within the time limit, in the new implementation the solution is found in 185 milliseconds. Test scenario 4 can now be solved within the time limit as well. The performance improvements for the first and second test scenario are not as significant. It can be seen that the number of found models for scenario 2 even increased compared to the original implementation. The reason for that could be the behavior of the default solver that runs in the background. The underlying algorithm is possibly more suitable for the problem description in the original encoding.

Further Improvements To improve the performance of an ASP encoding Gebser et al. [31, pp. 153–173] introduce advanced modeling approaches. Based on a newly generated ASP code, the performance of this code is evaluated with some provided statements. Better performance of the ASP solving process can be achieved by revising the current encoding until all 9 statement in [31, p. 172] hold. The ASP implementation of incubed IT and BMW Group holds most of the statements, but further improvements could be desired. One improvement that could accelerate the encoding is the improvements of optimization criteria that are dependent on each other [31, p. 170]. The main idea of this improvement could be applied to the minimization of the total traveling costs in a later ASP version.

6.3.3. Selection of Solving Approaches

Clasp, the answer set solver of Clingo, provides options to modify the solving process. A selection of these options has already been introduced in chapter 2.3.3.
In close cooperation with professor Martin Gebser, a co-developer of Potassco, following solving approaches are examined that could improve the given ASP solving processes:

1. Opt-heuristic to alter the sign selection (command - *-opt-heuristic=1*)
2. Berkmin heuristic (command - *-heuristic=Berkmin*)
3. Vmtf heuristic (command - *-heuristic=Vmtf*)
4. Vsids heuristic (command - *-heuristic=Vsids*)
5. Unit heuristic (command - *-heuristic=Unit*)
6. Arbitrary static ordering (command - *-heuristic=None*)
7. Domain heuristic with the domain heuristic to apply a false statement to all atoms that appear in optimization statements (command - *-heuristic=Domain - -dom-mod=5,8*)
8. Branch-and-bound-based hierarchical optimization (command - *-opt-strategy=bb,1*)

9. Competition-based multithreading with two threads (command - -*parallel-mode=2,compete*)
10. Competition-based multithreading with four threads (command - -*parallel-mode=4,compete*)
11. Splitting-based multithreading with two threads (command - -*parallel-mode=2,split*)
12. Splitting-based multithreading with four threads (command - -*parallel-mode=4,split*)

For the evaluation, the solving approaches 1 to 8 are analysed first. The best-found solving approach is then tested with the four different parallel-mode selections, as multi-threading can be used together with other solving approaches.
Test scenarios are set up separately for both companies. The number of assignable robots and tasks is chosen to be in a range where an optimal solution can be found within seconds for the default solving approach. To test the runtime of the different solving approaches, ten test runs with different input data are set up for the test scenarios.
In the following the optimal solving approaches are evaluated for BMW and incubed IT.

Selection of Solving Approaches at BMW

At BMW the task assignment and the park and charge assignment are executed in two separate executions. That allows to select different solving approaches for both assignments. Two evaluations are done to find the best approach that is indicated by the fastest solving time.

Solving Approach for the Task Assignment The test scenario for the performance improvement of the task assignment is the following. Given are 5 open tasks and 12 vehicles. All tasks and robots are in the same fleet and placed randomly in a 1000m x 1000m field. The time of creation is set randomly for every order, the battery level of the robots is set in a range of 25% to 100%. The input data with random variables is generated for ten test runs. To evaluate the best solving approach the performance of the different

solvers is observed for every of these 10 test runs.
The performance of the first 8 solving approaches, evaluated for the ten different test runs, is shown in a boxplot in figure 6.2. The underlying data is given in table A.2 in the appendix. For test runs where the solving processes takes more than 60 seconds a timeout is reached and the solving process is stopped. These test runs are not considered in the calculations for the boxplot and the table in the appendix.

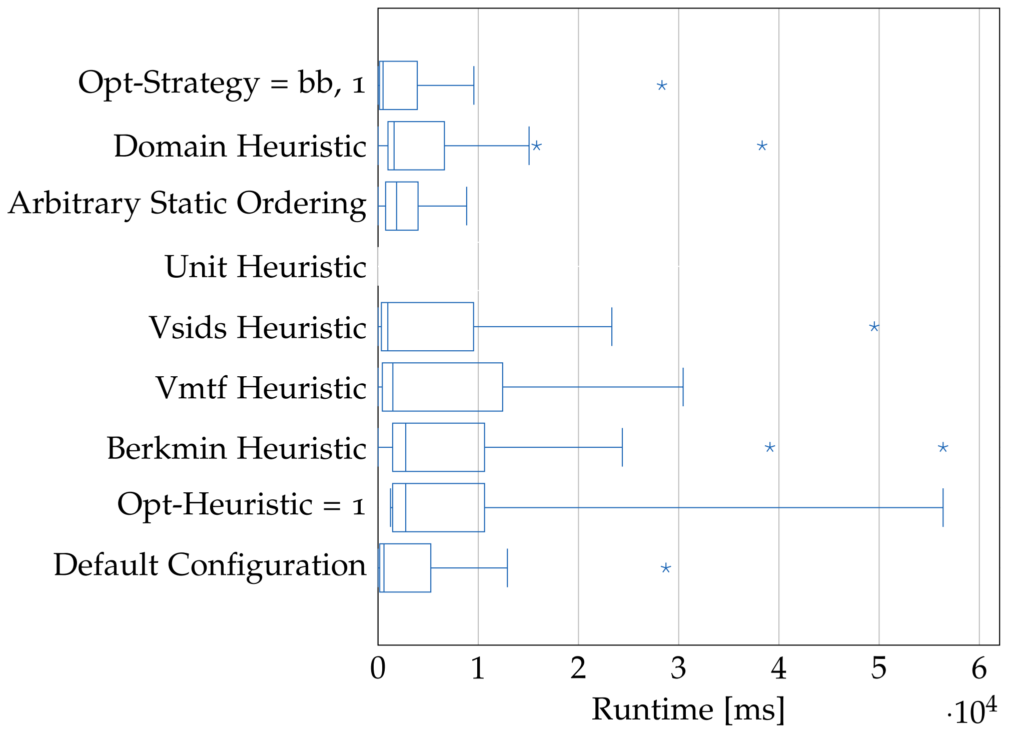


Figure 6.2.: Boxplot of the performance results of the BMW task assignment for different solving approaches

To evaluate the best-suited solving approach the runtimes for the different methods are analyzed. The solving approaches with the smallest boxes and the lowest upper whiskers in the boxplot in figure 6.2 are the default configuration, the arbitrary static ordering and the branch-and-bound-based hierarchical optimization. As the median in the boxplot as well as the mean

and the standard deviation in the table show the best performance for the branch-and-bound-based hierarchical optimization this strategy is chosen to be the best suiting solving approach.
In the next step the branch-and-bound-based optimization strategy is used together with different multithreading approaches to solve the 10 test runs again. The results are shown in figure 6.3. The underlying data is given in table A.1 in the appendix. The smallest box, median value and whisker in the boxplot and by that the best performance for the test scenario is given by the splitting-based approach with 4 threads. Additionally this approach is the only one without outliers.

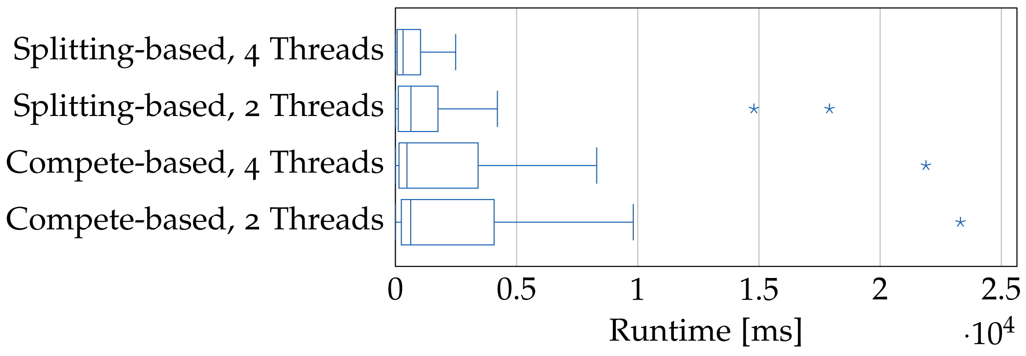


Figure 6.3.: Boxplot of the performance results of the BMW task assignment for different multi-threading approaches combined with the branch-and-bound-based optimization strategy

The best solving approach for the BMW task assignment is the branch-and-bound-based optimization strategy in combination with splitting-based search multithreading and four threads.

Solving Approach for the Park and Charge Assignment As a first step a test scenario is set up. This test scenario consists of 33 parking places, 17 charging stations, and 15 assignable vehicles. All stations and robots are in the same fleet and randomly placed in a 1000m x 1000m field. If different fleets would be used the number of found models from where the optimal solution has to be found, would be smaller and by that the runtime of the

solver would not only depend on the selected solving approach. The battery level of the robots is set in the range of 25% to 100%. Ten random test runs are generated and used to solve the park and charge assignment with the different solving approaches.
The performance of the first 8 solving approaches for the given test runs is shown in figure 6.4, the underlying data is provided in table A.4 in the appendix. For test runs where the solving processes takes more than 60 seconds a timeout is reached and the solving process is stopped. These test runs are not considered in the calculations for the box plot and the table in the appendix.

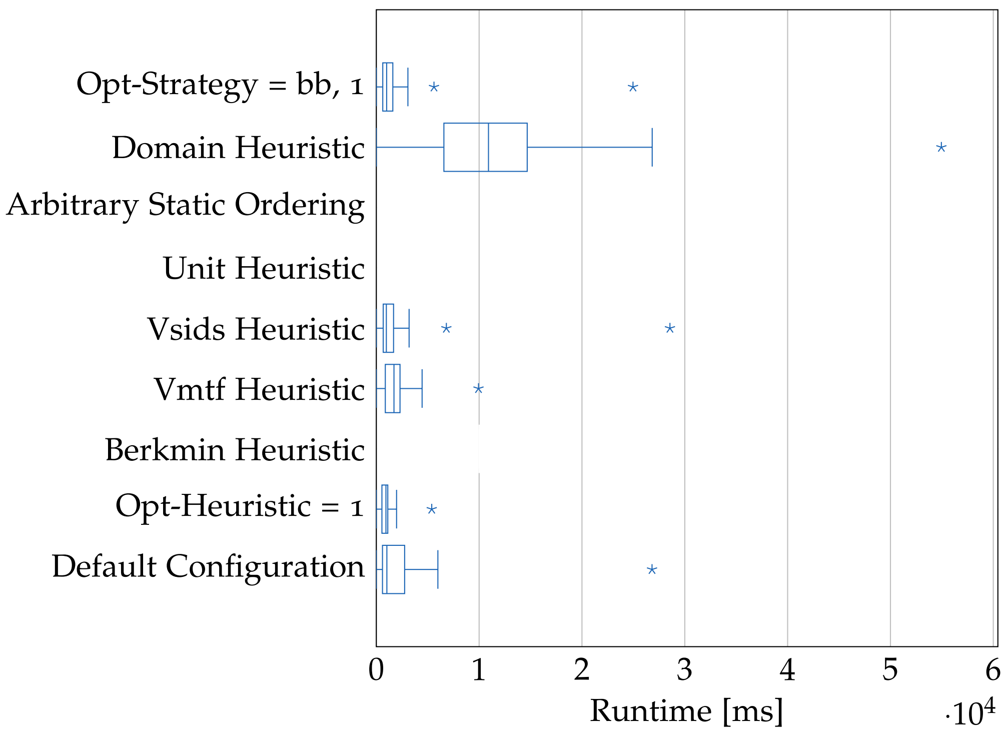


Figure 6.4.: Boxplot of the performance results of the BMW charge and park assignment for different solving approaches

Looking over the required runtimes to solve the test runs the four approaches default configuration, Vsids heuristic, Domain heuristic and branch-and-bound-based optimization strategy are the only approaches never reaching the time limit. All other approaches reach the time limit at least once,

with the Berkmin and Unit heuristic and with the approach of arbitrary ordering no test run was solved within 60 seconds. Looking at the box width and the median of the boxplot and analyzing the mean and standard deviation of the runtimes in table A.4 the branch-and-bound-based optimization strategy shows the best performance. This strategy is the best fitting approach for this assignment problem as well as already for the task assignment problem.
Now the best parallel-mode is analysed. Therefore the branch-and-bound-based strategy is used with the multithreading-based solving approaches. The runtimes are shown in figure 6.5, the underlying data is given in table A.3 in the appendix. The splitting-based multithreading with four threads shows the lowest upper whisker, the lowest median and the smallest box width. Based on these observations it can be stated that the park and charge assignment at BMW is solved the fastest by using the branch-and-bound-based optimization with splitting-based multithreading and four threads.

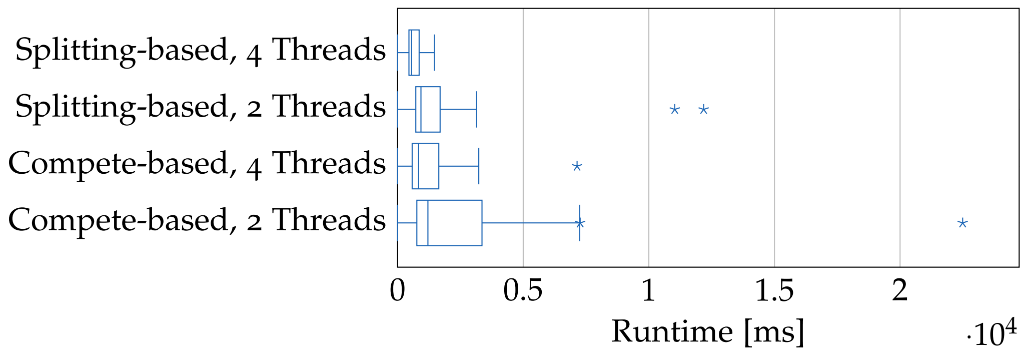


Figure 6.5.: Boxplot of the performance results of the BMW charge and park assignment for different multi-threading approaches combined with the branch-and-bound-based optimization strategy

Selection of Solving Approaches at incubed IT

To find the optimal solving approach for the assignment problem at incubed IT a test scenario is defined. In this scenario 15 robots with battery levels between 40% and 99% are placed randomly in a 100m x 80m field. The robots are optimally assigned to 20 tasks. Like for BMW, 10 test runs are

set up for the given test scenario. For every test run the vehicles, stations, and get stations are randomly placed on the map. By that it is ensured that solving approaches are not selected that fit only one specific test run.
The runtimes for the solving approaches 1 to 8 are shown in figure 6.6. The underlying data is shown in table A.6 in the appendix. If the optimal solution is not found within the incubed IT-specific time limit of 30 seconds, the solving process is aborted. These aborted test runs are not considered in the calculations for the boxplot and the mean and standard deviation in the appendix.

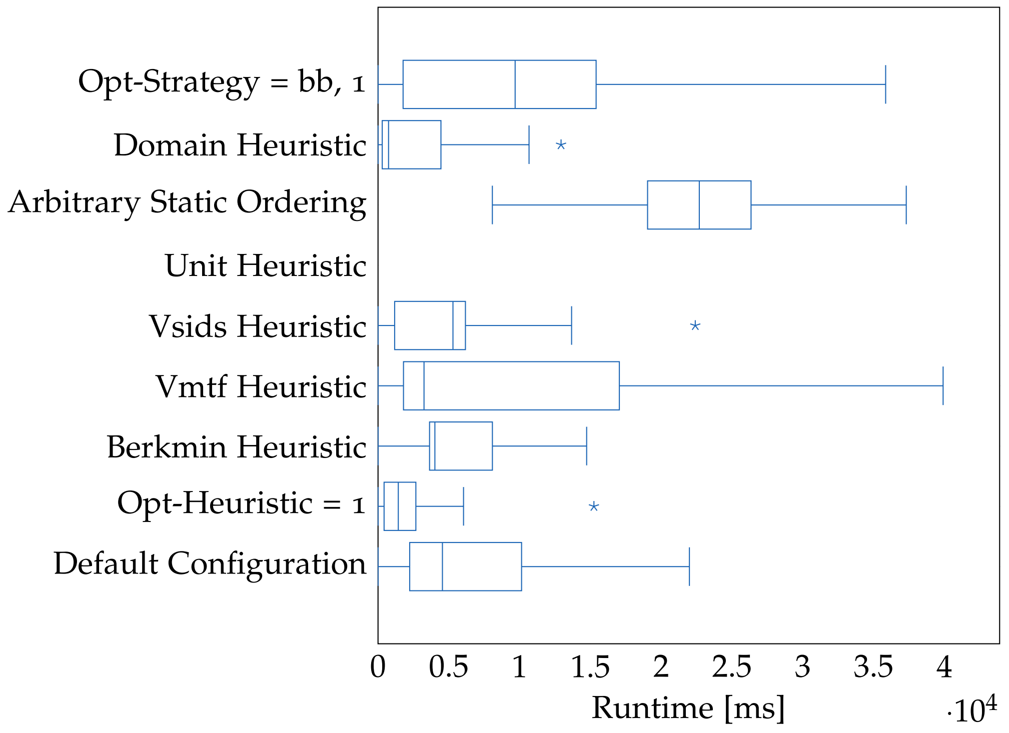


Figure 6.6.: Boxplot of the performance results of the incubed IT assignment for different solving approaches

It can be seen in table A.6 that with the default configuration, the Vsids heuristic and the branch-and-bound-based strategy all test runs, out of test run 4, are solved in time. All other solving approaches reach the time limit at least for two additional test runs. Analysing the performance of the three solving approaches with only one timeout the optimal solving approach is selected based on the runtime. The Vsids-heuristic shows the smallest box width and lowest upper whisker in the boxplot in figure 6.6. It is the solving approach with the best runtime to solve the test scenario and thereby selected to be the optimal solving approach for the incubed IT assignment. In the next step the best multithreading setting is evaluated. The 10 test runs are solved with the Vsids-heuristic and the four previous introduced multithreading settings. The required runtime for the different settings is shown in figure 6.7, the data is provided in table A.5 in the appendix.

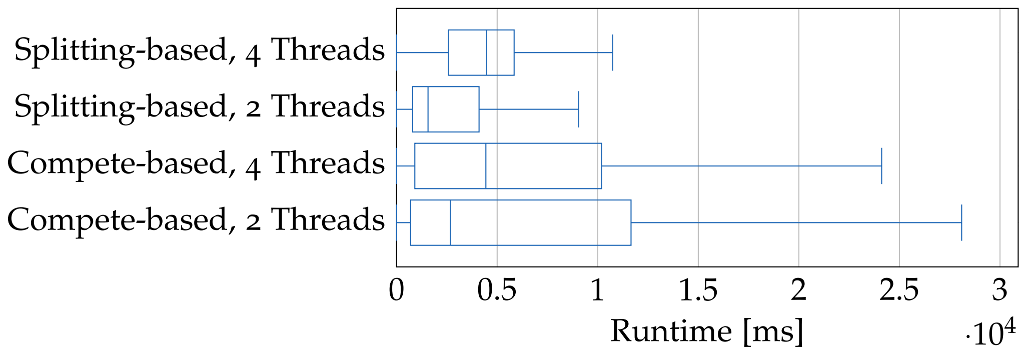


Figure 6.7.: Boxplot of the performance results of the incubed IT assignment for different multi-threading approaches combined with the Vsids heuristic

Again test run 4 is not solved within the time limit. The splitting-based multithreading can not find the optimal solution for test run 10 either (see table A.5). Looking at the two remaining solving approaches the compete-based multithreading with 4 threads shows the best performance, as in figure 6.7 this approach has a smaller box and by that closer distributed runtimes. Further the upper whisker is lower than the one for the solving approach with compete-based multithreading with 2 threads.
The optimal solving approach for the incubed IT assignment is the Vsids-heuristic combined with compete-based multithreading with four threads.

During the evaluation it was seen that test run 4 can not be solved by any of the solving approaches within the time limit. Analysing the position of the robots and the stations in the map it is seen that for test run 4, in opposite to the other test runs, the robots are placed mostly on the left and the stations on the right half of the map. Additionally some robots in the left half of the map and some stations in the right half of the map are placed very close to each other what makes it different to find an optimal solution, as the travelling costs are very similar for many assignments. A solution is found, but not after 30 seconds but more than 5 minutes of solving time for the default configuration.

6.4. Integration in Existing Systems

Until now the assignment problem described in the ASP language was solved calling Clingo with the different ASP programs and input data in an anaconda prompt. For the use in the real environment of the companies, the ASP programs have to be called from within the existing systems.

6.4.1. Decision for Python Modules

An problem described in ASP can not only be solved by calling it in an anaconda prompt but also with some APIs, like the one introduced in chapter 2.3.6. The two APIs supporting Potassco are the Python-API and the C-API. The Python built-in-module is better documented and easier to integrate than the built-in-module for C. As the Python script can be called from the FMS of both companies with less effort than a code written in C, especially at incubed IT, it is decided to use the Clingo built-in-module in Python to integrate the ASP programs into the existing systems.

6.4.2. Integration at BMW

In the original implementation of the FMS two management services are called. In the Job Manager the task assignment problem is solved and in the Fleet Manager the park and charge assignment problem is solved. Therefore in every management service a C# function is called. In the functions the

required data is requested from the database and the algorithm to solve the assignment problem is executed. The new assignments are published to the database.
In figure 6.8 the new software architecture of the ASP-based problem solving in the BMW FMS is shown. In C# the database handling and solving algorithm is replaced by a call of a Microsoft Azure Web Application. In this application a Python function is called. Within the Python function in a first step a database request is executed to obtain the input data needed to solve the assignment problem. The input data is modelled as a logic program following the stable model semantics. With the Clingo built-in-module for Python the task assignment problem and the park and charge assignment problem encodings are loaded into Python and the ASP-based solving is started, whereby the input data is provided as logic program. The ASP system returns answer sets to the Python function in where the new assignments are published to the database.

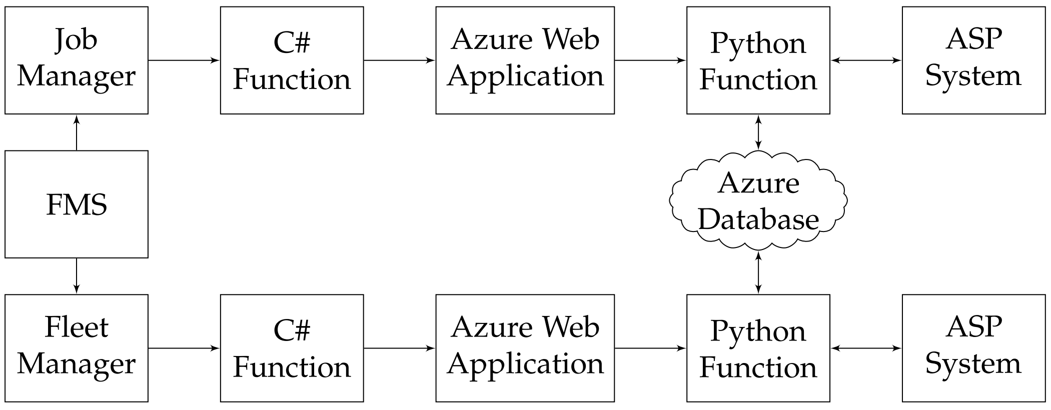


Figure 6.8.: Software architecture of the ASP-based assignment at BMW

In the next sections a detailed description of the input data modifications is given. The modifications are required to model the data in a lean logic program. Further the tasks of the Python function are explained more detailed.

Input Data for the Task Assignment

To solve the task assignment problem available vehicles and assignable missions have to be provided. After requesting the data from the database in a first step the received input data is shorted by removing all unused parameters. A parameter that is not needed for a successful task assignment is, for example, the software version of the vehicle.
Required vehicle parameters are:

- vehicle-ID
- fleet-ID
- battery level
- in charging station (Boolean)
- x- and y-position of the robot

Necessary mission parameters are:

- mission-ID
- fleet-ID
- date and time of creation
- ID of first station of mission

Needed station parameters are:

- station-ID
- x- and y-position of the station

The ASP input languages can only handle strings without decimal numbers, large initial characters, and punctuation marks. To fulfill these requirements the input data is adapted before passing it as facts in a logic program to the ASP grounder.

Input Data for the Park and Charge Assignment

To solve the park and charge assignment problem, all available vehicles and all parking and charging stations have to be given as input data. Like before in the task assignment only a few of the input data parameters are required to solve the assignment problem. Necessary vehicle parameters:

- vehicle-ID
- fleet-ID
- battery level
- in charging station (Boolean)
- x- and y-position of the robot

Necessary station parameters:

- station-ID
- fleet-ID
- x- and y-position of the station

Again, all decimal numbers have to be rounded and the hyphens of the IDs are removed. Furthermore, the modified input data is stored as a string.

Solving Process in Python

After the modification of the input data the ASP system can be called. Therefore, all ASP encodings are loaded into the Python file and the input data is added to the ASP grounder as logic program that contains only grounded facts. The Clingo solver returns the optimal answer set and provides it as a string in Python. In Python the string is modelled as list of assignments and these new assignment sets are published to the Azure database.

6.4.3. Integration at incubed IT

The Fleet Management System at incubed IT is based on a Java application. For the optimal assignment of vehicles to orders, charging stations and parking places an assignment function is integrated within this application. In the original imperative implementation the input data, stored in Java classes, is passed from the FMS to the assignment function. In the function the optimal assignment of robots to tasks, parking places and charging station is calculated. The resulting assignment sets are passed to the FMS as data stored in Java classes.
In figure 6.9 the new software architecture of the ASP system integrated in

the incubed IT FMS is shown. The existing Java-based assignment function is replaced by a new Java function. This Java function still receives the input data, stored in Java classes, from the FMS. Within this new Java function in a first step all orders, goals, and robots are saved in new classes that contain only the information that is needed for the ASP programs. In a second step a micro web framework, that provides all modified input data in a JSON format to a local IP-address, is called. The flask-module of a Python function is listening to this local IP-address. If new input data is provided by the web framework, the Python function transform this data from the JSON format to a logic program format following the stable model semantics. With the Clingo built-in-module for Python the ASP-based task assignment problem solving for the input data is started. The ASP system returns answer sets as strings to the Python function. In Python the answer sets are modelled in the JSON format and passed to the micro web framework. In Java the answer sets, received from the web framework in JSON format, are stored as new assignments in Java classes and passed to the FMS.

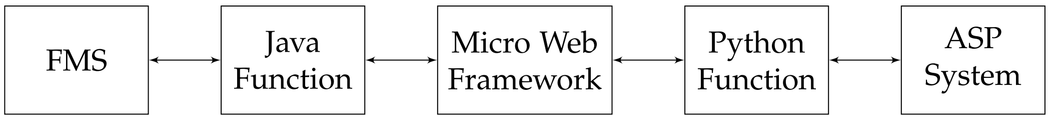


Figure 6.9.: Software architecture of the ASP-based assignment at incubed IT

Required Input Data

The data that is provided by the Java application contains much information that is required for the correct navigation and loading behaviour of the robot, necessary for the execution of orders and the maintenance of the different charging stations and parking places. However only a few of the provided parameters are required for the optimal assignment of vehicles. Therefore, in the newly generated Java function the input data is stored in new classes, containing only the elements that are required for the assignment algorithm. These parameters are listed below.

Required robot parameters are:

- name of the robot
- current position of the robot (x- and y-coordinates)
- list of all pools of the robot
- battery level
- special charging decisions (none, fixed timeslot charging, balance charging on next iteration)
- robot charge parameters
  - fully charged limit
  - idle charge limit
  - active charge limit
  - busy charge limit
  - critical charge limit
- robot currently in charging station (Boolean)

The robot data is stored in different lists, depending on the current state of the robot. These states are briefly introduced here:

- robots available for all assignments: robots in automatic mode which can be assigned to orders
- robots available for charge assignments: robots in idle or automatic mode which can be assigned to charging stations and parking places
- robots currently in delivery: all robots that currently deliver an order from the station of origin to the station of destination

For a successful assignment the following task parameter are required:

- ID of order
- robot the order is assigned to (only for orders currently in delivery)
- priority of the order
- name of the station of order origin
- name of the station of order destination
- pool of the order
- ID of the predecessor order
- ID of the parallel order

Like the robot data, the provided orders are stored in different lists, depending on their current state:

- assignable orders: orders that are accepted by the FMS and can be assigned to a robot to be delivered
- orders in delivery: orders that are currently delivered by a robot.
- predecessors: list of all finished orders that are predecessors of orders in the list of assignable orders

Required goal parameter:

- name of the station
- priority of the station
- pool of the station
- position of the station

All goals are stored in different lists for get stations (first station of a task), put stations (delivery station of a task), charging stations and parking places. During the process of saving all requested orders, robots and goals the input parameters are adapted to fulfill the ASP input language requirements. As the ASP grounder can only handle strings without decimal numbers, large initial characters and punctuation marks, the battery levels of the robots as well as the coordinates of the robots and goals are rounded. It is ensured to have only robot names, goal names and pool names with small initial characters so that the grounder identifies the input data as grounded facts. All just introduced lists are saved in a newly generated class. This class provides all data that is required for the task assignment and is used by the Web API to model the data in the JSON format.

Solving Process in Python

The ASP solving process is started in a Python script using the built-in-module for Clingo. A flask-module in Python access the input data that is provided by the micro web framework. The data is transformed from the JSON format into a string containing grounded facts for the ASP grounder. Python passes this string next all other ASP programs to the Clingo module and starts the grounding and solving process of ASP. After the Clingo process is finished and an optimal solution is found, this resulting answer set is published to the Python script as a string. From there the results are passed in the JSON format to the micro web framework to be accessible for the Java function. In the Java function the new assignments are stored in Java classes and passed to the FMS.

7. Evaluation

In chapter 5 the motivation for the replacement of parts of the imperative FMS at incubed IT and BMW by a declarative implementation was stated. Due to the growing number of autonomous robots that are used in intralogistics domains the relevance for an optimal assignment algorithm with high performance and maintainability increases. Using ASP instead of imperative methods benefits in the runtime of the solving process of the assignment problem were expected.
In this chapter the newly generated ASP programs are evaluated. It is looked over the overall implementation effort of replacing the current assignment algorithm with the ASP encoding. The performance of the ASP implementation is compared to the one of the original system as well as the quality of the obtained solutions.

7.1. Evaluation of the Integration Effort

When introducing a new programming approach not only the runtime and correctness of results are of interest, but also the effort of generating and integrating the new code in an existing system. This effort is in the following analysed for both companies.

7.1.1. Integration Effort at BMW

In this section the overall implementation effort for the different implementation steps introduced in chapter 6 are analysed. A software developer who is experienced in the BMW Group Fleet Management System and is familiar with ASP is considered to overtake the integration. The stated implementation efforts in this chapter are based on his experience.

Effort to Identify the Needs

ASP is known to be a good modeling and solving approach for applications that mainly face optimization and search problems. With a profound knowledge of the BMW Group FMS, problems with such optimization and search problems can be found within 1 hour. The requirements of these problems, namely the task assignment and the park and charge assignment, have to be documented in a second step. This documentation, in where all specifications are extracted from the code, can be finished in about one working day.

Effort for Design and Validation

Generating a new ASP encoding is known to be error-prone. Due to missing debugging-options the finding of mistakes and wrong implementations in the code is very hard. To reduce the effort of finding an error different test scenarios, introduced in paragraph 6.2.4, are solved after every change of the encoding. As for every test scenario the optimal solution is known the correctness of the ASP encoding can be validated.
While at the early stage of the ASP implementation with a few rules and constraints the finding of the error cause is fast, later on with many rules and constraints that depend on each other the complexity of finding the error cause increases significantly.
Due to the well-documented application requirements it is possible to generate a basic program for both assignment problems in about four days.

Effort of Performance Engineering

After the generation of the initial program the performance of the system is improved. This performance enhancement turned out to be significantly more complex than the development of the initial encoding.
The code improvements are analysed in the incubed IT problem descriptions. The performance-improving code components are applied at the incubed IT system as well as in the BMW system. The effort to find these improvements is described later in section 7.1.2. The integration of the improvements at BMW and the testing of correct functionality of the modified implementation required two working days.
After applying the code improvements different solving approaches are evaluated. With the ASP solver Clasp different approaches can be chosen that are more or less suitable for a given problem environment. To find the best approach for the BMW assignment problems different solving algorithms and heuristics were tested. The best suitable approach was found after one day of evaluation. In comparison to the required time for the code improvements the finding of appropriate solving approaches took only a fraction of the overall effort.

Effort of Integration in the Existing System

At BMW the original function call that starts the algorithm for the assignment problem solving is replaced by the call of a Microsoft Azure Application.
Therefore in a first step this Microsoft Azure Application was implemented. After that a Python script was written for this Azure Application. The integration of the Clingo API, the configuration of the solver settings and the modifications of the provided input data in Python took about one day. Another day was required to implement the Azure application call in C# and to modify the provided input data to fit the ASP language requirements.

7.1.2. Integration Effort at incubed IT

In the following the implementation effort at incubed IT is evaluated. A software developer that is experienced in the existing FMS at incubed IT and is familiar with the declarative language of ASP is considered to overtake the integration and implementation of the ASP. The stated implementation efforts in this chapter are based on his experience.

Effort to Identify the Needs

ASP is known to be a good alternative to imperative methods for incidences that face mainly search and optimization problems and are described by multiple rules [8]. The areas where ASP could bring a benefit in the existing FMS can be found by the software developer within 1 hour.
In a next step, the requirements of the new ASP-based implementation are documented carefully. As the software developer knows the FMS and has access to the FMS source code and a detailed documentation of the overall system the requirements, rules, and optimization goals can be defined in less than a day.

Effort for Design and Validation

The ASP encoding is developed by first assigning every order to a vehicle and then stepwise adding constraints. To validate the correctness of the implementation different test scenarios, introduced in paragraph 6.2.4, are set up where the optimal answer set is known.
For every newly implemented constraint and rule all test scenarios are solved and the answer set is compared to the provided optimal solution. As one additional constraint reduces the possible answer sets significantly, unexpected correlations with other rules and constraints can occur that result in a not optimal solution.
The overall task assignment problem encoding is set up and tested by the experienced ASP user within 4 days. Thereby most of the time is spent with the finding of errors and reasons why new constraints do not change the resulting answer sets as expected.

The implementation of the park and charge assignment encoding took less time than the implementation of the task assignment encoding. Reasons for that is the smaller number of required constraints and rules that decrease the effort of setting up the system and testing the correct execution.

Effort of Performance Engineering

To improve the runtime of the ASP implementation the initially implemented encoding is modified. Furthermore, different solving approaches are evaluated to find the best one suited for the given environment.
The most working effort for integrating an ASP program in the existing FMS turned out to be the improvement of the encoding to ensure a reduced runtime. Not only the modification and simplification of rules and constraints took some time, even more significant was the replacement of optimization criteria by constraints and rules. The first removed optimization criterion was the maximization of assigned orders. Different statements have been considered, implemented and tested on all test scenarios that have been shortly introduced in paragraph 6.2.4. The finding of the final implementation took about three days. The implementation of the additional traveling distance constraint to reduce the number of possible answer sets (see subsection 6.3.2) required about four days. The criterion to minimize the overall traveling distance was not possible to be removed in total, but in the end a constraint was found that reduced the number of possible solutions.
Particularly time consuming was the replacement of the constraint to maximize the number of robots assigned to charging stations and parking places. Using differently weighted optimization criteria, it was possible to ensure that robots more likely go charging than parking. It showed up that a similar implementation as the one for the task assignment was unrewarding. Instead, the overall assignment of robots to charging stations and parking places had to be reconsidered. With the help of two colleagues the overall problem was described only by rules and constraints. Starting from the assignment of robots to charging stations and parking places the possible answer sets have been stepwise reduced. A final resulting encoding was found after five days of working alone and half a day of working in a team of three.

Compared to the code improvements the selection of an optimal solving approach is significantly faster. The performance of the solving approaches is evaluated by solving different test scenarios. The best solving approach for the incubed IT assignment problems is found within one day of working time.

Effort of Integration in the Existing System

At incubed IT the assignment problem is solved within the FMS in one specific function. To integrate the ASP program in the existing system the provided input data is modified and a micro web framework is set up in Java. A Python script with a flask application and the Clingo-API is provided. The whole integration can be implemented in one working day. Even though the ASP program requires more integration effort than a Java function this effort of one day is acceptable. The integration is done only once and does not effect the effort of modifying and adding new rules and constraints during development.

7.2. Evaluation of Runtime and Quality of Results

There are high requirements on the runtime of systems that are used in a real-time environment. An important argument for using ASP instead of an imperative method for the solving of task assignment problems is the possible reduction of the overall runtime. At the same time the quality of the results has to be the same or better than the one of the existing FMS. In the following test scenarios close to real use cases are defined for incubed IT and BMW Group. The required time for the solving process and resulting answer set are compared to the existing systems.

7.2.1. Evaluation at BMW

Previous to the evaluation of the runtime and quality of the new ASP implementation test scenarios were defined. These scenarios represent realistic environments of the BMW FMS. For the evaluation of the task assignment algorithm test scenarios with 5, 20 and 50 open orders are introduced. Each number of orders is used in 3 test scenarios, where a different number of available and currently not delivering robots are provided. All robots have a battery level above 25%. The time of generation of the orders is set randomly in a time range of one month. In every test scenario one map and one fleet are used and the robots are freely movable in the whole environment. The positions of the robots and get stations of the tasks are randomly set on a 1000m x 1000m area. In table 7.1 all nine test scenarios used for the task assignment are listed. For the park and charge assignment algorithm

Test Scenario	**Assignable Orders**	**Assignable Robots**
1		2
2	5	3
3		4
4		6
5	20	12
6		18
7		15
8	50	30
9		45

Table 7.1.: Evaluation scenarios for the BMW task assignment

similar scaled test scenarios are defined. 5, 20 and 50 stations and different numbers of robots are provided. All robots are available for the assignment and currently not in delivery. Each of them has a battery level between 40% and 80% percent. By that every robot can be assigned to charging and parking stations. The given stations are split into 1/3 charging stations and 2/3 parking places. In the scenarios only one fleet and one map are used. The stations and robots are placed randomly on a 1000m x 1000m area, and the robots are freely movable in the whole environment. The nine test scenarios for the park and charge assignment with the according number of stations and assignable robots are shown in table 7.2. To avoid misleading measurement results due to random test assemblies that lead to a better or worse runtime, every test scenario of the task assignment and the park and charge assignment is executed 10 times, where every time different random data is used. If the assignment problem of a test run requires more than 60 seconds of solving time, the solving of this test run is interrupted[1].

Test Scenario	Assignable Charging Stations	Assignable Parking Places	Assignable Robots
1	2	3	2
2			3
3			4
4	7	14	6
5			12
6			18
7	17	33	15
8			30
9			45

Table 7.2.: Evaluation scenarios for the BMW park and charge assignment

In chapter 6.3.3 the solving approach with the best performance, namely the splitting-based multithreading with four threads in combination with the branch-and-bound-based optimization strategy, was evaluated and is used in the test scenarios for both the task assignment and park and charge assignment.
At BMW the ASP grounder and solver are called in a built-in-module in Python. This Python script is started from the C# implementation of the FMS as a Microsoft Azure Application function. It showed up that the use of Microsoft Azure functionalities for test scenarios are not allowed, as the functionalities immediately affect the real intralogistics process at the BMW plant Regensburg. For example it is not possible to generate random data for the test scenarios and store it in the Azure database, as this test data would be used in the real intralogistics process. Therefore, a temporal test environment is set up. The input data is not provided by a database-request

[1]The time limit of 60 seconds was chosen in consultation with requirement engineers at BMW

but as a JSON file that is loaded into C#. In C# no Azure Application function is called to execute the ASP program. Instead the Python function with the build-in-module for Clingo is called as a process resource in C#. To analyse the runtime effects of calling the ASP program as a standalone application and within the Python process resource in C#, both runtimes are tracked and compared to the performance of the original implementation.

Performance and Quality of the BMW Task Assignment Algorithm

In this section the runtime and the quality of the result of different scenarios for the task assignment algorithm are compared. The test scenarios that have been introduced at the beginning of this chapter and are shown in table 7.1 are used for the evaluation.

Performance Evaluation To analyze the performance of the implementations of the BMW task assignment algorithm it is focused on two topics. First the runtime for different implementations of the algorithm are evaluated. Thereby test scenario 5 is considered, as this scenario is in between the low scaled and high scaled scenarios. Second the performance for the different scaled scenarios is compared.
In figure 7.1 the performance of the task assignment in different implementations for test scenario 5 is shown. The detailed measurements are listed in table B.5 in the appendix. Next to the performance of the original implementation and the implementation of ASP integrated into C# the runtimes of ASP integrated in Python and the standalone ASP grounding and solving are given. The runtime at test scenario 5 show a significant better performance of the current implementation compared to the ASP-based implementations. The box, representing the runtime distributions, is significantly smaller than the boxes of the ASP implementations. While the runtime of the C# implementation is mostly below 1 millisecond, the same scenario requires by the combined FMS of imperative and declarative methods up to 15 seconds (see table B.5), but for that another more complex optimization strategy was used.
Comparing the required runtimes by starting the ASP program as standalone application, from within C# and from within Python it is seen that

the execution in C# takes the longest. The upper whisker has the highest value, and also the box in the plot shows the highest distribution of runtimes. The fastest solution is found by calling the ASP from within Python. There are two possible reasons why the ASP called from Python is faster than as standalone application. In Python only the last, optimal solution is printed out. Calling ASP as a standalone application in an anaconda prompt every possible solution is printed on the terminal window. This behaviour leads to a runtime increase. Furthermore, by calling Clingo from within Python it is started in multi-shot mode what can lead to different solving times. The higher runtime for the ASP implementation in C# than the implementation in Python shows that the Python call from within C# is not solved optimally.

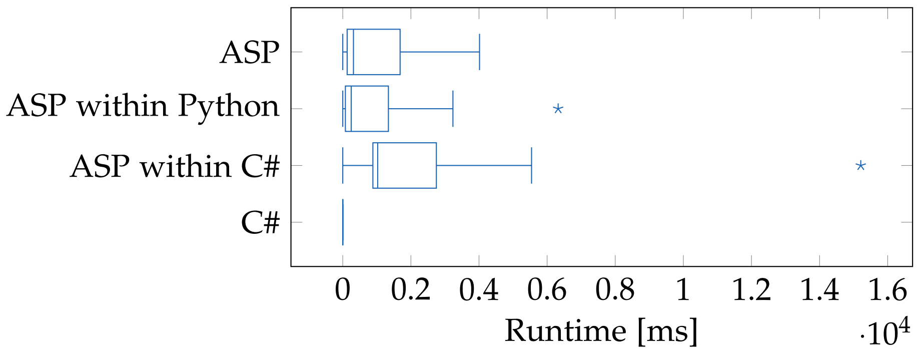


Figure 7.1.: Runtime of the different BMW task assignment implementations at test scenario 5

In table 7.3 the mean value and the standard deviation of the runtimes for all test scenarios is shown and the number of solved test runs is given. The detailed results of all test runs for the different test scenarios are provided in table B.1 to table B.9 in the appendix. If the optimal solution is not found within the BMW-specific time limit of 60 seconds, the solving process is aborted. These aborted test runs are not considered in the calculations for the mean and standard deviation.

The mean performance of the imperative method is for every scenario the best. The standard deviation shows the best results for the imperative method, the worst are given for the ASP implementation in C#. Analysing the number of solved test runs it can be seen that all implementations solve every test run of scenario 1 to 5. For higher scaled problems the ASP

Test Scenario	**C# Implementation**			**ASP within C# Implementation**			**Standalone ASP**		
	μ [ms]	σ [ms]	# TRS	μ [ms]	σ [ms]	# TRS	μ [ms]	σ [ms]	# TRS
1	0	0	10	427.2	40.3	10	10.9	10.46	10
2	0	0	10	415.5	18.16	10	8.3	8.92	10
3	0	0	10	471.8	83.61	10	9.6	8.26	10
4	0	0	10	523.2	65.38	10	17.4	4.79	10
5	0.3	0.48	10	2802.2	4445.21	10	1428.9	2746.91	10
6	1.2	0.42	10	23637.4	18196.13	5	8691.2	7222.24	5
7	1.3	0.67	10	5224.71	5983	7	3444.29	5083.52	7
8	0	0	10	/	/	0	/	/	0
9	0.4	0.7	10	/	/	0	/	/	0

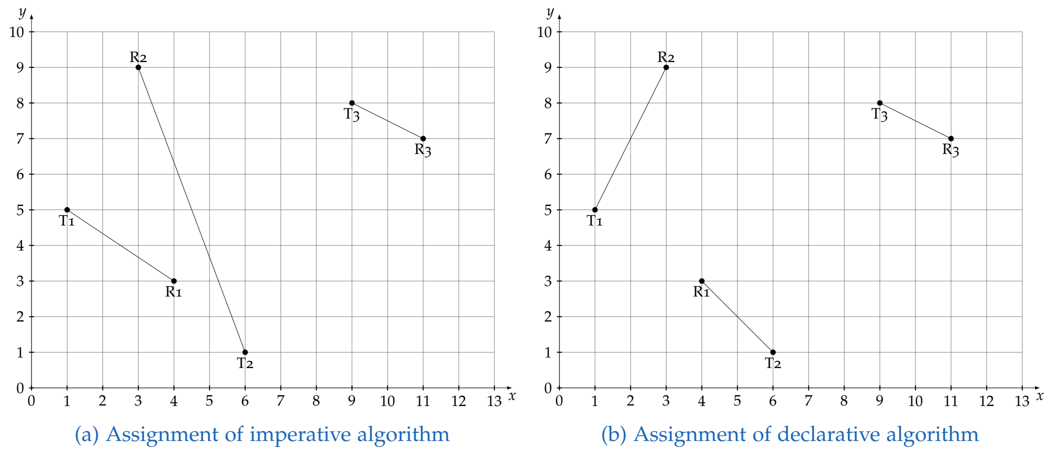
Table 7.3.: Runtime and solved test runs (TRS) for the different BMW task assignment implementations

implementations show its limits. While for test scenario 6 and 7 some test runs can be solved, the test runs of scenario 8 and 9 can not be solved with ASP but only with the imperative approach.

Quality Evaluation For the evaluation of the ASP implementation not only the runtime, but also the quality of the provided solutions is of interest. The ASP solver did not find an optimal solution for all test scenarios within the time limit, but valid answer sets are provided for every test.
As already mentioned in chapter 6.2.2, the optimal assignment of robots to open orders differ in the used algorithms in C# and ASP. In the original implementation a list of all open task and a list of all assignable robots are given. The task with the earliest time of creation is assigned to the robot with the shortest Euclidean distance. The task with the second-earliest time of creation selects then the closest of the remaining robots. This selection algorithm is continued until the list of open tasks or the list of assignable robots is empty. Summarizing this algorithm optimizes not the whole assignment but finds the optimal solution for the highest prioritised robots.
In the ASP implementation a different algorithm is used. Not the Euclidean distance for single problems is optimised, but the traveling costs of the whole fleet. The answer set for which the sum of the Euclidean distances of all assignments is minimal, is found to be the optimal solution.

A graphical representation in figure 7.2 shows the difference between both algorithm. 3 open tasks and 3 assignable robots are given, where every robot can be assigned to every task. The task with the earliest creation time is T1, T3 is the latest created task. The algorithm that is implemented in the current imperative FMS would in a first step select task T1 and assign it to the closest robot, namely R1. In the next step task T2 is selected and assigned to the closest of the remaining robots, the robot R2. Task T3 is assigned to robot R3. In the ASP encoding all possible answer sets are compared, and the one with the lowest overall traveling costs is set to be the optimal solution. In the provided example the optimal solution is found to be the set of assignments of task T1 to robot R2, task T2 to robot R1 and task T3 to robot R3. When comparing the sum of the Euclidean distances of both algorithms it can be seen that the result reduces from 7.1 measurement units in the original implementation to 4.9 measurement units in the implementation of ASP.

The original implementation is suitable for situations, in where orders are



(a) Assignment of imperative algorithm

(b) Assignment of declarative algorithm

Figure 7.2.: Graphical representation of the optimal assignment

generated in a broad time range. However, in the application area of the FMS new tasks are generated in ranges of seconds. Having time differences of only a few seconds the orders have to be delivered nearly simultaneously. Optimising the overall traveling distance ensures that all tasks are finished in time, even tough the earliest created task is probably delivered not as fast as by the use of the algorithm of the original implementation.

In test scenario 8 an optimal solution was not found for any of the ten test runs within the 60 seconds time limit. However, the ASP solver returned possible answer sets continuously during the solving process. In the following the quality of the results is evaluated for test scenario 8. The best answer set that is found by the ASP solver after 1 second, 5 seconds and 60 seconds runtime is compared to the result of the imperative method.

The traveling costs for the found assignments are shown in figure 7.3. The traveling costs of the ASP-based implementations are significantly lower than the one of the C# implementation. For test run 6 the costs decreased by 66%. It can further be seen that the performance improvements of the ASP solver for different time limits are barely identifiable. As the ASP solver supports an anytime algorithm, the solver returns possible answer sets after less than a second of solving time. The answer sets that are returned after

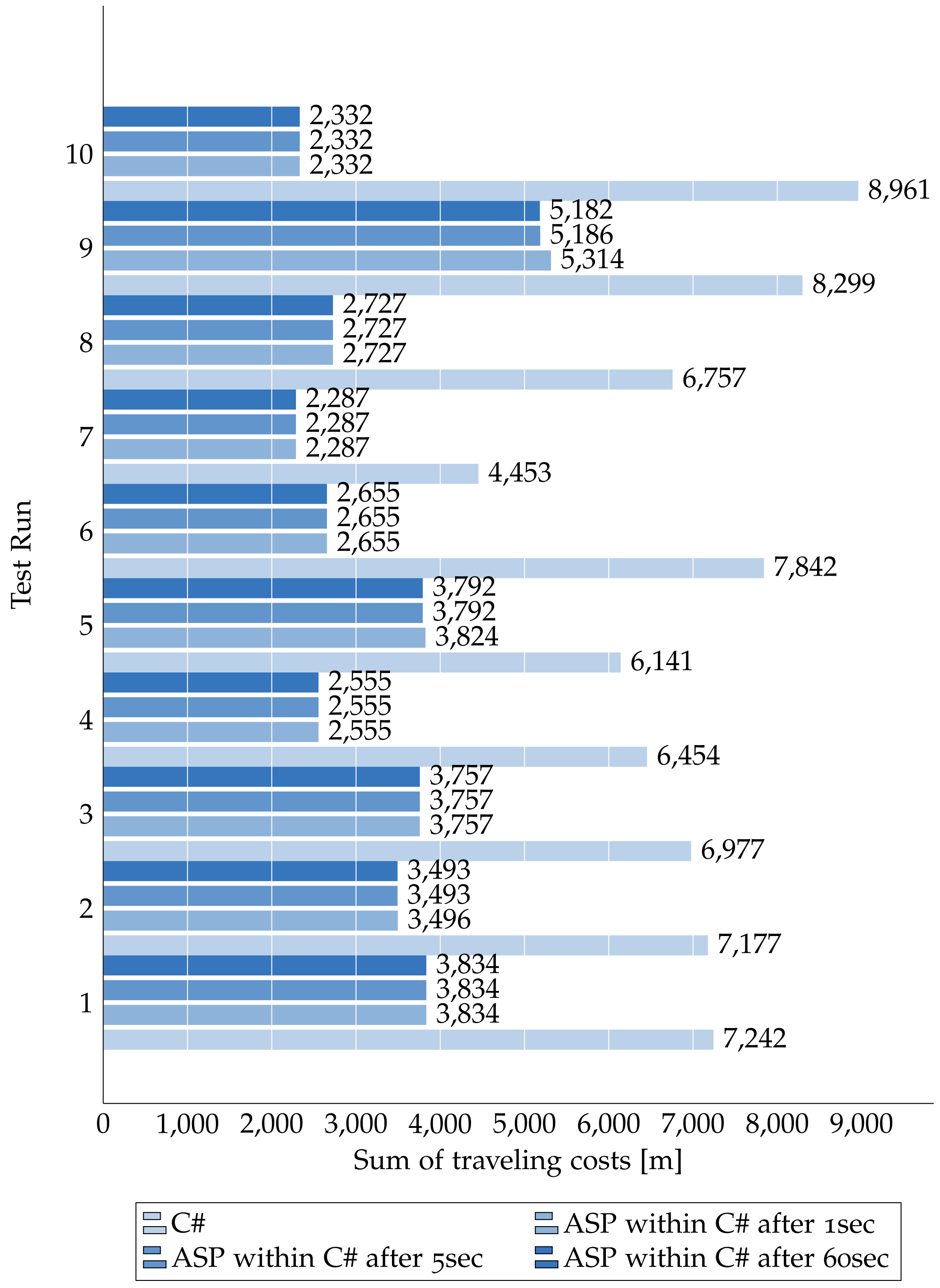


Figure 7.3.: Traveling costs for the BMW task assignment at test scenario 8

a few milliseconds are solutions with low travelling costs, but are not the optimal assignment. However after 60 seconds of solving time the costs are reduced only by a few centimetres. Using ASP with a solving time limit of 1 second will provide very good answer sets in a short runtime for the FMS.

Performance and Quality of the BMW Park and Charge Assignment Algorithm

Like for the evaluation of the task assignment algorithm in the following the runtime and the quality of the results for the new implementation of the park and charge assignment algorithm are analysed. The test scenarios that have been introduced at the beginning of this chapter and are shown in table 7.2 are used for the evaluation.

Performance Evaluation In figure 7.4 exemplary the runtimes for the fifth test scenario are shown to evaluate the performance of the different implementations. The underlying data of the boxplot is given in table B.14 in the appendix.

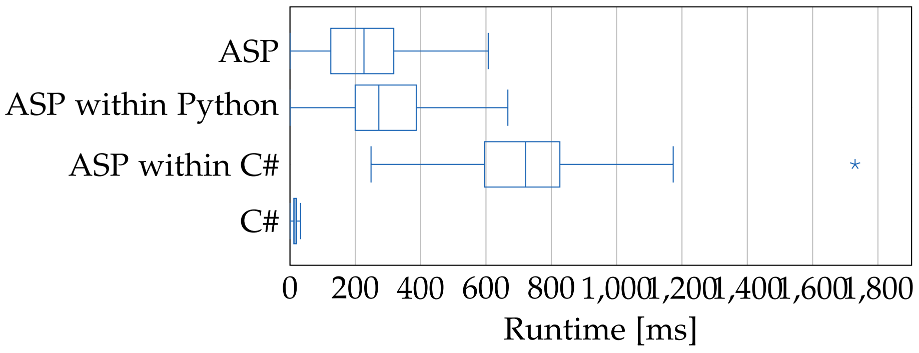


Figure 7.4.: Runtime of the different BMW park and charge assignment implementations at test scenario 5

The performance of the original implementation is better than the performance of the ASP implementations. This can be seen on the smaller distribution of runtimes, indicated by the boxes, and the upper whisker that is significantly lower for the C# implementation. While for the task assignment the runtime of the ASP call in Python was the fastest of all ASP implementations, in the park and charge assignment the fastest solutions are found for the standalone ASP. The runtime of the ASP program in C# shows as before in the task assignment the worst performance and indicates an required improve of the ASP call in C#.

In table 7.4 the mean value and the standard deviation of the runtime of every test scenario is shown. The results of the different test runs for every scenario are provided in the tables B.10 to B.18 in the appendix. If the solving of a test scenario reaches the BMW-specific time limit of 60 seconds the solving process is interrupted. These test runs are not considered in the calculation of the mean and standard deviation. The C# implementation shows for all scenarios a better performance than the ASP implementations, but for that in the ASP encoding the more complex optimization strategy is used that is harder to be solved. The test scenarios 1 to 6 can be solved by all implementations within the time limit. In test scenario 7 the original implementation in C# can solve all problems in a mean runtime around a quarter of a second, in the ASP implementations only 9 test runs can be solved. For test scenario 8 and 9 it is seen that ASP reaches its performance limits and no test run was solved optimally within the time limit of 60 seconds. Further can be observed that the ASP implementations show a lower mean runtime for test scenario 7 than for test scenario 6. It indicates that the encoding is better fitting problems in where more stations than robots are given. For all test scenarios and implementations it can be seen that the standard deviation increases with higher mean runtimes.

Quality Evaluation The algorithm that is used to assign the robots optimally are different in C# and in the ASP encoding. In C# the robot with the lowest battery level is assigned to the nearest charging station, the robot with the second-lowest battery level is assigned to the closest of the remaining charging stations. In the ASP encoding the overall traveling distance is optimised, what is a more complex optimization strategy and is harder to be solved.
The original algorithm is suitable for situations, in where one robot has a

Test Scenario	C# Implementation			ASP within C# Implementation			Standalone ASP		
	μ [ms]	σ [ms]	# TRS	μ [ms]	σ [ms]	# TRS	μ [ms]	σ [ms]	# TRS
1	0	0	10	393.7	13.78	10	13.9	8.21	10
2	0	0	10	473.8	81.24	10	13.9	8.88	10
3	0	0	10	451.1	57.43	10	12.8	6.75	10
4	1.7	1.34	10	578.3	158.7	10	33.4	6.31	10
5	16.2	4.87	10	788.1	350.75	10	341.7	404.17	10
6	62.8	5.67	10	21967	16538.62	10	14660.8	14248.6	10
7	251.1	25.08	10	10677	15169.36	9	7426.44	7932.5	9
8	1753.3	127.58	10	/	/	0	/	/	0
9	5541.2	465.79	10	/	/	0	/	/	0

Table 7.4.: Runtime and solved test runs (TRS) for the different BMW park and charge assignment implementations

very low critical battery level and the other robots have an acceptable battery level. As the critical robot has to go charging urgently it should be assigned to the closest appropriate charging station. However, this implementation shows its limits on circumstances where multiple robots have critical battery levels that differ only in a very small amount. An assignment where the overall traveling costs are reduced is more suitable, as all assigned robots have an appropriate traveling distance and not only the one with the lowest battery level.
It has been shown in table 7.4 that the optimal assignment sets were not found for the test scenarios 8 and 9 within the time limit. In the task assignment algorithm the quality of the results for scenario 8 showed better results for the ASP-based implementation even tough the optimal solution was not found.
In figure 7.5 the overall traveling distances for the test runs of scenario 8 of the park and charge assignment are displayed. The costs of the imperative method are compared to the declarative solutions that are found within 1 second, 5 seconds and 60 seconds solving time. It can be seen that most of the test runs, in contrast to the test runs of test scenario 8 of the task assignment, find no possible answer set even after the time limit of 60 seconds is reached. Test runs 8 and 10 return solutions after 1 second, but the traveling costs are worse than the one of the original implementation. This observation leads to the assumption that the encoding of the park and charge assignment problem in ASP is not optimal, as the performance of the task assignment encoding for similarly scalled problem is significantly better. As the ASP solver does not find at least one possible answer set for most of the test runs of high-scaled test scenarios, the current implementation of the declarative approach is not suitable for the solving of the park and charge assignment problem at BMW.

7.2.2. Evaluation at incubed IT

To evaluate the performance and correctness of the new generated ASP, two test setups with different scenarios are defined and solved. The results are compared to the current Java implementation.
One test setup is used to test different problem scales, whereas the second

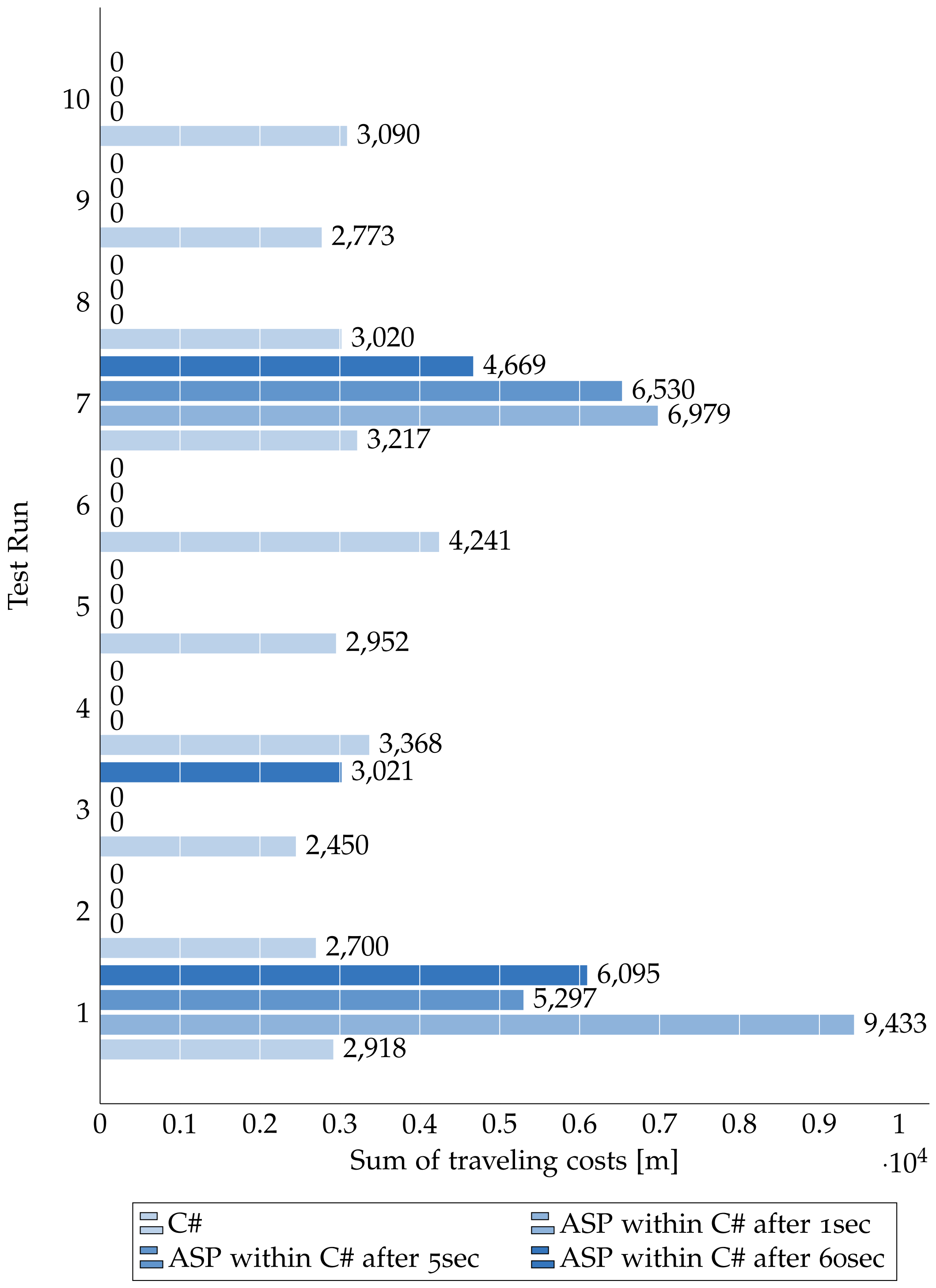


Figure 7.5.: Traveling costs for the BMW park and charge assignment at test scenario 8

setup evaluates the impact of an increasing number of constraints that have to be considered for the assignment problems.
Both test setups are used on the currently most complex intralogistics environment where the incubed IT FMS is integrated. This environment has a floor area of 100m x 86m where the robots are freely movable. On the map 36 get stations, 109 put stations, 19 charging stations and 44 parking places are located. Every hour up to 800 new tasks have to be delivered by 30 robots.
For both test setups the charging limits of the robots are set to the following:

- fully charged limit: 90 %
- idle charge limit: 70 %
- active charge limit: 50 %
- busy charge limit: 35 %
- critical charge limit: 30 %

The performance of the assignment algorithm influences the whole intralogistics environment. If the solving of a given problem requires more than 30 seconds the solving process will be stopped due to the reached time limit. The time limit was set to 30 seconds in consultation with the requirement engineers at incubed IT.

Evaluation of Test Setup 1

In the first test setup different problem scales are evaluated. At the FMS start-up 30 robots have to be assigned to orders, charging stations and parking places. After the start-up process most of the robots will be in a delivering process. Only a small number of robots is expected to be available for new assignments. Therefore not only test scenarios with 30 robots are set up, but also scenarios with 10 and 5 robots.
In table 7.5 the different test scenarios are shown. For each of the problem scales of 5, 10 and 30 assignable robots different test scenarios are set up. To evaluate the performance of the charging and parking assignment, in one scenario the number of open orders is half of the number of available robots. By that all unassigned robots are assigned to charging stations and parking places. In another scenario the number of robots equals the number

of orders. Third, the impact of scenarios with twice as much open orders than available robots is analysed.

Test Scenario	Assignable Robots	Assignable Charging Stations	Assignable Parking Places	Assignable Orders
1				2
2	5	3	7	5
3				10
4				5
5	10	6	14	10
6				20
7				15
8	30	18	42	30
9				60

Table 7.5.: Test scenarios for the first test setup at incubed IT

All robots, stations and orders are in the same pool, and the battery level of the robots is set randomly in a range of 40% to 99%. To evaluate the runtime and quality of the new ASP implementation, every test scenario is tested with 10 different test runs. The get stations, charging stations and parking places used in a test run are selected randomly from the provided stations on the original incubed IT map. The positions of robots are set randomly.

Performance Evaluation In table 7.6 the mean value and standard deviation of the runtimes for the test scenarios solved with the original code and with the in Java integrated ASP are shown. The results of the different test runs for every scenario are provided in the tables C.1 to C.9 in the appendix. To analyze the runtime influences that are affected by calling the ASP from within Java and the use of a micro web framework and the built-in-module Clingo for Python, the runtime for the standalone ASP solving is listed as well. A timeout is reached when a test run requires more than 30 seconds to find an optimal solution. Test runs that reached the timeout are not considered in the calculation for the mean and the standard deviation. It

Test Scenario	**Java Implementation**			**ASP within Java Implementation**			**Standalone ASP**		
	μ [ms]	σ [ms]	# TRS	μ [ms]	σ [ms]	# TRS	μ [ms]	σ [ms]	# TRS
1	327.6	173.42	10	283.3	186.29	10	100.9	47.81	10
2	278.3	226.64	10	121.3	143.5	10	71.9	14.96	10
3	275.6	121.5	10	132.9	76.3	10	71.4	7.93	10
4	363.1	358.06	10	254.6	220.59	10	90.44	8.65	10
5	491.3	223.23	10	495	339.16	10	278.67	236.67	10
6	656.1	337.56	10	244.2	167.39	10	120.1	35.76	10
7	8411.3	16712.51	10	13052.67	7099.53	3	29995	9485.25	1
8	2572.8	4894.22	10	/	/	0	/	/	0
9	1497	834.1	10	/	/	0	/	/	0

Table 7.6.: Runtime and solved test runs (TRS) for the different incubed IT assignment implementations in test setup 1

can be seen that for smaller test instances the mean system performance can be improved using declarative methods. Especially for scenarios where more open orders than robots are provided the ASP shows its benefits. At the test scenario 6 the required mean runtime is reduced by 63% using the ASP implementation in Java. However, the table shows the limits of ASP as well. While for the test scenarios 1 to 6 the ASP implementations can solve all test runs within the time limit, for the test scenario 7 to 9 with 30 robots the optimal solution can not be found for every test run. In scenario 7 the ASP implementation in Java can solve 3 test runs, but for test scenario 8 and 9 no optimal solution was found within the time limit. Looking at the standard deviation for scenario 1 to 6 the standalone ASP application returns the smallest values. The Java implementation has a slightly higher standard deviation than the ASP program implemented in Java. In general it can observed that with a higher mean runtime the standard deviation increases.
Comparing the mean runtime required by calling the ASP from within Java and as a standalone application, it shows that the mean runtime of the standalone application is smaller. The reason for the differences is the additional time that is required to save the provided input data as new classes in Java and the passing of the JSON data via the web framework to the Python script.

Quality Evaluation Not only the runtime is a quality criterion for a new system, but also the quality of the provided solutions.
Comparing the optimal answer set of the Java implementation and the ASP implementation for testing setup 1 to 6 both return the same solution, as they have the same underlying assignment strategy and the ASP solver finds the optimal solution within the time limit. To analyze the quality of the results for test scenarios where the ASP solver does not find an optimal solution within the time limit the traveling costs of scenario 7 are compared to the implementation in Java. The quality of the result is evaluated by using the best-found answer set after 1, 5 and 15 seconds. The time range is set to the maximum value of 15 seconds as this time value is similar to the maximal mean runtime required by the Java implementation for test scenario 9. In figure 7.6 a bar chart is shown with the traveling costs of test scenario 7. The travelling costs are shown in two groups, as at incubed IT

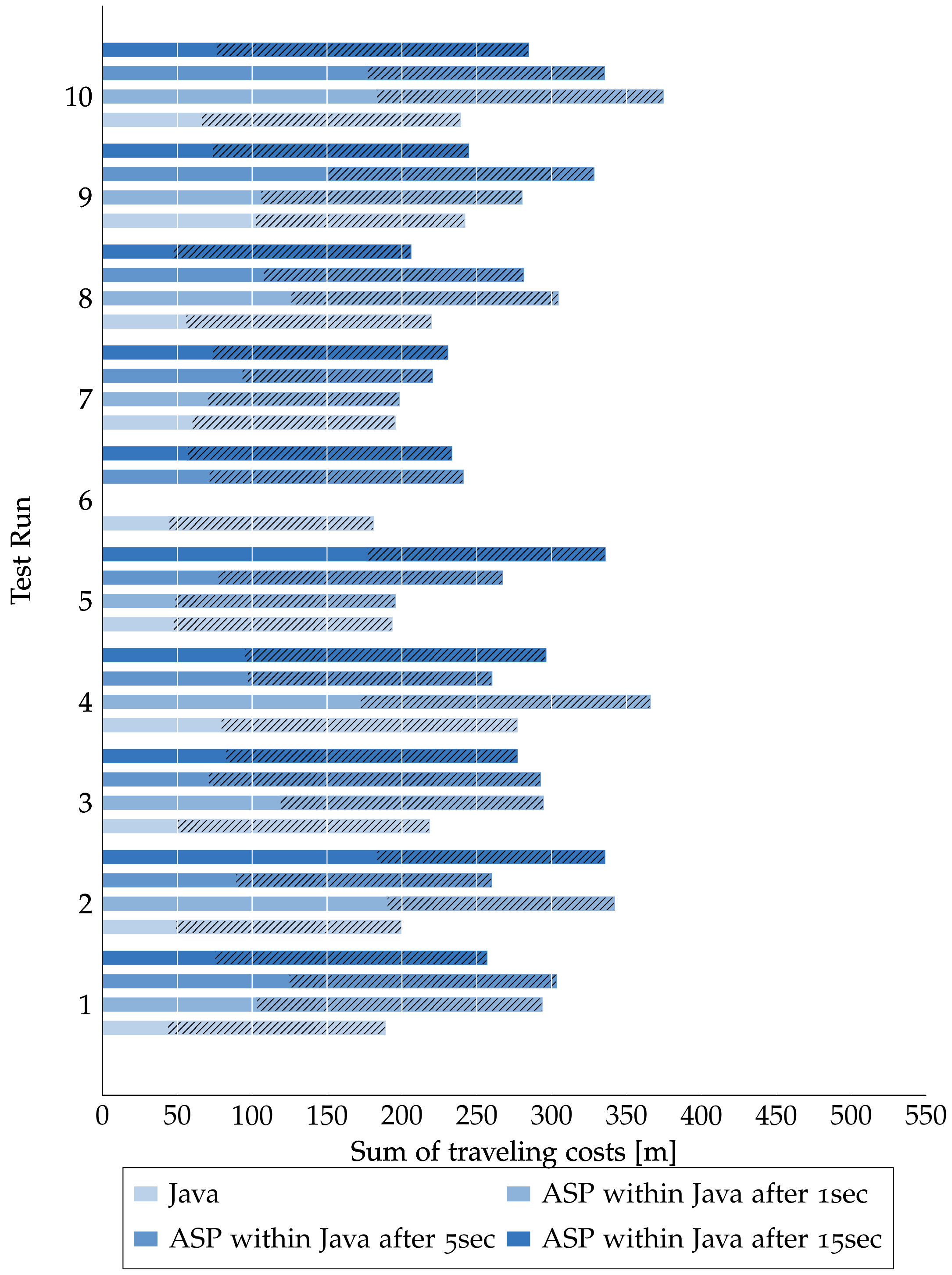


Figure 7.6.: Traveling costs for the incubed IT assignment at test scenario 7

the optimization strategy consists of two differently weight criteria. Most important is the reduction of travelling costs of robots that are assigned to tasks or charging stations due to forced timeslot charging, critical charging or busy charging. These traveling costs are represented as unicolored bar charts. Less important and by that lower weighted are the traveling costs of robots that are assigned to parking places and charging stations for idle charging. The costs of this second optimization criterion are visualized in figure 7.6 by the dashed bars.
For all test runs, except for test run 6, a first solution is returned by ASP anytime algorithm within 1 second. By that possible assignment sets can be provided to the FMS, although they are not the optimal one. It can be seen that the overall traveling costs can increase over time, for example in test run 5 and test run 7. At the beginning of these two test runs the ASP solver finds possible solutions with low traveling costs, but the traveling costs for the more important criterion are not optimal. Looking at the traveling costs for the more important criterion it can be seen that the ASP-based solving process finds better solutions over time for most of the test runs. Exceptions are the test runs 2 and 5. When the solving process stopped after 15 seconds the costs for the more important criterion, represented by the unicolored bars, are higher than after 5 seconds of solving time. The reason why the ASP solver considers the solution after 15 seconds to be better than the one after 5 seconds is probably based on the internal behaviour of the multithreading solving approach and influences of the Vsids heuristic.
Looking at the overall results for test scenario 7 it can be seen that the original implementation in Java provides a significantly faster and better solution than the ASP implementations.

Evaluation of Test Setup 2

In the second test setup the impact of different constraints is analysed. The test scenarios that are set up differ significantly from the one of the first test setup. In here different pools are set, robots in delivery and in charging stations are considered for the task assignment, and orders with predecessors have to be assigned.
20 orders, 10 robots, 6 charging stations and 14 parking places are given. 3 robots and 6 orders are assigned to pool A, 3 robots and 6 orders to pool B

and the remaining 4 robots and 8 orders are not assigned to a pool. The test scenarios that are analyzed in this test setup are the following:

- 6 robots are currently in delivery.
- 2 robots are in delivery and 2 robots are currently charging with battery levels above the active charge limit.
- 2 robots are in delivery and 2 robots are currently charging with battery levels above the active charge limit. 2 orders have predecessors, where none of the predecessors is currently finished.

.

Like in the first test setup 10 different test runs are set up for every test scenario. In there the pickup stations, charging stations and parking places are selected randomly from the provided stations on the original incubed IT map. The priorities for orders, charging stations and parking places are set randomly in a range of 1-99. The battery levels of the robots are set randomly in a range of 40% to 99%. The robots are placed randomly in the map and are freely movable in the whole area.

Performance Evaluation The mean and standard deviation of the runtime for every test scenario is shown in table 7.7. Detailed results of the different test runs for every scenario are provided in the tables C.10 to C.12 in the appendix.

Test Scenario	**Java Implementation**		**ASP within Java Implementation**		**Standalone ASP**	
	μ [ms]	σ [ms]	μ [ms]	σ [ms]	μ [ms]	σ [ms]
1	314.4	115.9	121.6	58.73	60.7	8.76
2	371	172.34	142.3	83.69	117.2	32.34
3	338.1	154.61	158.3	92.68	96.6	14.25

Table 7.7.: Runtime for the different incubed IT assignment implementations in test setup 2

Like for the run 6 in test setup 1, in where the same number of robots and orders is given, the runtime for the solving with declarative methods is faster than with imperative methods. Comparing the mean runtime of

test scenario 6 in the first test setup (see table 7.6) with the results of test scenario 1 in this second test setup following can be observed. In the scenario in test setup 2 the number of pools increased from 1 (no pool) to 3 different pools (no pool, pool A, pool B). Furthermore, 6 out of the 10 robots are currently in delivery. The mean runtime of the Java implementation to solve the problem in test setup 2 decreased by 52% compared to the runtime at test setup 1, the standard deviation decreased by 66% . Using the ASP encoding integrated into Java, the required mean runtime decreased by 50%, the standard deviation decreased by 65%. This observation gives the assumption that these constraints affect the imperative and declarative method the same way. No method shows a better or worse performance for problems with more constraints.
Looking at the results for the second scenario an increased mean runtime and standard deviation of the ASP program integrated into Java is observed. The reason for that is the decreased number of delivering robots. With the assignment of tasks to currently delivering robots the number of possible answer sets in test scenario 1 is lower, as some combinations of tasks and robots do not have to be considered.
In test scenario 3 the second setup is extended by two orders that have predecessors that are not yet finished. It is expected that the runtime decreases, as in the program for input data modifications all tasks with unfinished predecessors are sorted out and not considered for the later assignment algorithm. However, the mean runtime in table 7.7 for the ASP within Java shows an increase of the runtime. The performance is still better than the one of the original Java implementation.
Looking at the runtime of the ASP standalone application for test scenario 3 a runtime improvement can be observed. The time difference to test scenario 2 is only 7.5 milliseconds, but it still shows a small performance increase. The cause for the increased runtime for the ASP within Java is thereby not the ASP itself but is based on the data modifications in Java and Python.

Quality Evaluation To evaluate the code quality the optimal assignment set of the Java implementation is compared to the one provided by the ASP integrated into Java. As both systems have the same underlying assignment strategy and the ASP-based implementation finds the optimal solution within the time limit both systems return the same results.

8. Conclusion

In this work the integration of the declarative language ASP into existing fleet management systems to replace existing task assignment problem solving methods is investigated.
After an overview of the task assignment problem with a focus on multi-robot task allocation and a introduction to Answer Set Programming was given, the language syntax of this declarative method has been stated. Related research is discussed where ASP is used to solve assignment problems. Based on the implementation suggestions of [20] ASP-based systems are implemented in the FMS of BMW and incubed IT. After the identification of needs it was decided to replace the algorithms, that assign intralogistics robots to orders, charging stations and parking places, by a declarative method. After defining the requirements and constraints of the assignment problems they are encoded in ASP. To reduce the runtime of the implementation, different performance improvements have been applied. The use of different optimization criteria can improve and downgrade the performance of the system significantly. As in this work the ASP system Clingo is used it was possible to improve the performance further by the selection of different solving approaches that are provided by the solver Clasp.
One main quality criterion of the FMS is the performance and the quality of the results. To evaluate the criterion, test scenarios have been set up that are based on typical use cases of the FMSs. The runtime and the quality of the provided solutions for the test scenarios are compared. Therefore, the imperative and the declarative assignment algorithms are used to solve the problems.

At the BMW Group the evaluation of the performance and quality of the provided solutions of the ASP-based and imperative implementation gave following results. The runtime of the ASP-based solving is, other than expected, worse than the runtime of the imperative method, but for that

in ASP a optimization strategy was implemented that is a harder problem to solve. For small problem instances the difference in the runtime is especially notable for the task assignment, but the park and charge assignment algorithm shows decreased performance as well. Very critical is the fact that for higher-scaled problems not even one possible solution for the park and charge assignment within a time limit is found.
The quality of the result, however, showed up to be better with the ASP implementation. Looking at the overall traveling costs of the answer sets that are provided by the ASP solver and the imperative implementation an improvement was possible. For problem instances where the ASP-based implementation did not find the optimal task assignment in a specified time limit some promising results showed up after a second of solving time. For the current FMS implementation at BMW Group Answer Set Programming can be a helpful tool for rapid prototyping and the testing of new assignment strategies. The required time to set up a new implementation of the assignment problem can be decreased significantly using the ASP. Under consideration of high-performance requirements for the solution finding the use of imperative methods is currently preferred.

Compared to BMW, the assignment strategy at incubed IT is more complex. A higher number of constraints and rules have to be considered, for example the assignment rules for chained orders and the different charging states. Furthermore, the optimization criterion is more complex as the traveling costs of the whole fleet is optimized. The increased complexity shows the limits of the imperative system and benefits of the ASP-based implementation. The performance of the FMS for relatively small problem instances up to 10 assignable vehicles and 20 open orders is better for the ASP-based implementation.
On the other hand, at high scaled assignment problems, like one with 30 assignable vehicles and 60 open orders, the ASP system reaches its performance limits. Whereas the imperative implementation can find an optimal solution in less than 15 seconds, the ASP solver does not find one possible answer set in twice the length of computational time. Such high-scaled problem instances appear very infrequently at incubed IT. A situation where high-scaled problem instances are considerable is the start-up of the intralogistics area and the FMS. At the very beginning all vehicles are unassigned, but a set of open orders is already provided.

The significantly more common application field for the assignment algorithm is a constantly running FMS. Every couple of seconds the assignment solver is called. Therefore only small instances of assignable vehicles are provided. For such small problem instances the ASP system showed up to have a better performance than the current imperative program.
For problem instances where ASP finds the optimal solution within a specified time limit, the quality of results is the same for the imperative and declarative implementation. However, for problem instances where ASP does not find the optimal solution within a time limit the results of the imperative method occurred to be better.

9. Outlook

In resent years the declarative method Answer Set Programming became popular as a promising method to solve assignment problems. Particularly for problems related to the optimal assignment of robots in logistic domains the potential of ASP has been shown in some research papers.
BMW and incubed IT currently face the same problems regarding their intralogistics. Due an increasing number of autonomously robots the existing Fleet Management Systems show a decreasing performance and high complexity for implementing new rules.
In this thesis ASP programs were used to replace existing imperatively described assignment problems in the Fleet Management Systems at BMW and incubed IT. The ASP system Clingo was used to model and solve these problem instances. It became apparent that depending on the environment the ASP shows better performance for systems with many rules and constraints but is not capable of solving high-scaled assignment problems with a low number of limiting constraints and rules optimally. However due to the ASP anytime algorithm possible solutions for the assignment problem are provided after a few milliseconds.
High-scaled assignment problems occur very infrequently, like at the start-up of an intralogistics area and the FMS. It could be considered for future implementations to accept a start-up- and initialization process of the FMS with assignment solutions that are not optimal. After the system start up and the first non-optimal assignments it is expected to have always only a few new assignable instances in the constantly running FMS. These instances can be modelled and solved with less computational effort by the ASP system than by an imperative method.
The fields of application for ASP are quite limited. There is no IDE under active development. Whereas the language syntax of ASP is very clear and lean, the finding of errors is hard and very time-consuming. A provided IDE could make the implementation of new ASP programs easier. Another

improvement that would make the ASP programming more applicable is the providing of some more APIs that can be used with multiple grounders and solvers. By that a declarative program could be implemented directly in an imperative system without the consideration of some additional languages, like it was required at incubed IT to set up an additional Python function. However, looking at resent research work an increased relevance for the IDEs and APIs can be observed. In the next years new APIs and IDEs will highly likely be developed and lead to an increased field of applications for the declarative method ASP.

Appendix

Appendix A.

Evaluation of Solving Approaches in Clasp

A.1. Solving Approaches for the BMW Task Assignment

Test Run	Compete-Based, 2 Threads [ms]	Compete-Based, 4 Threads [ms]	Splitting-Based, 2 Threads [ms]	Splitting-Based, 4 Threads [ms]
1	3142	3741	2034	1197
2	828	687	625	490
3	1344	2421	922	560
4	437	125	109	67
5	6867	5304	14793	2083
6	250	62	94	46
7	125	47	47	34
8	381	266	656	88
9	23313	21890	17908	13885
10	234	219	156	141
μ	3692.1	3476.2	3734.4	1859.1
σ	7204.7	6724.2	6715.88	4276.44

Table A.1.: Runtime for multi-threading approaches, combined with branch-and-bound-based optimization, at the BMW task assignment

Test Run	**Default Configuration [ms]**	**Opt-Heuristic = 1 [ms]**	**Berkmin Heuristic [ms]**	**Vmtf Heuristic [ms]**	**Vsids Heuristic [ms]**	**Unit Heuristic [ms]**	**Arbitrary Static Ordering [ms]**	**Domain Heuristic [ms]**	**Opt-Strategy = bb,1 [ms]**
1	4929	39108	16079	14588	11588	Timeout	12017	38328	4750
2	629	2762	2244	1477	1003	Timeout	2844	3556	484
3	1402	10626	10876	12434	3365	Timeout	1281	15824	1406
4	118	1379	771	478	224	Timeout	734	1348	109
5	6265	56381	21470	20750	16799	Timeout	4375	Timeout	6313
6	178	1505	1487	428	353	Timeout	391	769	172
7	153	1245	369	382	328	Timeout	328	1855	141
8	555	3172	3458	2205	943	Timeout	828	1065	516
9	28727	Timeout	Timeout	Timeout	49522	Timeout	42364	Timeout	28331
10	172	1460	500	359	328	Timeout	2423	359	141
μ	4312.8	13070.89	6361.56	5900.11	8445.3	/	6758.5	7888	4236.3
σ	8854.07	20337.99	7856.07	7845.42	15525.23	/	12989.61	13317.16	8744.26

Table A.2.: Runtime for solving approaches at the BMW task assignment

A.2. Solving Approaches for the BMW Charge and Park Assignment

Test Run	Compete-Based, 2 Threads [ms]	Compete-Based, 4 Threads [ms]	Splitting-Based, 2 Threads [ms]	Splitting-Based, 4 Threads [ms]
1	782	1125	1094	668
2	813	703	766	439
3	2057	1859	1860	920
4	1484	562	719	544
5	1266	969	1188	569
6	719	422	469	356
7	547	531	531	349
8	7267	1812	11032	5657
9	22491	7144	12182	9877
10	1148	632	742	491
μ	3857.4	1575.9	3058.3	1987
σ	6841.88	2022.52	4531.35	3208.84

Table A.3.: Runtime for multi-threading approaches, combined with branch-and-bound-based optimization, at the BMW park and charge assignment

Test Run	Default Configuration [ms]	Opt-Heuristic = 1 [ms]	Berkmin Heuristic [ms]	Vmtf Heuristic [ms]	Vsids Heuristic [ms]	Unit Heuristic [ms]	Arbitrary Static Ordering [ms]	Domain Heuristic [ms]	Opt-Strategy = bb,1 [ms]
1	719	750	Timeout	875	766	Timeout	Timeout	2221	719
2	641	547	Timeout	1031	641	Timeout	Timeout	10829	599
3	1797	1688	Timeout	2438	1828	Timeout	Timeout	15226	1751
4	1000	938	Timeout	1719	906	Timeout	Timeout	8278	969
5	1056	1044	Timeout	2307	1047	Timeout	Timeout	6005	1047
6	484	391	Timeout	500	438	Timeout	Timeout	10990	422
7	344	359	Timeout	455	328	Timeout	Timeout	1266	344
8	5635	5391	Timeout	9939	6813	Timeout	Timeout	18853	5610
9	26814	Timeout	Timeout	Timeout	28556	Timeout	Timeout	54952	24963
10	1219	1117	Timeout	1734	1234	Timeout	Timeout	13027	1175
μ	3970.9	1358.33	/	2333.11	4255.7	/	/	14164.7	3759.9
σ	8172.95	1567.94	/	2943.35	8748.53	/	/	15345.91	7607.81

Table A.4.: Runtime for solving approaches at the BMW park and charge assignment

A.3. Solving Approaches for the incubed IT Assignment

Test Run	Compete-Based, 2 Threads [ms]	Compete-Based, 4 Threads [ms]	Splitting-Based, 2 Threads [ms]	Splitting-Based, 4 Threads [ms]
1	Timeout	18081	12080	12246
2	242	399	214	295
3	571	11111	654	413
4	Timeout	Timeout	Timeout	Timeout
5	4274	4439	4604	5028
6	18525	10191	3932	5197
7	739	319	1791	3916
8	9366	3651	1335	7783
9	1071	907	847	3297
10	18887	4915	Timeout	Timeout
μ	6709.38	6001.44	3182.13	4771.88
σ	7998.53	6003.67	3923.51	3912

Table A.5.: Runtime for multi-threading approaches, combined with Vsids heuristic, at the incubed IT assignment

Test Run	**Default Configuration [ms]**	**Opt-Heuristic = 1 [ms]**	**Berkmin Heuristic [ms]**	**Vmtf Heuristic [ms]**	**Vsids Heuristic [ms]**	**Unit Heuristic [ms]**	**Arbitrary Static Ordering [ms]**	**Domain Heuristic [ms]**	**Opt-Strategy = bb,1 [ms]**
1	20714	Timeout	Timeout	Timeout	22404	Timeout	Timeout	Timeout	29943
2	464	219	3263	1923	238	Timeout	Timeout	299	249
3	3510	912	4015	1025	6172	Timeout	Timeout	186	1770
4	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout
5	10313	2913	Timeout	14782	5292	Timeout	Timeout	Timeout	9683
6	9961	Timeout	Timeout	20453	7410	Timeout	29996	Timeout	15401
7	3427	266	Timeout	1673	2026	Timeout	Timeout	294	3238
8	4548	15246	Timeout	19300	5385	Timeout	Timeout	12925	17002
9	1045	1948	Timeout	3248	943	Timeout	Timeout	1188	1162
10	6370	Timeout	12138	Timeout	1174	Timeout	15381	5528	13823
μ	6705.78	3584	6472	8914.86	5671.56		22688.5	3403.33	10252.33
σ	6290.62	5806.91	4921.28	8860.93	6790.38		10334.37	5093.92	9853.39

Table A.6.: Runtime for solving approaches at the incubed IT assignment

Appendix B.

Evaluation of Performance at BMW

B.1. Evaluation Results for the BMW Task Assignment

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	0	423	16
2	0	504	0
3	0	404	0
4	0	389	19
5	0	403	0
6	0	438	16
7	0	479	30
8	0	451	12
9	0	396	0
10	0	385	16
μ	0	427.2	10.9
σ	0	40.3	10.46

Table B.1.: Runtime for the BMW task assignment implementations at test scenario 1

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	0	401	16
2	0	408	16
3	0	440	0
4	0	430	0
5	0	400	0
6	0	399	14
7	0	396	21
8	0	416	0
9	0	418	0
10	0	447	16
μ	0	415.5	8.3
σ	0	18.16	8.92

Table B.2.: Runtime for the BMW task assignment implementations at test scenario 2

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	0	491	0
2	0	470	16
3	0	446	16
4	0	456	16
5	0	374	16
6	0	655	0
7	0	571	16
8	0	420	0
9	0	424	16
10	0	411	0
μ	0	471.8	9.6
σ	0	83.61	8.26

Table B.3.: Runtime for the BMW task assignment implementations at test scenario 3

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	0	440	16
2	0	534	16
3	0	611	15
4	0	592	16
5	0	556	16
6	0	496	16
7	0	603	31
8	0	463	16
9	0	493	16
10	0	444	16
μ	0	523.2	17.4
σ	0	65.38	4.79

Table B.4.: Runtime for the BMW task assignment implementations at test scenario 4

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	ASP within Python Implementation [ms]	Standalone ASP [ms]
1	0	2559	1355	1406
2	0	1121	296	370
3	1	1641	421	448
4	0	925	123	156
5	0	3315	1335	2516
6	0	772	72	47
7	1	627	78	31
8	1	931	199	258
9	0	15209	6324	8892
10	0	922	74	165
μ	0.3	2802.2	1027.7	1428.9
σ	0.48	4445.21	1926.02	2746.91

Table B.5.: Runtime for the BMW task assignment implementations at test scenario 5

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	2	Timeout	Timeout
2	2	22451	6377
3	1	17977	9714
4	1	Timeout	Timeout
5	1	17561	3569
6	1	Timeout	Timeout
7	1	56831	22561
8	1	Timeout	Timeout
9	1	3367	1235
10	1	Timeout	Timeout
μ	1.2	23637.4	8691.2
σ	0.42	18196.13	7222.24

Table B.6.: Runtime for the BMW task assignment implementations at test scenario 6

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	2	3208	1618
2	1	Timeout	Timeout
3	2	2041	234
4	2	Timeout	Timeout
5	1	Timeout	Timeout
6	1	1356	375
7	1	8581	3240
8	2	19060	16583
9	1	1216	656
10	0	1111	1404
μ	1.3	5224.71	3444.29
σ	0.67	5983	5083.52

Table B.7.: Runtime for the BMW task assignment implementations at test scenario 7

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	0	Timeout	Timeout
2	0	Timeout	Timeout
3	0	Timeout	Timeout
4	0	Timeout	Timeout
5	0	Timeout	Timeout
6	0	Timeout	Timeout
7	0	Timeout	Timeout
8	0	Timeout	Timeout
9	0	Timeout	Timeout
10	0	Timeout	Timeout
μ	0	/	/
σ	0	/	/

Table B.8.: Runtime for the BMW task assignment implementations at test scenario 8

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	0	Timeout	Timeout
2	1	Timeout	Timeout
3	0	Timeout	Timeout
4	0	Timeout	Timeout
5	0	Timeout	Timeout
6	2	Timeout	Timeout
7	0	Timeout	Timeout
8	0	Timeout	Timeout
9	1	Timeout	Timeout
10	0	Timeout	Timeout
μ	0.4	/	/
σ	0.7	/	/

Table B.9.: Runtime for the BMW task assignment implementations at test scenario 9

B.2. Evaluation Results for the BMW Park and Charge Assignment

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	0	421	0
2	0	385	16
3	0	379	0
4	0	382	16
5	0	383	16
6	0	407	13
7	0	408	19
8	0	393	16
9	0	388	27
10	0	391	16
μ	0	393.7	13.9
σ	0	13.78	8.21

Table B.10.: Runtime for the BMW park and charge assignment implementations at test scenario 1

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	0	667	16
2	0	431	16
3	0	423	16
4	0	529	0
5	0	535	16
6	0	416	0
7	0	422	16
8	0	463	16
9	0	435	31
10	0	417	12
μ	0	473.8	13.9
σ	0	81.24	8.88

Table B.11.: Runtime for the BMW park and charge assignment implementations at test scenario 2

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	0	410	16
2	0	445	0
3	0	400	16
4	0	403	16
5	0	412	16
6	0	506	0
7	0	469	16
8	0	396	16
9	0	512	16
10	0	558	16
μ	0	451.1	12.8
σ	0	57.43	6.75

Table B.12.: Runtime for the BMW park and charge assignment implementations at test scenario 3

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	1	484	31
2	1	599	31
3	3	521	51
4	5	884	35
5	1	504	31
6	1	569	31
7	2	845	31
8	1	481	31
9	1	433	31
10	1	463	31
μ	1.7	578.3	33.4
σ	1.34	158.7	6.31

Table B.13.: Runtime for the BMW park and charge assignment implementations at test scenario 4

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	ASP within Python Implementation [ms]	Standalone ASP [ms]
1	20	629	195	170
2	10	778	375	284
3	14	825	421	125
4	12	496	127	101
5	14	535	294	248
6	25	615	535	125
7	20	1730	248	1459
8	16	830	331	281
9	11	762	250	419
10	20	681	201	205
μ	16.2	788.1	297.7	341.7
σ	4.87	350.75	121.29	404.17

Table B.14.: Runtime for the BMW park and charge assignment implementations at test scenario 5

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	74	18522	14408
2	54	42153	8872
3	63	20183	3267
4	58	1982	5846
5	63	14264	19006
6	62	50358	50961
7	63	27955	20214
8	69	34418	16408
9	58	3876	5548
10	64	5959	2078
μ	62.8	21967	14660.8
σ	5.67	16538.62	14348.6

Table B.15.: Runtime for the BMW park and charge assignment implementations at test scenario 6

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	292	2632	1281
2	264	3461	3805
3	254	5014	19804
4	242	10169	1437
5	232	4965	1140
6	279	2180	5342
7	256	4063	5515
8	213	Timeout	Timeout
9	260	12082	6014
10	219	51527	22500
μ	251.1	10677	7426.44
σ	25.08	15169.36	7932.5

Table B.16.: Runtime for the BMW park and charge assignment implementations at test scenario 7

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	1911	Timeout	Timeout
2	1676	Timeout	Timeout
3	1785	Timeout	Timeout
4	1662	Timeout	Timeout
5	1748	Timeout	Timeout
6	1556	Timeout	Timeout
7	1669	Timeout	Timeout
8	1729	Timeout	Timeout
9	1992	Timeout	Timeout
10	1805	Timeout	Timeout
μ	1753.3	/	/
σ	127.58	/	/

Table B.17.: Runtime for the BMW park and charge assignment implementations at test scenario 8

Test Run	C# Implementation [ms]	ASP within C# Implementation [ms]	Standalone ASP [ms]
1	5722	Timeout	Timeout
2	5766	Timeout	Timeout
3	5874	Timeout	Timeout
4	6242	Timeout	Timeout
5	6172	Timeout	Timeout
6	5154	Timeout	Timeout
7	5125	Timeout	Timeout
8	5155	Timeout	Timeout
9	5171	Timeout	Timeout
10	5277	Timeout	Timeout
μ	5541.2	/	/
σ	465.79	/	/

Table B.18.: Runtime for the BMW park and charge assignment implementations at test scenario 9

Appendix C.

Evaluation of Performance at incubed IT

C.1. Evaluation Results for the incubed IT Assignment at Test Setup 1

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	605	439	162
2	457	491	194
3	634	303	147
4	237	97	78
5	151	163	66
6	190	106	66
7	267	642	78
8	226	198	78
9	261	116	78
10	248	278	62
μ	327.6	283.3	100.9
σ	173.42	186.29	47.81

Table C.1.: Runtime for the incubed IT assignment implementations at test scenario 1

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	903	487	78
2	249	82	109
3	338	103	62
4	217	114	62
5	171	109	62
6	130	104	62
7	163	129	62
8	193	99	78
9	197	98	78
10	222	110	66
μ	278.3	143.5	71.9
σ	226.64	121.3	14.96

Table C.2.: Runtime for the incubed IT assignment implementations at test scenario 2

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	538	336	66
2	426	100	78
3	293	132	78
4	300	99	62
5	194	93	81
6	231	172	62
7	188	130	78
8	208	89	69
9	237	87	62
10	141	91	78
μ	275.6	132.9	71.4
σ	121.5	76.3	7.93

Table C.3.: Runtime for the incubed IT assignment implementations at test scenario 3

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	1374	845	374
2	313	94	78
3	301	251	94
4	194	95	94
5	228	83	84
6	262	185	78
7	302	241	94
8	219	266	104
9	184	203	94
10	254	283	94
μ	363.1	254.6	90.44
σ	358.06	220.59	8.65

Table C.4.: Runtime for the incubed IT assignment implementations at test scenario 4

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	896	390	260
2	877	360	250
3	518	203	125
4	490	878	709
5	422	360	99
6	312	329	109
7	409	1199	623
8	411	780	359
9	329	281	109
10	249	170	125
μ	491.3	495	278.67
σ	223.23	339.16	236.67

Table C.5.: Runtime for the incubed IT assignment implementations at test scenario 5

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	1138	399	110
2	1216	174	90
3	486	160	91
4	568	195	121
5	873	667	109
6	422	111	109
7	834	228	211
8	354	175	94
9	257	175	125
10	413	158	141
μ	656.1	244.2	120.1
σ	337.56	167.39	35.76

Table C.6.: Runtime for the incubed IT assignment implementations at test scenario 6

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	1079	5637	Timeout
2	913	Timeout	Timeout
3	678	Timeout	Timeout
4	934	Timeout	Timeout
5	388	Timeout	Timeout
6	926	14176	Timeout
7	411	19345	29995
8	309	Timeout	Timeout
9	1053	Timeout	Timeout
10	477	Timeout	Timeout
μ	8411.3	13052.67	29995
σ	16712.51	7099.53	9485.25

Table C.7.: Runtime for the incubed IT assignment implementations at test scenario 7

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	16475	Timeout	Timeout
2	1116	Timeout	Timeout
3	1250	Timeout	Timeout
4	1065	Timeout	Timeout
5	1139	Timeout	Timeout
6	804	Timeout	Timeout
7	1004	Timeout	Timeout
8	1205	Timeout	Timeout
9	1386	Timeout	Timeout
10	284	Timeout	Timeout
μ	2572.8	/	/
σ	4894.22	/	/

Table C.8.: Runtime for the incubed IT assignment implementations at test scenario 8

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	2278	Timeout	Timeout
2	2224	Timeout	Timeout
3	2674	Timeout	Timeout
4	2302	Timeout	Timeout
5	1589	Timeout	Timeout
6	1311	Timeout	Timeout
7	899	Timeout	Timeout
8	785	Timeout	Timeout
9	398	Timeout	Timeout
10	510	Timeout	Timeout
μ	1497	/	/
σ	834.1	/	/

Table C.9.: Runtime for the incubed IT assignment implementations at test scenario 9

C.2. Evaluation Results for the incubed IT Assignment at Test Setup 2

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	419	97	47
2	273	88	62
3	241	74	47
4	502	112	78
5	358	160	62
6	201	95	62
7	423	99	62
8	372	270	62
9	190	79	62
10	165	142	63
μ	314.4	121.6	60.7
σ	115.9	58.73	8.76

Table C.10.: Runtime for the incubed IT assignment implementations at test scenario 1

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	665	101	109
2	566	84	109
3	281	105	94
4	319	100	94
5	214	345	172
6	243	91	172
7	231	232	125
8	560	137	94
9	437	88	78
10	194	140	125
μ	371	142.3	117.2
σ	172.34	83.69	32.34

Table C.11.: Runtime for the incubed IT assignment implementations at test scenario 2

Test Run	Java Implementation [ms]	ASP within Java Implementation [ms]	Standalone ASP [ms]
1	646	78	93
2	390	81	109
3	477	199	109
4	311	100	109
5	193	280	78
6	247	158	94
7	481	129	78
8	219	351	109
9	203	126	78
10	214	81	109
μ	338.1	158.3	96.6
σ	154.61	92.68	14.25

Table C.12.: Runtime for the incubed IT assignment implementations at test scenario 3

Bibliography

[1] Michael Abseher, Martin Gebser, Nysret Musliu, Torsten Schaub, and Stefan Woltran. “Shift Design with Answer Set Programming.” In: *Fundamenta Informaticae* 1 (2016). Inclezan, Daniela (Editor) Maratea, Marco (Editor) Marek, Victor (Editor), pp. 32–39. ISSN: 01692968. DOI: 10.1007/978-3-319-23264-5_4 (cit. on pp. 41, 64).

[2] Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. “Advances in WASP.” In: *Logic Programming and Nonmonotonic Reasoning*. Ed. by Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski. Vol. 9345. Lecture Notes in Computer Science 9345. Cham: Springer International Publishing, 2015, pp. 40–54. DOI: 10.1007/978-3-319-23264-5_5 (cit. on p. 24).

[3] Hans-Jürgen Appelrath, Volker Claus, Günter Hotz, Lutz Richter, Wolffried Stucky, Klaus Waldschmidt, and Peter Thiemann. *Grundlagen der funktionalen Programmierung*. Wiesbaden: Vieweg+Teubner Verlag, 1994. 348 pp. DOI: 10.1007/978-3-322-89207-2 (cit. on p. 17).

[4] Mutsunori Banbara, Benjamin Kaufmann, Max Ostrowski, and Torsten Schaub. “Clingcon: The next generation.” In: *Theory and Practice of Logic Programming* 17.04 (2017), pp. 408–461. ISSN: 03879539. DOI: 10.1017/S1471068417000138 (cit. on p. 26).

[5] Chitta Baral. “Knowledge Representation, Reasoning and Declarative Problem Solving.” In: *Knowledge Representation, Reasoning and Declarative Problem Solving* (2003). DOI: 10.1017/CBO9780511543357 (cit. on pp. 15, 17, 19, 20, 28–30).

[6] Bayerische Motoren Werke Aktiengesellschaft, ed. *Annual Report 2018. #Milestones in Future Mobility*. 2018. URL: https://www.press.bmwgroup.com/global/article/attachment/T0293372EN/426960 (visited on 06/14/2019) (cit. on p. 48).

[7] D. P. Bertsekas. "The auction algorithm: A distributed relaxation method for the assignment problem." In: *Annals of Operations Research* 14.1 (1988). PII: BF02186476, pp. 105–123. ISSN: 0254-5330. DOI: 10.1007/BF02186476 (cit. on pp. 5, 8, 10).

[8] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. "Answer set programming at a glance." In: *Communications of the ACM* 54.12 (2011), p. 92. ISSN: 00010782. DOI: 10.1145/2043174.2043195 (cit. on pp. 18–20, 40, 65, 118).

[9] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. "Assignment Problems." In: *Advances in Soft Computing* 54 (2012). DOI: 10.1137/1.9781611972238 (cit. on pp. 5–7, 9–11).

[10] Paula-Andra Busoniu, Johannes Oetsch, Jörg Pührer, Peter Skočovský, and Hans Tompits. "SeaLion: An eclipse-based IDE for answer-set programming with advanced debugging support." In: *Theory and Practice of Logic Programming* 13.4-5 (2013), pp. 657–673. ISSN: 03879539. DOI: 10.1017/S1471068413000410 (cit. on p. 37).

[11] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. *ASP-Core-2 Input Language Format*. 2012 (cit. on pp. 22, 27).

[12] Francesco Calimeri, Davide Fuscà, Simona Perri, and Jessica Zangari. "I-DLV: The New Intelligent Grounder of DLV." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10037.June (2016), pp. 192–207. ISSN: 97833194. DOI: 10.1007/978-3-319-49130-1_15 (cit. on pp. 21, 23).

[13] *Clingo C API*. 20.06.2019. URL: https://potassco.org/clingo/c-api/current/ (cit. on p. 37).

[14] *Clingo Python API*. 20.06.2019. URL: https://potassco.org/clingo/python-api/current/clingo.html (cit. on p. 37).

[15] Brian Coltin and Manuela Veloso. "Optimizing for Transfers in a Multi-vehicle Collection and Delivery Problem." In: *Distributed Autonomous Robotic Systems*. Ed. by M. Ani Hsieh and Gregory Chirikjian. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 91–103. ISBN: 978-3-642-55146-8 (cit. on p. 39).

[16] Carmine Dodaro and Marco Maratea. “Nurse Scheduling via Answer Set Programming.” In: *Logic Programming and Nonmonotonic Reasoning*. Ed. by Marcello Balduccini and Tomi Janhunen. Cham: Springer International Publishing, 2017, pp. 301–307. ISBN: 978-3-319-61660-5 (cit. on pp. 40, 41).

[17] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner, eds. *Answer Set Programming: A Primer*. Vol. 5689. 2009. DOI: 10.1007/978-3-642-03754-2_2 (cit. on pp. 21–23, 27, 30).

[18] Esra Erdem, Erdi Aker, and Volkan Patoglu. “Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution.” In: *Intelligent Service Robotics* 5.4 (2012). PII: 119, pp. 275–291. ISSN: 1861-2776. DOI: 10.1007/s11370-012-0119-x (cit. on pp. 41, 64).

[19] Wolfgang Faber, Nicola Leone, and Francesco Ricca. “Answer Set Programming.” In: *Reasoning Web. Semantic Technologies for Intelligent Data Access*. Ed. by Sebastian Rudolph, Georg Gottlob, Ian Horrocks, and Frank van Harmelen. Vol. 9. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, p. 365. ISBN: 978-3-642-39783-7. DOI: 10.1002/9780470050118.ecse226 (cit. on p. 41).

[20] Andreas Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, and Erich C. Teppan. “Industrial Applications of Answer Set Programming.” In: *KI - Künstliche Intelligenz* 32.2-3 (2018), pp. 165–176. ISSN: 0933-1875. DOI: 10.1007/s13218-018-0548-6 (cit. on pp. 47, 65, 67, 89, 143).

[21] Onofrio Febbraro, Giovanni Grasso, Nicola Leone, and Francesco Ricca. “JASP: A Framework for Integrating Answer Set Programming with Java.” In: *Proc. of 13th International Conference on Principles of Knowledge Representation and Reasoning (KR2102)* (2012) (cit. on p. 38).

[22] Onofrio Febbraro, Kristian Reale, and Francesco Ricca. “ASPIDE: Integrated Development Environment for Answer Set Programming.” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6645 (2011), pp. 317–330. ISSN: 97833194. DOI: 10.1007/978-3-642-20895-9_37 (cit. on p. 37).

[23] Maribel Fernández. *Programming Languages and Operational Semantics*. London: Springer London, 2014. 211 pp. ISBN: 978-1-4471-6367-1. DOI: 10.1007/978-1-4471-6368-8 (cit. on pp. 15, 16).

[24] Merrill M. Flood. "The traveling-salesman problem." In: *Operations research for management* 2 (1956), pp. 340–357 (cit. on p. 8).

[25] Davide Fuscà, Francesco Calimeri, Jessica Zangari, Simona Perri, and Carmine Dodaro. "Efficiently Coupling the I-DLV Grounder with ASP Solvers." In: *Theory and Practice of Logic Programming* 9203 (2018), pp. 1–20. ISSN: 03879539. DOI: 10.1017/S1471068418000546 (cit. on pp. 22–24).

[26] Michail G. Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton Kleywegt, Sven Koenig, Craig Tovey, Adam Meyerson, and Sonal Jain. "Auction-Based Multi-Robot Routing." In: *Science and systems 1*. Ed. by Sebastian Thrun, Gaurav Suhas Sukhatme, and Stefan Schaal. MIT Press, 2005. ISBN: 978-0-26270-114-3. DOI: 10.15607/RSS.2005.I.045 (cit. on p. 39).

[27] Maurizio Gabbrielli and Simone Martini. *Programming Languages: Principles and Paradigms*. London: Springer, 2010. 450 pp. ISBN: 978-1-84882-913-8. DOI: 10.1007/978-1-84882-914-5 (cit. on p. 17).

[28] Martin Gebser, Amelia Harrison, Roland Kaminski, Vladimir Lifschitz, and Torsten Schaub. "Abstract gringo." In: *Theory and Practice of Logic Programming* 15.4-5 (2015), pp. 449–463. ISSN: 03879539. DOI: 10.1017/S1471068415000150 (cit. on pp. 21, 27).

[29] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, Sven Thiele, and Philipp Wanko. *POTASSCO User Guide. Second edition, version 2.2.0*. 2019. URL: https://github.com/potassco/guide/releases/download/v2.2.0/guide.pdf (cit. on pp. 25–27, 31–33, 35, 36).

[30] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Patrick Lühne, Philipp Obermeier, Max Ostrowski, Javier Romero, Torsten Schaub, Sebastian Schellhorn, and Philipp Wanko. "The Potsdam Answer Set Solving Collection 5.0." In: *AI Magazine* 32.2-3 (2018), pp. 181–182. ISSN: 01234567. DOI: 10.1007/s13218-018-0528-x (cit. on p. 21).

[31] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. "Answer Set Solving in Practice." In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6.3 (2012), pp. 1–238. ISSN: 97816084. DOI: 10.2200/S00457ED1V01Y201211AIM019 (cit. on pp. 15, 16, 18–22, 24, 26–31, 33, 34, 36, 99).

[32] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. "Multi-Criteria Optimization in Answer Set Programming." In: *ICLP (Technical Communications)*. Vol. 11. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011, pp. 1–10 (cit. on p. 25).

[33] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. "Potassco: The Potsdam answer set solving collection." In: *AI Communications* (2011). DOI: 10.3233/AIC-2011-0491 (cit. on pp. 24, 25).

[34] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. "clasp: A Conflict-Driven Answer Set Solver." In: *Logic Programming and Nonmonotonic Reasoning*. Ed. by Chitta Baral, Gerhard Brewka, and John Schlipf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 260–265. ISBN: 978-3-540-72200-7 (cit. on p. 24).

[35] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. "Conflict-driven answer set solving: From theory to practice." In: *Artificial Intelligence* 187-188 (2012), pp. 52–89. DOI: 10.1016/j.artint.2012.04.001 (cit. on p. 24).

[36] Martin Gebser, Nicola Leone, Marco Maratea, Simona Perri, Francesco Ricca, and Torsten Schaub. "Evaluation Techniques and Systems for Answer Set Programming: A Survey." In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI'18. AAAI Press, 2018, pp. 5450–5456. ISBN: 978-0-9992411-2-7 (cit. on pp. 21, 23, 24).

[37] Martin Gebser, Philipp Obermeier, Thomas Otto, Torsten Schaub, Orkunt Sabuncu, van Nguyen, and Tran Cao Son. "Experimenting with robotic intra-logistics domains." In: *Theory and Practice of Logic Programming* 18.3-4 (2018), pp. 502–519. ISSN: 03879539. DOI: 10.1017/S1471068418000200 (cit. on p. 44).

[38] Martin Gebser, Philipp Obermeier, Torsten Schaub, Michel Ratsch-Heitmann, and Mario Runge. "Routing Driverless Transport Vehicles in Car Assembly with Answer Set Programming." In: *Theory and Practice of Logic Programming* 18.3-4 (2018), pp. 520–534. ISSN: 03879539. DOI: 10.1017/S1471068418000182 (cit. on p. 43).

[39] Michael Gelfond and Vladimir Lifschitz. "The Stable Model Semantics for Logic Programming." In: *Logic programming. Proceedings of the Fifth International Conference and Symposium*. Ed. by Robert A. Kowalski. MIT press series in logic programming. International Conference on Logic Programming et al. Cambridge, Mass.: MIT Press, 1988, pp. 1070–1080. ISBN: 0-262-61056-6 (cit. on pp. 28–30).

[40] Brian P. Gerkey and Maja J. Matarić. "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems." In: *The International Journal of Robotics Research* 23.9 (2004), pp. 939–954. DOI: 10.1177/0278364904045564 (cit. on pp. 12–14).

[41] Eugene Goldberg and Yakov Novikov. "BerkMin: A fast and robust Sat-solver." In: *Discrete Applied Mathematics* 155.12 (2007), pp. 1549–1561. ISSN: 0166218X. DOI: 10.1016/j.dam.2006.10.007 (cit. on p. 25).

[42] Youssef Hamadi and Lakhdar Sais. *Handbook of Parallel Constraint Reasoning*. Cham: Springer International Publishing, 2018. 681 pp. ISBN: 978-3-319-63515-6. DOI: 10.1007/978-3-319-63516-3 (cit. on pp. 22, 23).

[43] Roland Kaminski, Torsten Schaub, and Philipp Wanko. "A Tutorial on Hybrid Answer Set Solving with clingo." In: *Reasoning Web*. Vol. 10370. Lecture Notes in Computer Science. Springer, 2017, pp. 167–203. ISBN: 978-3-319-61032-0 (cit. on pp. 17, 26).

[44] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. "A comprehensive taxonomy for multi-robot task allocation." In: *The International Journal of Robotics Research* 32.12 (2013), pp. 1495–1512. DOI: 10.1177/0278364913496484 (cit. on pp. 13, 14).

[45] H. W. Kuhn. "The Hungarian method for the assignment problem." In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97. DOI: 10.1002/nav.3800020109 (cit. on pp. 5, 8).

[46] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. "The DLV system for knowledge representation and reasoning." In: *ACM Transactions on Computational Logic* 7.3 (2006), pp. 499–562. ISSN: 15293785. DOI: 10.1145/1149114.1149117 (cit. on p. 21).

[47] Nicola Leone and Francesco Ricca, eds. *Answer Set Programming: A Tour from the Basics to Advanced Development Tools and Industrial Applications*. Vol. 9203. 2015. DOI: 10.1007/978-3-319-21768-0_10 (cit. on p. 41).

[48] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. "Chaff. Engineering an efficient SAT solver." In: (2002), pp. 530–535. DOI: 10.1145/378239.379017. (Visited on 06/17/2019) (cit. on pp. 24, 25).

[49] James Munkres. "Algorithms for the Assignment and Transportation Problems." In: *Journal of the Society for Industrial and Applied Mathematics* 5.1 (1957), pp. 32–38. ISSN: 0368-4245. DOI: 10.1137/0105003 (cit. on p. 7).

[50] Van Nguyen, Philipp Obermeier, Tran Cao Son, Torsten Schaub, and William Yeoh. "Generalized Target Assignment and Path Finding Using Answer Set Programming." In: *IJCAI International Joint Conference on Artificial Intelligence* (2017), pp. 1216–1223. ISSN: 97809992. DOI: 10.24963/ijcai.2017/169 (cit. on pp. 43, 63, 64).

[51] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. "A taxonomy for task allocation problems with temporal and ordering constraints." In: *Robotics and Autonomous Systems* 90 (2017). PII: S0921889016306157, pp. 55–70. ISSN: 0921-8890. DOI: 10.1016/j.robot.2016.10.008 (cit. on p. 39).

[52] Francesco Ricca, Giovanni Grasso, V. LIO, and S. Iiritano. "Team-building with answer set programming in the Gioia-Tauro seaport." In: *Theory and Practice of Logic Programming* 12.03 (2012), pp. 361–381. ISSN: 03879539. DOI: 10.1017/S147106841100007X (cit. on pp. 40, 41).

[53] Zeynep G. Saribatur, Esra Erdem, and Volkan Patoglu, eds. *Cognitive factories with multiple teams of heterogeneous robots: Hybrid reasoning for optimal feasible global plans*. 2014. DOI: 10.1109/IROS.2014.6942965 (cit. on pp. 43, 91).

[54] S. Schieweck, G. Kern-Isberner, and M. ten Hompel. "Using answer set programming in an order-picking system with cellular transport vehicles." In: *IEEE International Conference on Industrial Engineering and Engineering Management* (2016), pp. 1600–1604. ISSN: 97815090. DOI: 10.1109/IEEM.2016.7798147 (cit. on pp. 42, 43).

[55] Patrik Simons, Ilkka Niemelä, and Timo Soininen. "Extending and implementing the stable model semantics." In: *Artificial Intelligence* 138.1-2 (2002), pp. 181–234. DOI: 10.1016/S0004-3702(02)00187-X (cit. on p. 25).

[56] Tommi Syrjänen. *Lparse 1.0 User's Manual*. URL: www.tcs.hut.fi/Software/smodels/lparse.ps (cit. on p. 21).

[57] Tommi Syrjänen. "Omega-Restricted Logic Programs." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2173 (2001), pp. 267–280. ISSN: 97833194. DOI: 10.1007/3-540-45402-0_20 (cit. on p. 21).

[58] R. Trejo, J. Galloway, C. Sachar, V. Kreinovich, Chitta Baral, and Chi Le TUAN. "From Planning to Searching for the Shortest Plan: An Optimal Transition." In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 09.06 (2001), pp. 827–837. ISSN: 0218-4885. DOI: 10.1142/s0218488501001277 (cit. on p. 44).

[59] Christian Wagenknecht. *Programmierparadigmen*. Wiesbaden: Springer, 2016. 249 pp. DOI: 10.1007/978-3-658-14134-9 (cit. on pp. 16, 17).

[60] Zachary B. Rubinstein, Stephen F. Smith, and Laura Barbulescu. "Incremental Management of Oversubscribed Vehicle Schedules in Dynamic Dial-A-Ride Problems." In: *Proceedings of the National Conference on Artificial Intelligence* 3 (2012) (cit. on p. 40).